Applications: Objects, Components and Middleware

Battling MOMs

March 1998/By Max Dolgicer

There were a number ofmessage-oriented middleware (MOM) products on the market before IBM's release of MQSeries, but IBM's arrival truly validated the market and provided the fuel engine required for growth. Now the world of MOMs is on the verge of further change as Microsoft brings forth its MSMQ product.

Mission-critical enterprise applications are still new territory for Microsoft, and MSMQ does not support a number of features supported by MQSeries, but from a features/functions and robustness point of view, Microsoft's MSMQ clearly can challenge IBM. MSMQ appears less complex to administer, but MQSeries, for its part, supports far more operating system heterogeneity.

The MOM market encompasses many products and vendors. This market includes pioneers like PeerLogic and Momentum Software as well as late comers like Talarian, just to name a few. It includes companies selling products spun out from home-grown message queuing efforts developed on Wall Street, as well as VCOM from Verimation, which arose from automaker Volvo. However, at the end of the day, IBM's MQSeries is the undisputed market leader. Microsoft is trying to change this.

Microsoft's MSMQ is a worthy competitor to IBM's MQSeries middleware if for no other reason than it comes from Microsoft, and the company has substantial distribution channels. In a number of important technical areas, it is even superior to IBM's MQSeries.

Bundled with the OS

Microsoft has released MSMQ as part of the NT 4.0 Service Pack 3. There, it is bundled with the Microsoft Transaction Server (MTS). Microsoft is now beginning to ship this to channels. It is the first company in the middleware space that bundles two premier middleware products with the operating system, thus completely redefining the pricing and statistics models used to indicate and measure the success of any middleware product. Up until now, the success was measured by the number of licenses sold and the revenues derived from these licenses. As Microsoft ships MTS and MSMQ with the NT operating system, it is safe to say that MSMQ will quickly pass MQSeries in terms of the number of licenses sold.

In terms of revenue, the picture is more clouded. IBM's just-released MQSeries V5 pricing model is more complex and expensive than previous MQSeries releases. For its part, Microsoft is bundling MSMQ with two variations of the NT Server, Enterprise Edition (EE) and Standard Edition (SE). In order to compare apples with apples in terms of pricing, an example of a real world application development and deployment needs to be considered. Without going into the details of such an exercise, we predict that a total cost of ownership (TCO) study will show that MSMQ has a pricing advantage over MQSeries.

In technological terms, apples and apples are compared since both products are clear-cut message queuing products. But there are issues to keep in mind. MSMQ is in its first release while MQSeries has been on the market for over five years. Surprisingly, Microsoft's product appears quite complete and stable for the first release. IBM's extensive experience in this market cannot be ignored.

Clearly, while developing MSMQ, Microsoft had an established product to which it could refer. In contrast, with the exception of DECmessageQ (now owned by BEA Systems), IBM had no commercial-strength products on the market to refer to while designing MQSeries. Furthermore, IBM had to labor hard to create a viable market by increasing the awareness of corporate developers and corporate decision-makers about message queuing and its uses. Microsoft does not have to deal with any of these issues. The company, however, has a substantially different problem to deal with.

MSMQ as well as MTS are both positioned as keys to Microsoft's continuous effort to become a provider for enterprise solutions, just like IBM, HP, Sun and others. But it is a big step to make a transition from selling Microsoft Office with endless upgrades, big profit margins, and where little support and consulting are required, to selling software aimed at mission critical applications running banks, airlines and trading floors. Moreover, Microsoft's MOM product is heavily oriented toward its Windows environments. This is key among several issues to consider when comparing middleware approaches.

Software is a very complex and competitive business. That is what makes it so exciting. Good products do not always ensure success, and bad products do not always fail. Of course all of this is more complicated than it appears. There are many items that corporate decision-makers employ before they write checks to software companies and their distributors. However, technical superiority always makes the decision easier. Our partial evaluation shows that both MQSeries and MSMQ are technically sound ready-for-an-enterprise products. This article captures what in our opinion are essential features of message queuing systems.

Heterogeneity

When it comes to supporting heterogeneous operating systems and communication protocols, IBM has the upper hand. This is one of the few features where the verdict is decisive. Message queuing is used to integrate applications that in any modern enterprise are running on heterogeneous operating systems and communication protocols, thus IBM's operating system edge is significant. Microsoft's MSMQ is capable of supporting TCP/IP and IPX/SPX while MQSeries also supports SNA LU6.2. But on the operating systems front, MSMQ runs natively on NT (clients and servers) and Win 95 (clients only), while IBM covers more than 20 operating systems, including strong offerings on OS/390.

But to say that Microsoft has completely neglected support for non-Microsoft operating systems is not true. Largely, Microsoft is relying on third party vendor support. Interoperability with MQSeries is offered via the FalconMQ Bridge available from Level 8 Systems. Using the product, MSMQ applications can communicate with MQSeries applications. Note that the bridge is available on the NT Server Enterprise Edition only and is provided at extra cost. For applications that do not require full message queuing support, MSMQ client software is available from Level 8 Systems as well. Using client software, an application can make calls to a MSMQ Server using the MSMQ API. Level 8 Systems also offers connectivity between CICS Transient Data Queue and MSMQ Queue.

Quality of Service

Quality of Service (QoS) is an umbrella name that encompasses many important middleware features. In its basic form, QoS means support for guaranteed and assured message delivery as well as for message prioritization. Both MSMQ and MQSeries support all three features.

Beyond the basics, QoS encompasses additional features like time-to-live, tracking and notification. Both IBM and Microsoft's products support time-to-live (or what is know in the MOM vocabulary as expiration of messages) where applications can be notified when a particular request has expired and where message queuing systems are capable of handling message expiration. Of course, the sending application should be notified when a particular request has expired. This leads to comment on another requirement -- tracking and notification. Both MQSeries and MSMQ are capable of supporting tracking and notification so that an application can be notified when a request is delivered to a remote queue and the target application receives it. In IBM jargon, the features are called Confirm on Arrival (COA) and Confirm on Delivery (COD), respectively. Using these features, a report message (with or without application data) is generated and can be sent to a specific queue. Of course to make any sense out of this message, an application needs to read the message from such a queue and interpret it according to the application-specific design. MSMQ supports COA and COD features, as well as some additional capabilities. While MQSeries supports only positive acknowledgments for COA and COD, MSMQ supports both positive (ACKs) and negative (NACKs) acknowledgments. A report message is generated when the application message does not make it to a queue or has not been read by a receiving application.

An additional feature under the umbrella of Quality of Service is reporting. In order to track messages within the system, reporting is required. However, for message queuing to detect and report on items such as messages expired or messages reaching the remote queue, it is necessary for the message queuing system to generate and to handle different types of meaningful events. This brings to light another feature called instrumentation.

Broadly speaking, both products are instrumented to support different types of events for different situations. Of course, MSMQ generates NT events that are reported to the operating system log and can be easily interpreted by the NT Performance Monitor. Because MQSeries is platform neutral, these events are logged to different event queues for interpretation by system management tools. The different approaches taken by Microsoft and IBM to system management have a very direct effect on the I/S managers' checkbook.

A good example to demonstrate importance of Quality of Service is a money transfer application. An application at a particular bank wants to transfer money to another bank. The sending application could specify a time-to-live of one hour and also request acknowledgment. A positive acknowledgment could automatically generate a confirmation letter to the customer while a negative acknowledgment could trigger an investigation to find out why money transfer failed. It is safe to say that both products cover QoS quite comprehensively.

Both products support location transparency as far as message queues are concerned. However, there is a big difference in how such location transparency is implemented. MQSeries does not support any built-in naming service where the queue manager, channel and queue names -- and their respective attribute information -- can be stored. Such definitions need to be stored locally. For its part, MSMQ comes with a built-in naming service, called the MSMQ directory, which serves as a comprehensive and distributed repository to store important information. MSMQ directory, also called MSMQ Information Server, contains information about queues, including queue locations, an unique ID associated with queues and queue attributes. MSMQ queues can be created, changed, deleted and located by names either administratively with the easy-to-use MSMQ Explorer or programmatically using the MSMQ API calls. The MSMQ directory, which makes it all possible, is real time and fully replicated, thus preventing a single point of failure.

For now, MSMQ is using it own proprietary directory that also requires a version of SQL Server. MSMQ comes with a license limited for support of MSMQ. Most likely, future versions of MSMQ will be using an NT Server Active Directory that will be the future NT naming service.

Queue order/ selecting messages

Both IBM and Microsoft's products support flexible message ordering and message selection. MQSeries, for example, supports queue ordering based on first-in-first-out and message (0 through 9) priority. Similarly, MSMQ supports queue ordering based on message (1 through 7) priority, and on priority established by the time the message was sent relative to messages with similar priority. In addition to flexible queue ordering, it is important the receiving application will have the ability to receive messages using flexible selection criteria. Using MQSeries, messages based on correlation-ID or message-ID can be retrieved. In the releases preceding MSMQ's commercial release, MSMQ also supported message retrieval based on user defined fields. For now, this feature has disappeared, but probably will reappear in future releases of the product.

When it comes to reading messages, multiple methods should be provided. The most important methods are: (a) where a receiving application waits on a queue until the message (any message or specific message which will, of course, require the receiving application to know the unique message identifier) arrives at the queue; (b) where a receiving application polls the queue until a message (again, this could be any message or specific message) arrives at the queue; or (c) where a receiving application registers a callback, which in turn will get executed when a message arrives at the queue. Basically, the first two methods are synchronous calls while the third one is purely asynchronous and event driven. Unfortunately, MQSeries does not support the third method. MSMQ supports all three methods.

Transaction support

As noted in "Message Queuing Middleware: No Longer High Risk for I/S," (see *Application Development Trends*, December 1997, p.33), transaction support has three requirements: (a) support for internal transactions where an application can send and receive messages within a single transaction coordinated by the message queuing engine; (b) transaction support coordinated with database updates where an application program can send (or receive) messages to the queue (or from the queue) and update a database in a single transaction without the requirement of a transaction monitor; and (c) support for external transactions, where queuing operations and updates to a database are coordinated by an external Transaction Processing (TP) monitor. As a side point, it is important to know that when applications use transaction support (internal or external), all messages are delivered exactly once, and in the order in which they were sent. This is another case in which both middleware products support the requirements.

The MQSeries V5 comes with an XA compliant transaction coordinator capable of handling updates to databases and message queues in a single transaction. However, only local transactions are supported, meaning that databases and message queues need to be co-located. Not all databases are included, and at this point, only Oracle and DB2 are supported. Support for additional databases will be forthcoming.

MSMQ on the other hand does not come with a database coordinator. It instead relies on Microsoft's Distributed Transaction Coordinator (MS DTC) to handle transactions that involve database and queue updates. From a transactional support point of view, both products appear to be strong. For now, updates to MSMQ queues can only be coordinated with updates to a SQL Server database. This is more of a MTS (or, to be precise, MS DTC) limitation rather than a limitation of MSMQ itself. MTS 2.0 includes support for additional databases such as Oracle, so it is expected that database limitation will be lifted soon.

Language support

The middleware products from both these industry titans support a variety of programming languages. C and C++ support is taken for granted. MQSeries also supports mainframe-oriented languages such as Cobol. MSMQ relies on a third-party vendor for Cobol support. Both products already support Java. As expected there is a big difference between MQSeries and MSMQ when it comes to ActiveX support.

Some complain that one of MQSeries' inherent problems is its limited support for ActiveX controls. This leads to somewhat cumbersome integration with desktop applications (such as Excel) and desktop application development tools (such as Visual Basic and PowerBuilder). As of now, MQSeries supports Automation only (formerly called OLE Automation).

As expected, MSMQ supports a large number of ActiveX interfaces including those that combine multiple MSMQ functions. Developers can use these ActiveX interfaces and if necessary, use the lower level functions of the native MSMQ API. MSMQ also supports dual interfaces, allowing C++ and tools such as Visual Basic and Delphi access ActiveX interfaces. The bottom line here is that MSMQ is a clear winner when it comes to ActiveX support.

Both products' queue managers are multithreaded. The new release of MQSeries addressed the problem of consistent thread support across multiple operating systems. In its previous versions, the queue manager was multithreaded on some platforms (for example, OS/2 and NT) and single threaded on others (for example, Unix flavors). The lack of thread support created some problems, especially for applications that utilize MQSeries in a client/server model. Using a client/server model, clients have no queuing capabilities. They simply connect to a remote queue manager that is responsible for queuing operations. With the queue manager as a single-threaded process, each client's connection created two processes (one in each direction), thus imposing severe scalability limits. In contrast, the MSMQ engine was multithreaded from inception.

Despite the recent advances in MQSeries architecture, there is still a programmatic difference in how threads are treated. The MQSeries API does not recognize threads. The following example demonstrates such a limitation.

Assume that a program has two threads, one (the writer thread) is responsible for writing messages to the queue, and the other (reader thread) is responsible for reading incoming messages from the queue. In the case of MQSeries, before the writer thread can perform any queuing operations, it must first perform two other operations. First, it must connect to the queue manager to obtain the connection handle and second, it must open the queue to obtain the queue handle. Only after the connection handle and queue handle are obtained can a request message be put on a queue, and processed by another application which in turn will put a reply message on the same queue. Naturally since both threads are using the same queue manager and are sharing the same queue, one would expect that both the connection handle and the queue handle can be passed from the writer thread to the reader thread and the reader thread can obtain the reply simply by reading the messages from the queue. Well, what is natural and logical is not always the way things work.

Since MQSeries does not allow these handles to be shared across threads, the reader thread needs to obtain its own connection handle and queue handle simply by repeating operations already performed by the writer thread. MSMQ does not have such a problem since MSMQ fully recognizes threads at its API level and allows context to be shared across threads.

Naming services

A naming service is a must-have component within any distributed computing infrastructure. One would anticipate that a messaging scheme would have a naming service to store information about queues. Unfortunately, MQSeries does not come bundled with any naming service. This has a negative effect upon configuration, administration and monitoring.

MSMQ, on the contrary, comes bundled with a fully functional distributed and replicated naming service (also called Directory Service or MSMQ Information Server) which is one of the reasons why administration and monitoring of MSMQ rates favorably compared to MQSeries.

Administration, etc.

Imagine an application spanning thousands of bank branches with message queues located at each branch. Each branch also has to communicate with the glass house represented by an

IBM mainframe. Each branch has request queues and reply queues. Quickly, the enterprise is engulfed with thousands of message queues. To define, administer and monitor these queues becomes a monumental task.

Thus, the ability to define and administer these queues from a central location is an important requirement for a message queuing architecture. In fact, many feel strongly that a system management component should be an integral part of any message queuing system. This view is shared by Microsoft. MSMQ comes with an impressive MSMQ Explorer, a GUI-based console capable of supporting most of the essential functionality for managing MSMQ clients and servers from a central console. This is not the case with MQSeries, although the recent release of MQSeries comes with some Web-based administration. MQSeries does not come bundled with easy-to-use management capabilities; it relies on third-party system management tools, which in addition to being expensive, are difficult to use. People who are buying the MQSeries management product quickly discover the bad news -- buying a MQSeries management product can cost more than MQSeries itself. Alternatively, companies can develop home-grown facilities to meet their requirements. All of this is possible since MQSeries is fully instrumented for management and it provides a Programmable Command Format, which is a management API in canonical format. The bottom line is that MSMQ's built-in management capabilities clearly differentiate it from MQSeries.

Client-to-server/server-to-server

MQSeries is implemented in two modes -- clients and servers. Clients are not capable of performing any queuing operations and are dependent on the message queuing server to send and receive messages. Many companies opt for the client/server mode since it involves much less administrative burden (no queues need be administered on the client) and it is certainly cheaper (client pricing ranges from free to very minimal). Despite such clear advantages, client architecture also has a number of distinct disadvantages.

FIRST MQSeries clients are not supported on all platforms. MQSeries does not support MVS clients. This means that an intermediate tier running an MQSeries server capable of supporting MQSeries clients is required. Using such an architecture, MQSeries clients might, for example, communicate with either NT or Unix-based MQSeries servers that, in turn, provide server-to-server communication to MVS.

SECOND all MQI calls are synchronous. Since MQSeries does not support callbacks, clients can only receive messages by either waiting on the arrival of a message to queue or by continuously polling the queue.

THIRD MQSeries clients exhibit less performance than MQSeries servers.

FOURTH network resource utilization is an issue. The server-to-server communication implies session concentration where only connections between queue managers will have to be established. This is not possible to achieve using client-to-server communications where there is no queue manager on the client side.

Although MSMQ uses different terminology, it can support both the client and server architecture. On the client side it supports something called the dependent and independent client. The MSMQ client is an equivalent of the MQSeries client. As the name suggests, it is dependent on the MSMQ server to send and receive messages. MSMQ also supports independent clients, however, that are capable of supporting queuing operations. This raises a question -- what is the difference between MSMQ independent clients and MSMQ servers? There is a substantial difference.

There is no question that the MSMQ Server is the most important part of the MSMQ product. MSMQ server runs exclusively on the Windows NT Server. In addition to being able to support queuing operations, MSMQ supports API requests from the MSMQ Explorer console application, allowing administrators to manage the system from anywhere on the network. The MSMQ Server includes a routing component that routes messages intelligently between applications and queues across the network.

Often situations will arise where a given message may need to travel complex network routes before it hits the receiving queue. These routes can span fast and slow network links and be part of a LAN or WAN. In addition to having the capability of storing messages in intermediate MSMQ Servers, MSMQ is capable of distinguishing between routes, and routing the message intelligently while taking into consideration a number of criteria such as cost and link speed. MSMQ does not support dynamic routing where the routing decisions are based on run time information available, but it does support static routing where it relies on route information configured by an administrator.

MSMQ provides a home to the MSMQ Information Service, which is a naming service (in other words, a directory) that contains all routing and configuration information for MSMQ. (SQL Server 6.5 is required for this purpose.) This directory is replicated throughout the enterprise, and each MSMQ Server should have an identical copy. MSMQ automatically synchronizes multiple copies of the directory.

To summarize, MSMQ-dependent and -independent clients are both available on Windows 95 and Windows NT. The MSMQ Server is available on NT Server 4.0 only. Something similar to the MSMQ independent client also exists in MQSeries. IBM has released a stripped down version of the queue manager available for Windows 3.1 and Windows 95. This version, called MQWin is capable of supporting queuing operations but does not have all of the intelligence that is built into the full scale queue manager.

Security

Since message queuing technology caters to mission critical applications, security is always a concern. However, the asynchronous nature of the message queuing model poses new problems to security. Most security solutions on the market today, including Kerberos, which is the base for Distributed Computing Environment (DCE) security, make an incorrect assumption that distributed applications are communicating synchronously. For example, the Kerberos mechanism is based on third-party authentication where both principals (sending and receiving application) are communicating with a DCE Security Server (which is assumed to be secure) and granted electronic tickets that are subsequently validated. Since DCE is based on the synchronous remote procedure call (RPC) communication model, there is no danger the electronic tickets will expire before the RPC request is received by the other side. When using message queuing, however, there is no guarantee as to when the request will be received by the other side. Meanwhile the electronic tickets can expire.

MQSeries provides multiple levels of security. The MQSeries API is generally represented by a library that can be protected, thus preventing an unauthorized party from making a MQSeries call. MQSeries also provides security at queue, queue manager and channel levels. For example, it can allow only an authorized party to create a queue, read from a queue and write to a queue. MQSeries also provides security exits at a channel level, adding additional security measures just before the message moves across the network. As part of its message description header MQSeries also supports user-ID, password and alternate password. On MVS, MQSeries is tightly integrated with RACF.

The only security problem not tackled by MQSeries is end-to-end authentication (also called application-to-application security). End-to-end authentication means the receiving application can be assured the message is authentic and from the sending application. To implement end-to-end authentication, messages need to be digitally signed. MQSeries does not provide digital signatures, although a third-party product called MQSecure does. MQSecure is available from Candle Corp., and it implements a RSA encryption.

MSMQ supports end-to-end message authentication. Both message encryption and digitally signed messages are available. Support for such a sophisticated feature in the first release is indeed impressive.

There are other differentiators to consider. MQSeries supports triggers where processes can be started based on the arrival of messages to the queue. Flexible trigger criteria such as queue depth is supported. Moreover, IBM's newly released Version 5 of the product supports large messages (messages sized up to 100 Mbytes), queue groups and coordination between queuing operations and database operations.

For its part, MSMQ can act as a transport for MS RPC. Programmers, of course, inherit all of the advantages provided by RPC including automatic marshaling and the Interface Definition Language, without compromising on features offered by MSMQ such as support for Quality of Service. At this point, RPC calls running over MSMQ transport are not capable of receiving any "out parameters" -- only one-way communication is supported. NT 5.0 will rectify this problem. MSMQ also supports connectivity to Microsoft Exchange. This comes in two flavors. First, two MAPI enabled applications can communicate with each other over MSMQ transport. Second, it supports the Exchange Connector where Exchange users can send messages and forms to MSMQ applications for processing. To be fair, it should be noted that MQSeries has been well integrated with Lotus Notes where it provides important connectivity between Lotus Notes Servers and non-Notes applications. It also provides support for MQLSX which allows the MQSeries API to be called from LSX, the Lotus Notes scripting language.

Since MQSeries has been around longer than MSMQ, it provides support for Windows 3.1, thus supporting the huge installed base of Windows 3.1. It remains to be seen whether Microsoft will address this issue. For now, MSMQ clients (dependent and independent) are only available on 32-bit operating systems (Windows 95 and Windows NT).

Rivals show sense

Despite all the rivalry between Microsoft and IBM in the last years, common sense sometimes prevails. Both companies recognized that interoperability between the two products is important and the MSMQ-MQSeries bridge, called the FalconMQ Bridge, is available. This means that an application using the MQSeries API is capable of communicating with an MSMQ based application. As good as this solution sounds, a number of issues remain unresolved. These include, for example, administration, configuration, monitoring, and security questions.

Stay tuned ...

There is no question that MQSeries is the undisputed leader in this important and rapidly growing MOM market. The experience and credibility that IBM has in providing solutions for mission-critical applications cannot be ignored. Despite some shortcomings, MQSeries will continue to be the market leader. Still, the lack of built-in system management capabilities, which increase the cost of ownership for the product, the lack of a built-in naming service, the limited security model and most importantly, the product's complexity might eventually slow down IBM’s market penetration and may even compromise its number one position.

While MSMQ (and Microsoft as a whole) still lacks real world experience in development of mission critical applications, it clearly offers a strong alternative from a features/functions and robustness point of view. It does not support all features supported by MQSeries, but it does offer support for message encryption and digital signatures, a built-in comprehensive directory service and, most noticeably, bundled easy-to-use administration, and a few others.

Will MSMQ really challenge MQSeries hegemony? What will be the impact of Microsoft’s entrance into the MOM market? It is too early to say. One thing is clear, companies looking for a full implementation of message queuing middleware across many operating systems will quickly find out that MSMQ’s lack of platform support will make the decision to go with Microsoft difficult. Customers who are ready to endorse NT as their primary server operating system either for a specific set of applications or for the entire enterprise might find MSMQ appealing even though their mainframes (with, possibly, MQSeries) are there to stay. Lack of heterogeneous operating systems support for MSMQ and the complexity of MQSeries might ultimately create an opportunity for a newcomer.

This article appeared in the March 1998 issue of Application Development Trends.