Articles & White Papers

Message queuing middleware can provide huge benefits, provided developers can work out complex design issues

Unmasking the mysteries of message queuing

There is no question that middleware has become a mandatory architectural component for large-scale distributed applications. But developers still must decide which middleware approach is best for specific applications.

The two fundamental approaches to middleware are data passing and process transfer. The former is built to directly access a database with or without database gateways. Process transfer includes some sort of program-to-program communication accomplished either by Remote Procedure Calls (RPCs) or messaging. Messaging is the foundation of Message-Oriented Middleware (MOM).

Messaging is the most confusing middleware category. There is no common source code as in the case of the Distributed Computing Environment (DCE) from the Open Software Foundation, Cambridge, Mass., and there is no common set of specifications such as those available with the Common Object Request Broker Architecture (Corba) standard from the Object Management Group (OMG), Framingham, Mass. Furthermore, only a half page was dedicated to MOM in a recent book on middleware entitled, "The Essential Corba" by Thomas J. Mowbray, Ph.D., and Ron Zahavi. (See reference.) The authors describe the messaging technology as immature.

Furthering the confusion, there are more than 18 sometimes drastically different products touted as message-oriented middleware. These products encompass both message passing and message queuing technologies. This article will focus on the concept, the advantages and disadvantages of message queuing technology, and not on the message passing offerings from vendors such as PeerLogic Inc. of San Francisco; Software AG of Reston, Va.; and Talarian Corp. of Mountain View, Calif. The message passing products are valuable technology, but are not the subject of this piece.

Defining message queuing

Message queuing technology is incorporated in several products, including IBM's MQSeries; Digital Equipment Corp.'s DECmessageQ; NetWeave DS from NetWeave Corp. of Wilmington, Del.; X-IPC from Momentum Software Corp. of Englewood, N.J.; and VCOM from Verimation Inc. of Woodcliff Lake, N.J. In addition, Microsoft is incorporating message queuing technology into its Back Officeoffering.

IBM describes message queuing as a software architecture that allows programs to communicate via message queues rather than by calling each other directly. Messages are placed in queues, which can be memory- or disk-based, so that programs can run independently, at different speeds, in different locations and without a logical connection between them. This description does not address any implementation details, leaving software suppliers and corporate application developers to their own interpretation of the definition of message queuing.

The OMG has moved to stem the confusion with a recent request for proposals to be used in creating an asynchronous messaging standard. However, the OMG generally discusses just interfaces and not implementation details. So the debate over the definition of message queuing systems will continue long after OMG adds asynchronous message queuing to its Common Object Services Specifications (COSS). This article focuses on general message queuing features and the types of applications for which they are best used.

Despite the confusion surrounding its definition, message queuing is a mature technology and is incorporated in hundreds of mission-critical production applications. For example, many mission-critical applications running at Volvo, a Swedish automobile manufacturer, are using the VCOM message queuing product from Verimation. To put it simply, when VCOM is down, Volvo does not manufacture cars. Fortunately, VCOM is a very stable product and as a result Volvo production stoppage is almost a nonissue. The same type of stability is applicable to a number of other message queuing products, including DECMessageQ, MQSeries, NetWeave DS and X-IPC. Message queuing technology is a mature and industrial strength technology. Even Microsoft is convinced of this, recently publishing plans for a message queuing product on its Web home page.

In any event, message queuing is a method of program-to-program communication. No one disputes this. However, a typical message queuing system incorporates a number of different aspects. They are as follows:

1. Messages are strings of bytes that have a meaning to two or more application programs. Messages can contain control data and application data. The control data is used by the message queuing products themselves and the application data is understood by the communicating application programs. Control data is generally added to the application-specific data to control how information is stored, routed, delivered and tracked.
2. Queues are "entities" for storing messages by a given application program, to be retrieved by another application program. Depending on the product specific implementation, queues can be explicit - exposed to the application programmer - or implicit - not exposed to the programmer. The question of which approach is better is the subject of industry debate.
3. Programs within a distinct application or separate applications can communicate by writing application specific data (that is, messages) to and/or from queues (explicit or implicit), without having dedicated, logical connections to link them. Furthermore, queues can be either objects that are named from an API (that is, external to the programmer), or hidden (internal to the programmer). Programs communicating through queues need not be executed concurrently as is required when using technologies like DCE, database middleware and Corba. In addition, the communicating application programs can run on the same processor, or on different processors.
4. The implicit element of a typical message queuing system is the software that controls the queues (external or internal), which can access the queues and guarantee that the message will be moved to a target destination on the same machine or on one across a network.

> Queue Manager is an acceptable name for the category. Queue manager is a system software that provides message queuing services to application programs.

Generally, queue manager is a major component of message queuing and can play multiple roles. For example, it can: (a) Manage queues where the messages are stored. (b) Furnish an API to allow different application programs to communicate. (c) Provide the communication service to assure that the message is being transferred (moved) across the network. (d) Provide a service to control the routing of messages through a network. This service can find alternate routes when a primary route is disrupted. The service can also deal with different levels of message confirmations and acknowledgments, naming, security and load balancing. (e) Provide administration services for creating and deleting queues and for altering the characteristics of queues, messages and other important attributes within the queuing system.

The above components are generic and applicable for most available message queuing products. However, the terminology used by different message queuing products can vary greatly, although the basic functionality of most message queuing systems is quite similar.

The two basic operations

For application developers, the message queuing paradigm is very simple and involves two basic operations - send (put a message on the queue) and receive (get a message from the queue). A typical application flow between communicating programs is as follows:

1. Both the sending and receiving application programs attach to the message queuingengine - generally a queue manager of whatever the terminology of the underlying product is.
2. A sending program sends a message to either the target queue or the application program it wants to communicate with, depending on whether the message queues are explicit or implicit.
3. The receiving application can be triggered by the arrival of a message. It can either receive a message from the receiving queue or receives a message from the partner program, depending on whether the message queues are explicit or implicit. If the receiving application is busy, the message will be stored in the queue until the receiver is ready.
4. After processing the message, the receiving application sends a reply back to the reply-to queue or to an application program.
5. When the information exchange is completed, both sides can detach from the message queuing engine or can continue the exchange.

While the flow described above could oversimplify the capabilities of some message queuing products, or make others too complicated, it does accurately portray the general flow of message queuing. Much of the difference between products is based on whether the queues and other important components of a given message queuing offering (queue managers, network "moving" service) are exposed programmatically or administratively and by the complexity of the API.

Regardless of the complexity of the API, using message queuing services to design applications does require extra work and considerations. The design process of an application is not as straightforward as the use of traditional synchronous communication techniques like RPCs or SQL calls when both sides (the sender and receiver) are time dependent and require both sides of the application and communication resources to be constantly available.

Message queuing with explicit queues

Let’s look at a hypothetical message queuing product with explicit queues. As described above, both the receiving and the sending applications communicate via queues. Given the indirect style of communication - application program-to-local queue, local queue-to-network-to-remote queue and remote queue- to-application - there are simply many links in the chain that can potentially fail. In addition, most applications using message queuing products are designed to run in a time-independent fashion, thus allowing applications running at different times to communicate across heterogeneous networks and systems that may be temporarily offline. The message queuing engine guarantees that the message will eventually be delivered to the target destination. If the target application is available to receive the message, and all the necessary communication "links" are up and running, the message will reach the target destination immediately, making latency nonvisible. However, simultaneous availability is not a requirement. In sum, message queuing can provide an unprecedented flexibility and benefits by supporting synchronous blocking and asynchronous nonblocking styles of communication.

This flexibility does not come for free. From an application designer’s point of view, the simple message queuing paradigm as it is outlined above does not apply - designing applications using message queuing is not as simple as it first looks. It requires an understanding and experience in dealing with issues such as asynchronous, queue-based communication between application components, event-driven processing, recovery and others.

Some benefitsof messagequeuing

Nonetheless, commercial message queuing is in vogue now because it does offer several benefits. Some of them are unique to message queuing and some are common to many categories of middleware in general. Some of these benefits include thefollowing:

1. NETWORK INDEPENDENCE. Message queuing isolates applicationsfrom communication protocols, providing greater network independence whether the application is running across a TCP/IP, SNA, or other network. Changes in the underlying communication protocols do not effect an application.
2. FLEXIBILITY IN COMMUNICATION STYLES. Message queueing has support for synchronous/blocking and asynchronous/ nonblocking communication, which allows the issuing of multiple application requests in parallel without waiting for replies. It also provides time independence, so that programs do not have to execute serially, further increasing the possibility for true parallel processing. Simply put, if an application needs to make multiple independent requests, it can send each message without having to wait for replies. This feature allows requests to be executed in parallel, reducing response time and creating new application design possibilities.
3. SUPPORT FOR GUARANTEED MESSAGE DELIVERY. Message queuing systems guarantee message delivery to the target application

regardless of the network reliability. This makes message queuing an ideal mechanism for applications communicating over intermittent networks. The guaranteed delivery feature eliminates the need for an application program to deal with local-area network and wide-area network outages, thus decreasing programming complexity and eliminating unnecessary and costly exception handling.

4. LOW COMMUNICATION OVERHEADS. Most modern message queuing technologies support session concentration, which allows multiple application programs to communicate through a lesser number of communication links established between queue managers. Such indirect communication allows many application programs to "multiplex" across a limited number of communication links, cutting communication costs and network bandwidth requirements. In addition, the communication links between the queue managers can be preestablished if necessary.
5. LOAD BALANCING. To improve performance of the flow of messages across the network, new communication links between queue managers can be quickly established as workload increases. Further, communication links can be dropped as they become unnecessary, allowing reuse of precious network bandwidth elsewhere on a network.
6. State information is kept in messages passed among applications, thus making application programs "stateless." This "statelessness" allows faster recovery in the event of a failure.
7. LOCATION TRANSPANANCY. An application requesting a business function need not be concerned where the function is provided. Applications need only be concerned with the queue names or the target application's name, depending on whether the queuing is explicit or implicit.
8. BETTER GRANULARITY/REUSABILITY OF APPLICATION COMPONENTS. Message queuing enables application designers to split a business application into multiple units of work that can run independently. Thus, a single synchronous transaction can be divided into multiple asynchronous units of work without affecting the final result of the application.

   Such a feature has been utilized well in a retail operation with an extensive network of retail stores in multiple U.S. cities with a separate corporate headquarters. Each store is responsible for keeping track of storewide inventory stored in a database management system (DBMS) from Sybase Inc., Emeryville, Calif., running on an IBM RS/6000. The headquarters' enterprise-wide inventory is kept in a mainframe-based DB2 database running IBM's CICS. The company has a requirement that both databases must be synchronized. Therefore, each time a local inventory is changed, a change to the corporate "repository" is also required. This requirement can be met by using transaction processing middleware that allows synchronous updates of heterogeneous databases, even across wide-area networks. On the other hand, using message queuing, a single synchronous transaction can be divided into three smaller asynchronous transactions. Assuming that all three smaller transactions succeed, both databases will be updated without resorting to the complexity incurred with the use of distributed TP Monitors.

   Increasingly, this technique is used when distributed and related databases need to be updated without relying on synchronous updates. This technique can avoid potential locking problems and other difficulties associated with the two-phase commit protocol, including the substantial communication overhead derived from traffic generated by an exchange of control messages associated with the two-phase commit protocol. Further, applications can be designed as a sequence of queued processes. The sequence can be determined by a workflow manager. Thus allowing applications to be developed and to operate as discrete elements. Each element has a well-defined interface, making it easier to reuse in other applications.

9. SELECTIVE RETRIEVALS. The order of message retrievals can be different from the order that the messages are stored. Modern message queuing systems allow different queue ordering (for example, FIFO, LIFO, date and time and so forth), as well as retrieval of messages out of sequence.
10. AUTOMATIC MESSGE TRACKING. Generally, message queuing systems can provide message tracking capabilities. For example, an application program sending a message could request an automatic notification when the message will either arrive to the target queue or to the target application program.
11. TRIGGERING AND EVENT-DRIVEN COMMUNICATION. Every modern message queuing system should support triggering, or the ability to start an application instance whenever a request message or reply message has arrived and the application is not already active. Allowing the applications to be active only when there is work to be done avoids unnecessary consumption of resources. The event-driven approach also opens a door for subscribing and capturing business events, thus allowing applications to register for specific events and to be automatically notified when these events occur.

As noted, message queuing can provide numerous and sometimes unique benefits. However, the biggest benefit, in our opinion, is the fact that the message queuing concept is quite easy to understand and does not require the special skills that must be learned to use some other middleware technologies. Once application designers understand the application requirements and design an application properly with message queuing in mind, the actual application development is quite straightforward.

For example, message queuing allows multiple application programs to communicate indirectly by putting request messages to queue and getting response messages from the queue. The message queuing middleware can guarantee the delivery and integrity of the messages from the sending application to the receiving application regardless of the availability of the latter or of network availability.

For the above benefits, plus the relative simplicity, we believe that message queuing has achieved widespread support among many Fortune 500 corporations. Many companies view the technology as an easy way to distribute applications because engineers are not required to undergo the substantial and sometimes painful learning curves required for other middleware technologies, especially Corba and DCE.

Message queuing is applicable across a variety of applications. While this story forgoes the many case studies that we possess, we should state that the technology is a good fit for integrating different departmental applications throughout an enterprise and not just simply as a program-to-program communication method for a single application. Message queuing can even be utilized across autonomous enterprises where synchronous connection-oriented communication would be expensive and impractical. The technology is also an excellent choice for mobile computing, workflow, E-mail-enabled and many other applications.

There are still drawbacks

Despite the impressive list of benefits provided by message queuing, we do not want to imply that a winning technology has emerged from the confusing and competitive landscape of middleware technologies. Though the technology is real and ready for prime time, it has some objective and subjective drawbacks and is definitely not a technology everyone likes.

The primary objective drawback is the technology's inability to provide adequate end-to-end security. This is a problem. Most security products shipping today were developed assuming a direct connection between the communicating parties. There is no direct communication between application programs linked via message queuing. Furthermore, there are no assurances when a message will be received due to the time independent nature of message queuing. While some message queuing products attempt to protect resources such as queues, queue managers and the communication links between the queue managers, others offer only the famous "OPP - Other Peoples Problem" solution to security issues.

The security problem is especially troublesome when dealing with two major issues - authentication and authorization. The authentication capability must ensure that messages sent across a network are authentic. The authorization feature must ensure that certain business functions can be performed only by authorized users. Message queuing technologies can leverage the security capabilities provided by native operating systems, which range from bulletproof in MVS, to somewhat less so in Unix or OS/2. Obviously, integration with those security mechanisms providing authentication and authorization such as the one provided by DCE should be examined.

To date, most applications using message queuing either go into production without adequate security, or the security holes are plugged at the application level, forcing extra work onto developers.

A second problem with message queuing is naming. Today, most applications using message queuing services benefit from location transparency - application programs are immune from the physical location of queues. However, system administrators do not enjoy such a luxury. Changes of queue location can require some sort of administrator intervention. Most of the message queuing products either offer no, or very limited, naming services. Furthermore, today's offerings do not offer a media where the business application names, queue managers and message queues used within an application can be stored. In order to reduce administration, naming services such as that provided by DCE are desirable.

Critics also note that most message queuing products do not support data marshalling. At most, Ascii to EBCDIC conversion is supported, but not more sophisticated marshalling such as automatic conversion of floating point fields. Furthermore, there is no standard API for message queuing products. On the contrary, each message queuing product furnishes its own proprietary API. Corba, on the other hand, supports the Interface Definition Language (IDL), an industry standard notation language to express the interfaces (message formats) and methods (functions).

Finally, message queuing can add an unnecessary level of complexity to simple client/server applications. Most message queuing products are capable of supporting two application modes - client/server and peer-to-peer. The former allows an application client to "function ship" an application request to the server where message queues physically reside. The latter mode allows for full message queuing support everywhere. Strictly speaking, there are no clients or servers using this mode and the application roles are functions assigned based on the origination of an application request.There is a class of relatively simple client/server applications where the message queuing model would be an overkill. For these applications, the message queuing model can create an unnecessary burden of issues to deal with. This is especially true when it comes to systems management. For example, if application programs do not communicate directly, but rather through queues, a simple condition like a full queue must be detected and dealt with immediately, causing unnecessary complexity.

Message queuing overall

So the bottom line is that message queuing can simplify some complex applications while complicating simple applications. Nonetheless, the middleware technology debate leaves many developers confused. Many have a difficult time comparing the technologies. Most do generally include DCE and Corba among the possible middleware solutions. In theory, message queuing can be leveraged with other middleware technologies (DCE and Corba) and conversely, those technologies can benefit message queuing, for the benefit of enterprise developers.

Both message queuing and DCE are mature middleware technologies. DCE is based on the RPC communication protocol, which is fundamentally synchronous and connection-oriented. The RPC mechanism requires that the sender "bind" to the receiver, which can only be accomplished when the sender and the receiver are available at the same time. Furthermore, DCE requires that the communication protocols connecting both sides match - context bridging and cross protocol routing are not supported. Though DCE is adequate on single protocol networks such as TCP/IP, there are and will continue to be many heterogeneous networks that require cross protocol routing. Layering RPC on the top of message queuing technology could add cross protocol capabilities to DCE, as well as leverage a message queuing feature such as guaranteed delivery. And DCE could provide message queuing with the necessary solution to address the problems of security and naming services.

The Corba specification defines three types of communication between objects - one-way, synchronous request/reply and deferred synchronous, which does not require the sender to block. However, the sender must poll the receiver until the request is satisfied to get a reply. Message queuing could enhance a Corba-based ORB’s functionality by providing a platform for objects to communicate asynchronously and reliably. Furthermore, objects could leverage the "triggering" capability offered by some message queuing products. So implementing ORBs on top of message queuing could also be very beneficial.

In sum, message queuing is a mature and powerful middleware technology. The fact that many mission-critical applications using message queuing are in production, represents the best testament for the technology. It is one middleware technology that is real and ready for the prime time - what a pleasant surprise.1

Reference Mowbray, Thomas J., PhD. and Zahavi, Ron. The Essential Corba: Systems Integrity Using Distributed Objects. New York City: John Wiley & Sons, 1995. (back to top)

[ADT Main | Table of Contents | SPG Main]

---

 180 Turnpike Road, Westborough, MA 01581
(508) 366-3344 FAX (508) 366-8030 e-mail:general@spgnet.com size=1>http://www.spgnet.com