Architectures: Objects, Middleware & Components

MTS: Fast train coming?

August 1998/By Max Dolgicer

Today's I/S executives face yet another crucial decision -- selecting new transactional technology in a quickly changing market. These executives are standing in the path of runaway trains carrying products, buzzwords and technological choices. Steaming along the track into yet another category of middleware -- Object Transaction Monitors (OTMs) - is Microsoft Corp. with its Microsoft Transaction Server (MTS). Development managers must now decide whether to quickly board the MTS train or wait and assess the other OTMs expected to arrive later (see "An OTM primer"). However, stepping aside of the MTS train means running the risk of being left out and labeled indecisive. Even worse, corporate competitors who choose to use the new technology could develop a killer application that becomes a strategic weapon. But even those that get on the train face a bumpy ride. The difference between a good and a bad product choice can mean having a job or looking for one.

The selection of development, management and deployment tools for multitier distributed applications is especially tough, because the crucial decisions revolve around middleware. In fact, the selection of a middleware platform (yes, it is a platform) may be more important than the selection of a middle-tier operating system like Unix or Windows NT.

Development units need to decide between object middleware -- Corba and Microsoft MTS/Component Object Model (COM) -- where the world of objects and components becomes reality, and more traditional middleware (procedural).

An object middleware choice, for example, can cause in-house development pains. This would likely lead to the eventual hiring of consultants to develop value-added features to the base middleware product in order to provide scalability, stability and performance capabilities. Yet even traditional middleware has its problems. Procedural middleware has a lower level of abstraction -- it is definitely not object-oriented or even object-based. These technologies also need to reconcile varying product support for synchronous and asynchronous behavior, another big problem.

So, do corporate developers choose middleware that offers scalability, stability, performance and almost no risk of failure due to the stability of the many products in this category, or do they stick with middleware that offers object and component models but is not yet ready for prime time?

The arrival of OTMs to the middleware scene might reduce a lot of the worries associated with this decision. Although it might sound too good to be true, true believers in object middleware will not have to agonize over the choice of object vs. traditional middleware. For many, the decision will simply be which OTM product to endorse and use. This essay gives users and potential users alike, the inside track on the technical and business pros and cons of MTS.

Welcome to MTS

MTS is a distributed runtime environment for COM objects that began shipping approximately 18 months ago. It provides a sophisticated infrastructure for activating and running objects across the network. MTS provides automatic transaction management, database connection pooling, process isolation, automatic thread-pooling, automatic object instance management, resource sharing, role-based security, transaction monitoring within
distributed applications and much more. These services are necessary for scaling server-side components and supporting a substantial number of concurrent client requests. MTS performs all of these services automatically, and without the need for application developers to write special code. A developer can therefore develop server-side components with a single client in mind.

What if all COM-based servers were required to handle only one client at a time? Instead of having to share databases with other servers, they could run in a single thread without worrying about security authorization, object management and transaction support across multiple databases. They also would not have to worry about holding database locks. In case of failures, they could start afresh without even considering lost data. However, this is not feasible for real, mission-critical enterprise applications.

Microsoft has already captured the desktop market. To conquer enterprise territory, the firm added MTS and many other middleware products, such as MSMQ, to its BackOffice family in an attempt to capture the more interesting and highly competitive application server side (for example, middle tier) of distributed applications.

Version 1.0 of MTS revolutionized the way developers built server-centric applications by providing a true component-oriented runtime that did not require the writing of infrastructure code. Version 2.0 then extended this environment by marrying it to the Microsoft Internet Information Server (IIS 4.0), integrating it with MSMQ 1.0, adding transactional connectivity to Oracle databases, and providing transactional mainframe connectivity through Microsoft SNA Server 4.0. Failover protection is the responsibility of Windows NT clustering services (Wolfpack). All of these products are bundled with Windows NT Server.

MTS and three-tier architectures

But where does MTS fit in an architecture consisting of three or more tiers? From the perspective of a multitier architecture, the business components are implemented separately from the presentation and data components in order to enhance the scalability, reusability and robustness of the solution. Most middleware vendors are therefore turning their attention to the middle tier, which runs business logic packaged as software components. Instances of these components will manage requests coming in from the client tier (including Internet browser clients), and will update back-end databases on the data tier. While it sounds simple in theory, it is quite complex in practice.

For example, components will need valuable resources, such as database connections. Because the number of clients using the system can far exceed the number of component instances and database connections, database connections are a limited resource. Efficient management and reuse of these resources is therefore the key to scalability and high throughput. The single most important premise of OTM technology in general, and MTS in particular, is that management and reuse of such resources can be handled by middleware without imposing special coding requirements on the business application developer.

Ideally, developers will implement COM components with the single user in mind, and focus on fulfilling the business requirements. These components can then be deployed in an MTS environment to handle complex, multi-user problems.

Pooling and recycling for scalability

Configuring, maintaining and making a connection at runtime to a database for each individual client is a very expensive operation. A connection at the server side requires a great deal of memory and processor resources. When the number of clients increases, numerous server resources are required to maintain these client connections. This scenario not only limits scalability, it requires a significant programming effort.

MTS Resource Dispenser automati-cally organizes database connections in a reusable pool known as a connection pool. Once opened, Resource Dispenser caches these connections and reuses them for other requests. Because the connections do not have to be opened and closed with every request, connection pooling enhances performance to a satisfactory level.

Another big hurdle to scalability in a multiclient environment is concurrent access to data. For example, suppose a client locks a data record in order to modify it. Until it is unlocked, another operation cannot access this chunk of data. If another process is waiting for the locked resource, this can eventually result in performance degradation. The goal is to therefore minimize this locking period. MTS provides Shared Property Manager, a special resource dispenser that manages locks and concurrency, which enables different components to access the same data with optimal results.

Every time a user hops a process or machine boundary, they must pay the cost of using the unseen resources facilitating the process. For example, to get a service from a COM object, the client first needs to instantiate the COM object, then use the methods and finally destroy the object. In this simple case alone, the client crosses the boundary three times, and "chews up" system time when it creates and destroys the object.

MTS optimizes client throughput by ensuring that multiple clients can use middle-tier component instances. It does this by coordinating these instances so that as soon as one client's work is completed, its instances are freed up to work for another client. This feature or policy is known as "just-in-time activation."

MTS also introduces a Context Wrapper object that sits transparently between a client and the real MTS object at runtime. The Context Wrapper is responsible for creating and destroying an object. This is how it works: MTS deactivates the already instantiated component after the client has used its services and recycles its resources. When the component is requested again, MTS activates it and assigns all of the required resources.

Thread management requires a great deal of synchronization considerations, as well as complex coding. MTS' automatic thread-pooling helps ensure robustness. When MTS receives a request from a client to execute a component, it locates an available thread from a maintained thread pool and, after satisfying the client request, returns the thread back to the pool. This improves performance and reduces the overhead required to create and destroy the thread.

MTS and transaction management

Support for distributed transaction management is provided by MTS automatically. The level of transaction protection can be specified using property pages. In a complex enterprise, multiple databases (possibly on a variety of different servers) can be accessed by components running under MTS control within a single MTS transaction. MTS performs a two-phase commit with each database, as well as with the other resources (such as MSMQ) participating in the transaction. This is achieved with Microsoft Distributed Transaction Coordinator (DTC), which allows transactions to occur through many database servers as discussed above.

DTC supports X/Open's XA protocol, IBM's LU6.2 protocol and Microsoft's Object Linking and Embedding (OLE) transaction protocol for this purpose. Of course, the latter is the native MTS protocol and yields the best performance. One difference between MTS and other transaction management systems is that MTS introduces the notion of declarative rather than programmatic transaction management. Clients of MTS components are not involved in transaction management at all unless they want to be.

MTS objects are created from components that have been installed into an MTS runtime environment. However, MTS applications must be implemented as in-process COM servers, that is, as dynamic link libraries (DLLs). Because a library cannot run on its own, it requires some sort of container within which the DLL can execute. MTS provides this container.

COM objects used with MTS are often referred to as ActiveX components. A group of ActiveX components that all run in the same process comprise an MTS package. A single machine may have several different packages running at the same time, each in its own process with its own copy of the MTS Executive. Every MTS component must therefore be deployed within the context of an MTS package. As noted earlier, the MTS approach to transaction management is quite unique. In a traditional transaction system, the client programmatically indicates when a transaction starts and terminates. Although this approach is not a preferred one, it is supported by MTS. The MTS preferred approach is declarative transaction management, where the client remains unaware of when transactions start and terminate.

MTS accomplishes automatic and transparent transaction support using a Context Object. If a transaction needs to span the functions of two different objects, MTS simply uses the same Context Object to coordinate the activities of both objects. This allows the MTS transaction management logic to be separate from the application logic.

Transaction semantics are defined declaratively when ActiveX components (for example, application objects) are assembled into an application package. An MTS application package is actually a group of related ActiveX components. In order to create an application package, the relationships between the individual components must be configured using MTS Explorer. As each component is configured, a transaction attribute is assigned to it. There are four possible transaction settings:

- "Requires a transaction." This means that a component will always be created inside the context of a transaction. The component will inherit the transaction of its creator if one exists. If the creator is not running within a transaction, a new one is created.
- "Requires a new transaction." The component will always be created inside a new transaction.
- "Support transactions." The component will inherit the transaction of its creator if one exists. If the creator is not running in a transaction, the component will be created without a transaction.
- "Does not support transaction." The component will never be created inside the context of an MTS

transaction.

Every MTS object is therefore created either inside a transaction or without it, and is managed by MTS based on one of these four settings. It should be noted that an MTS object can be created either by an MTS client or by another MTS object. MTS is also the first OTM that uses the declarative rather then programmatic transaction management approach, thereby eliminating the need to write code against transactioning APIs.

Applications may have a number of components. To make an application more manageable and flexible, MTS puts the individual components inside component packages, a collection of ActiveX components that execute in the same process space and share common security. A security check is done only when accessing a package. Therefore, in a simple scenario, one thread or process failure may affect the entire application. However, because MTS isolates processes in a package, which runs in its own process space to prevent error cascading, the failure of one package does not impact the state of another package.

Integration with the Web and back-end legacy apps

There are two alternatives to accessing MTS components from the Web. The first alternative is using the COM/ DCOM protocol, as well as ActiveX controls and Active Server Pages (ASP), to allow Web clients to have full access to MTS components. Microsoft's Explorer and Netscape's Navigator are also supported. Because an ActiveX control is just an in-process COM object, it is a straightforward process to download ActiveX controls to communicate with a remote object via COM/DCOM.

The second alternative for browser-based access to an MTS component relies only on the traditional HTTP protocol -- no DCOM packets flow across the network. Instead, the browser accesses what appears to the user to be an ordinary Web page, one that is identified by an ordinary URL. Using ASP and the latest integration features in IIS 4.0 and MTS 2.0, users can build data-entry applications that can be accessed from standard HTML-based clients.

The Web has emerged as a potential source for delivering business information across lines of business and enterprises. For Web applications, the integration of IIS 4.0 and MTS 2.0 makes it easy to call MTS components that access databases, mainframe applications and so on. At the same time, MTS process isolation benefits IIS by providing a robust Web site.

To be successful, any given middleware technology must address interoperability with legacy applications. While up until recently interoperability with Unix-based applications was never a priority for Microsoft, this was not the case for mainframe-based applications. In fact, Microsoft has always been and still is very serious about interoperability with mainframe applications.

Using some of the technology it had previously acquired (remember TransAccess from Netwise?), Microsoft developed the COM Transaction Integrator (COMTI, formerly known as Cedar) for CICS and IMS to wrap legacy CICS and IMS transactions so that they appear to be typical MTS components. Today COMTI is shipped as part of Microsoft’s NT Server.

COMTI has the ability to integrate COM-based applications with legacy transaction processing systems, and can make mainframe applications appear as COM components to PC clients. COMTI provides the means of communication between application servers running NT and mainframe applications, as well as the development capabilities to "objectize" and "wrap" CICS and IMS transactions so that they appear to be Automation objects to COM applications. Using COMTI, any Web-based client (or Visual Basic client) is capable of accessing an MTS component, which can communicate with CICS or IMS transactions to access data "locked" and controlled by these two TP monitors.

COMTI has three major components -- a Component Builder, a Management Console and the COMTI Run Time Proxy. The Component Builder tool is used to convert a COM type library to the Cobol declarations (such as copy-boxes) for CICS and IMS applications, and to generate a type library from the Cobol declarations of CICS or IMS applications. COMTI also provides automatic mapping of data types when the sending and receiving programs are written in different languages. For example, it will automatically map Visual Basic data types (or the data types of any other language to which the COM component is written) to Cobol data types and vice versa.

The Management Console is used for administration pur-poses, while the Run Time Proxy provides a runtime environment (for example, a container) for the COM components created by the Component Builder.

As a side note, MTS can be configured to be fault-tolerant with automatic failover by using Windows NT Server clustering. In case of failure, Microsoft Cluster Server (MSCS) automatically transfers control to the standby machine.

MTS inherits Windows NT security and provides it in two ways -- declarative and programmatic security. Declarative security is a type of traditional authentication. This is specified when the MTS administrator adds the MTS components into the packages. The administrator declares who can access a particular package and what would be the required level of authentication.

In order to be flexible and to be able to integrate tightly with the existing Windows NT security system, MTS also uses the concept of roles. A role is a user-defined grouping of Windows NT users or groups that MTS components can use to maintain security.

Programmatic security is engineered to the components at development time by assigning roles to the components. For example, a broker and a shareholder represent two roles. The developer can code these business roles into the components and, depending upon the behavior of the components, the administrator can associate users with the correct roles using MTS Explorer. In this way, components can decide at runtime how to proceed with the operation depending upon who has activated the request. This is also known as role-based security.

Development tools, interoperability

MTS is COM-based. As discussed earlier in this article, MTS provides the runtime infrastructure for COM components, which allows developers to scale desktop applications to enterprise level. From a developer’s perspective, the COM component must meet some requirements to be classified as an MTS component. For example, components should be self-registering 32-bit DLL-based COM server (In-Proc serrver)-compatible. MTS component interfaces must also be either standard marshaled or Automation-compatible.

Any tool that can develop a COM DLL can produce an MTS component. For example, Microsoft Visual Basic, Visual C++, Visual J++ and vendor tools, such as Inprise’s Delphi and Powersoft's PowerBuilder (to name a few), can be used.

Customers always want a solution that can easily plug into another solution. Microsoft is therefore working to make COM available on Unix and other non-Microsoft operating systems. The firm already has some friends. Dublin, Ireland-based Iona Technologies, for example, has licensed COM to provide interoperability with Corba-based systems and plans to integrate its technology with MTS.

Software AG, Darmstadt, Germany, released COM for a variety of Unix platforms and is planning to release it on OS/390. Digital Equipment Corp./Compaq Computer, Maynard, Mass., is incorporating the COM technologies as a means of providing its customers with complete Windows NT integration solutions. And Silicon Graphics Inc., Mountain View, Calif., has licensed COM technology and plans to incorporate COM into a future release of its Irix operating system. So, in theory, COM is no longer simply a Wintel technology.

However, in practice this is really not the case. COM's strength is that it is the only middleware technology on the market that is tightly integrated with a specific operating system -- NT. It is also supported by thousands of ISVs who are developing vertical and horizontal applications for NT, thus building an industry around COM. Outside the Wintel operating system, however, COM does not have these advantages. Although Microsoft and its partners are attempting to show COM as a cross-platform technology, for all practical purposes, it remains Wintel technology.

When it comes to MTS availability outside NT, Microsoft does not attempt to confuse anyone. For now, there are no plans to port MTS to any other operating system. MTS is one of the weapons that Microsoft is intending to use in its determination to convince corporations that NT is the platform of choice for server-centric applications. Microsoft is simply planning to spend whatever it takes to get MTS right.

MTS today, COM tomorrow

We can already see the impact of MTS on the development of multitier enterprise solutions. After capturing the desktop, Microsoft is now prepared to replicate its success on the server; MTS is one of the major technologies to make this happen. MTS is a successful technology for developing multitier applications at a commodity price, and its future is quite predictable. Microsoft has even spelled out its plans: COM+.

COM+ is where separate products, such as MTS, COM, MSMQ and MSCS, come together under a single umbrella. With its support for universal clients, and integration with technologies such as IIS and COMTI, COM+ should position Microsoft as a strong middleware provider for middle-tier applications. The fact that COM+ will be bundled with Windows NT 5.0, guaranteeing middleware technology for a commodity price, clearly enhances Microsoft's chances in its endless pursuit to capture the server market for multitier enterprise applications.

(508) 652-1100 FAX (508) 652-1210 e-mail:general@spgnet.com http://www.adtmag.com