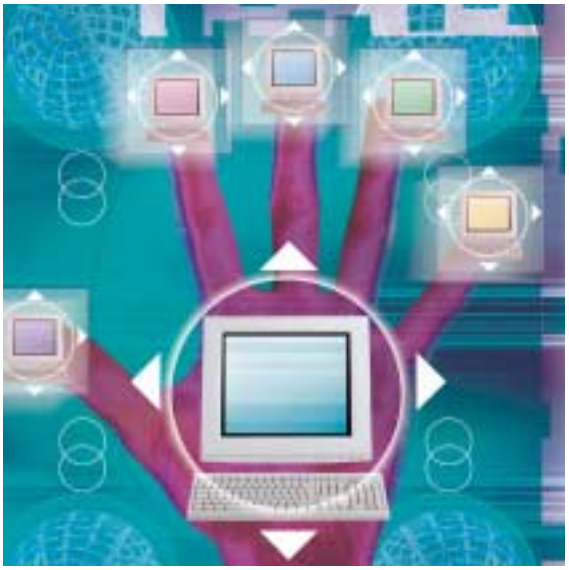

# Message Queuing for Business Integration

By Dieter Gawlick

**In response to increasingly demanding communication requirements, message queuing technology has become a central part of today's computing infrastructure. While most communication is still based on e-mail technology, the increased demand for structured communication with high service quality levels has led more companies to embrace and extend the use of message queuing technology. Message queuing technology has rapidly evolved to meet the challenges of companies integrating their businesses.**

## Message Queuing Need

As businesses grow and mature, they develop various autonomous, distributed applications to automate processes and manage tasks. These applications need to communicate with each other, coordinating processes and tasks in a consistent, reliable, secure, autonomous manner. They also need to efficiently exchange information or coordinate processes with customers and suppliers using low-cost, readily available channels such as intranets or the Internet. This must be done while preserving a traceable history of the communication — a requirement previously satisfied through now obsolete paper forms and application journals.

While the operational data of an application remains fully hidden within the application, the communication data, or message data, is used for interaction with the external world and needs to be well-specified, easily accessible, and supported by software providing superior operational characteristics (see Figure 1).

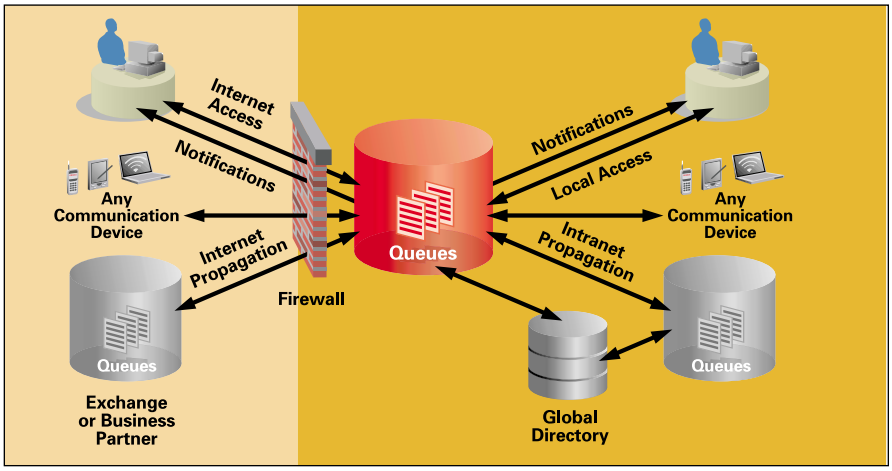


**Figure 1 — Typical Messaging Environment**

## Message Queuing Requirements

Businesses have been successfully using message queuing for more than 20 years, so requirements are well-established. As shown in Figure 2, they include a:

- Well-defined message format
- Mechanism for sending messages
- Mechanism for receiving messages.

A well-defined message consists of two parts: the payload and the header. The message payload contains the relevant information; the header contains additional information about the message such as where it originated and its urgency. Optionally, the message may contain additional attributes called properties. Messages are stored in staging areas called queues.

Message producers must be able to create messages by inserting them into a queue and message consumers must be able to act upon messages by retrieving them from the queue. Following the terminology of Java Messaging Services (JMSes), the most widely used messaging standard, producers must be able to send messages and consumers must be able to receive them.

Message recipients must be able to determine which messages to retrieve. Most basic queuing systems support a method of sending and receiving messages similar to an 800-number telephone system. Multiple producers can place messages into a queue and multiple consumers can remove messages from the queue. In this model, message consumers remove messages from the queue in the same order that the message producers originally inserted the messages into the queue. Once a message consumer has received a message, no other message consumers can access that message. Similarly, when you place a call to an 800 number to make a reservation, your call is answered in the order it was received. Once your call has been answered, you are effectively removed from the queue.

Besides supporting a simple first in, first out messaging model, many queuing systems support some form of subscription. This lets the message consumer express interest in receiving messages based on their content. For example, a user of a news reporting service might request information about a particular sports team or set of stock quotes.

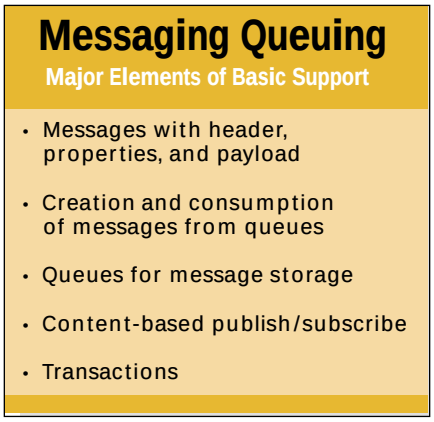


**Figure 2 — Basic Message Queuing Requirements**

## Advanced Requirements

Nearly all message queuing systems support these three basic elements:

- Message format
- Send method
- Receive method.

The sophistication of the message queuing software determines the amount of work required by the application developer. For example, some message queuing solutions include mechanisms for specifying who should receive the message and the delivery method (e-mail, fax, telephone, etc.), while others simply identify a method for removing the message from the queue and require the application to handle delivery.

To better meet the needs of modern businesses, several extensions to the basic queuing model have surfaced. These requirements (see Figure 3) include:

- Rich support for the payload
- Explicitly addressable consumers
- Superior operational characteristics.

The commonly used content-based subscription model requires a powerful type system to represent any payload and a language to express interest in messages. Using the previous example, messages containing stock quotes might be represented using a different message type than a message containing information about a golf tournament. The message subscriber would require a means of indicating which of these messages were of interest. As the complexity of an application increases, the need for a powerful type system and extensible subscription language becomes more apparent.

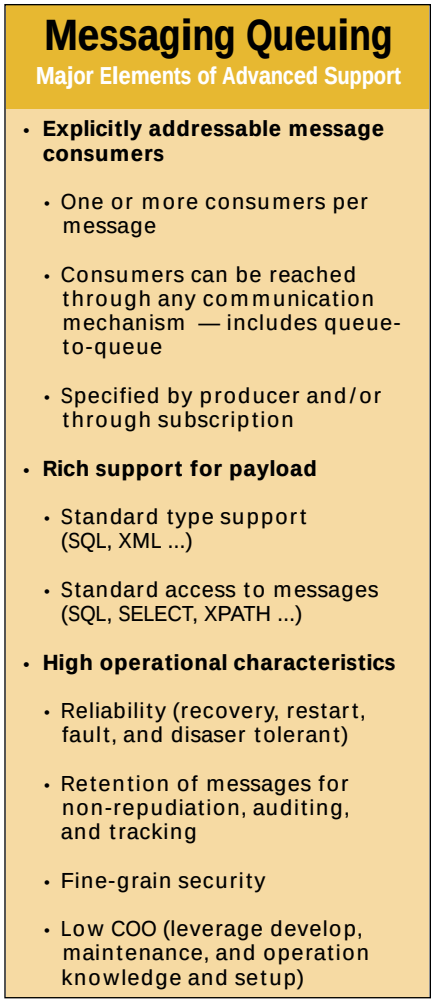


**Figure 3 — Advanced Message Queuing Requirements**

The subscription model places the control in the consumer's hands. It must also be possible for the producer to specifically address a message to a single consumer or multiple consumers. The ability to explicitly define the consumer list is especially important in business-to-business (B2B) communications. Consider the situation where the message isn't a simple reservation request, as in the 800-number example, but rather a bill from one company to another. In this example, it's important that only the intended recipient(s) receive this message.

Additionally, it must be possible to reach these consumers through any available communication mechanism such as an e-mail address, URL, or port. It must even be possible for a message recipient to be another message queue. This is re-ferred to as message propagation.

Support for message propagation is particularly important when it isn't pos-

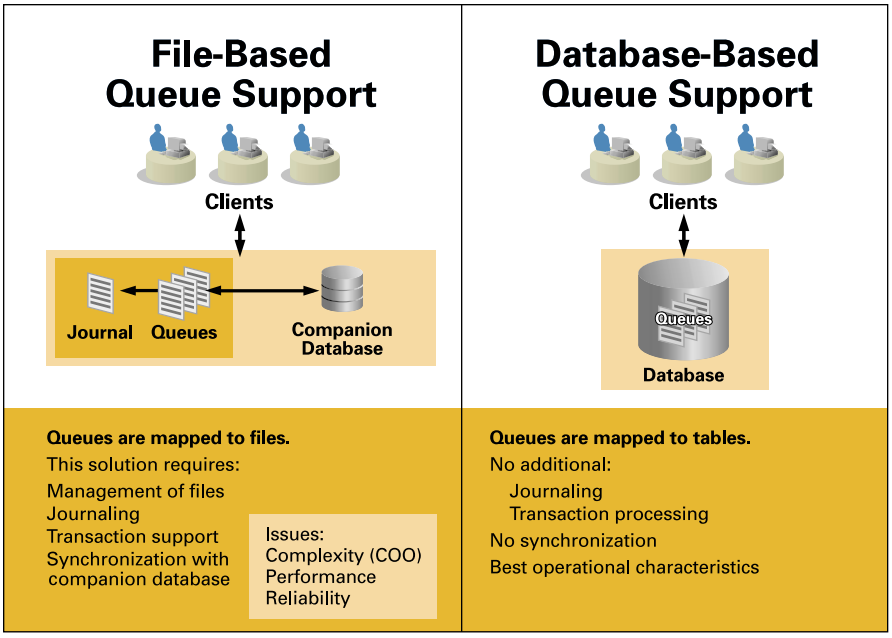


**Figure 4 — File-Based and Database-Oriented Message Queuing Systems**

sible for the message producer and message consumer to share the same queue or possibly even the same queuing system such as in B2B communications. Support for message propagation lets the message be propagated from a queue in one queuing system to a different queue in, potentially, a different queuing system at another company.

The issue of B2B communications also raises the issue of the need for full accountability. Especially when dealing with communications between two businesses, this communication must be non-refutable. It must be possible to prove that a message was sent and that its intended recipients received it. To provide this level of accountability, no message must ever be lost. This points to the need for sophisticated recovery, fault tolerance, and disaster protection.

The next requirement for complete accountability is the ability to see, or query, all messages, at all times, regardless of format. These messages include those just sent, waiting to be received, and even already consumed. So retention of consumed messages and easy access to any message, consumed or not, is crucial.

Besides the message content, information about the time of creation and consumption, and about the producer and consumer must be retained to provide correct auditing and tracking of the communication. Propagation can act as a proxy producer and consumer. In this case, the tracking information must include when propagation was attempted and when it was confirmed.

The final step in making communication non-refutable is the use of retained digital signatures to identify producers and consumers.

## Message Queuing Systems

Messages and queues as core elements of message queuing systems have to be mapped to underlying constructs. One way to do this is to represent queues as files and messages as records in these files. Probably the best-known and most widely used example of a file-based message queuing system is IBM's WebSphere MQ (formerly MQSeries).

In database-oriented message queuing systems, one or more queues are mapped to tables and messages are mapped to database records, normally called rows. This approach to message queuing is much more recent.

Either the file- or database-oriented approach can handle the basic message queuing requirements. Both methods provide support for a message format and methods for sending and receiving messages. Supporting the advanced requirements is much more complex, although leading file-based and database-oriented message queuing systems have evolved to address these requirements.

With either file-based or database-oriented message queuing, messages can be formatted with the required structure and created and consumed using one of the standard interfaces such as the JMS send and receive verbs. Most file-based messaging systems implement content-based subscriptions using some subset of XML to describe the payload and XML PATH Language (XPATH) to test for subscription matches within the payload.

With database-oriented messaging, the basic functionality is easily mapped to the existing type system, which typically includes XML and XPATH. For example, message creation can be translated into a Structured Query Language (SQL) INSERT statement, while message consumption can be translated into a SELECT. The powerful type system and variety of supported languages of a database allow definition and processing of even domain- and company-specific data structures.

## Transaction Support

Messages are often used to integrate applications, which are supported by an underlying database. In this context, it's usually preferable to generate a message in the context of a database transaction. For example, a message should be sent only to the billing department if the transaction fulfilling the order commits successfully.

With file-based message queuing, creation and consumption of messages transactionally consistent with database updates require:

- Support for distributed two-phase commit coordination
- Journaling
- Journal management
- Recovery.

The cost of distributed two-phase commit processing for the synchronization with the companion database can diminish performance and scalability. While the technology is well-understood, the actual implementation proves to be extremely difficult and expensive. Only leading file-based message queuing systems provide transactional support for message queuing.

In contrast, transaction support is inherent in the database. With a database-enabled messaging system, messages can be transactionally stored in queues mapped to database tables. These operations can be committed or rolled back as part of a transaction that includes other standard database operations. By col-

locating the operational data with the queues, the need for distributed, two-phase commit is eliminated.

## Operational Characteristics

Users of message queuing systems seek superior operational characteristics, including outstanding performance, scalability, reliability, and security. Potentially large amounts of messages have to be propagated and consumed as fast as possible, requiring high-performance and scalability. Database-oriented queuing systems can meet these requirements by leveraging the inherent scalability and performance characteristics of the database, such as sophisticated buffer management capabilities. File-based queuing systems can provide high levels of performance and scalability, too, but only if they also support sophisticated buffer management technologies.

Obviously, access to messages (consumption and auditing) has to be protected by the highest level of security. The use of retention, auditing, and tracking requires a type system, a query language, and a well-performing index technology.

With database-based message queuing systems, retention is easy to support. Instead of deleting a message when consumed, it can be updated and marked as used. Auditing and tracking applications can be built using the existing type support and query language. Note that this query language, SQL, and type system are the same as those used for content-based subscriptions.

A database-oriented message queuing system leverages the operational procedures of the database, reducing the operational burden on the IT staff. The operational characteristics are all inherited from the database and support the requirements for strategic applications. This includes fault tolerance, disaster protection, and fine-grain security. The existing operational content has to be extended to deal with the messaging objects such as the mapping of tables to queues. The major operational tasks — such as space management, recovery, and restart — don't change.

Using a file-based messaging system in a strategic environment or with sensitive data security requirements typically requires the use of a companion database to replicate all critical messages and retain historical information. When used with a companion database, file-based message queuing systems can provide a high level of reliability, including fault tolerance, disaster protection, and security.

With the introduction of a companion database, however, messages have to be created in, or consumed from, two different places using two different storage technologies. This increases both complexity and resource usage. Programmers must manage both message queuing and database technology when designing and developing applications. A file-based message system also requires a set of new operational procedures to manage various objects and tasks such as queues, recovery, restart, and security. Figure 4 provides a comparison of file-based and database-oriented message queuing systems.

## Conclusion

Autonomous applications perform tasks that need to be coordinated with tasks performed by other applications within the same or in different companies. Message queuing, one of the core technologies for this task, has to provide the communication in a highly reliable, scalable, secure manner while providing auditing and tracking. Similar requirements are found in other environments such as business-to-consumer (B2C) and peer-to-peer.

For many of the same reasons that companies have chosen to move their operational data out of files and into a database — reliability, security, scalability, performance — they now can choose to move their communication data out of file-based message queuing systems and into database-enabled message queuing systems. eAI

### About the Author

*Dieter Gawlick is an architect at Oracle's Server Technology Division. For more than 30 years, he has developed database, workflow, and integration products at IBM, Digital, and Oracle. Voice: 650-506-8706; e-Mail: dieter.gawlick@oracle.com; Website: www.oracle.com.*