Business integration solutions
To support your IT objectives

WebSphere® software

IBM

Connecting your applications without complex programming.

by James Hart, IBM Software Group

September 2003

Contents

Introduction

Today, you're probably sitting on a disparate, widely distributed, increasingly complex, enterprise-computing infrastructure. One that's made of different kinds of systems—located in and managed through different departments and geographic locations. With a dynamic messaging infrastructure, you can connect new applications and leverage existing ones cost-effectively, while minimizing your risk. And integrate these applications across your entire organization and with those of key trading partners, suppliers and customers.

To stay competitive, you can't continue to rely on manual processes to manage information that's distributed through a wide range of disconnected systems. It's expensive to maintain. More prone to human error. And doesn't accommodate future growth. A strong messaging architecture can provide you with the security-rich foundation to provide your goods and services over the Web. Facilitate more effective interactions. Streamline business-critical processes and enhance productivity across your value chain. So the flow of transactions, information and ideas can ripple immediately through your enterprise—and beyond.

IBM WebSphere® MQ software is an innovative, connectivity product—and the accepted industry standard—that has propelled the software industry into the new world of message-oriented middleware. WebSphere MQ provides assured, once-only delivery of data across a wide range of operating systems. Across industries, from banks and telecommunications companies to government agencies, IT departments have reaped the benefits of using a common technology to connect disparate systems.

In the last decade, the software industry has converged on certain standards. Web services for service discovery and invocation. Java™ 2 Platform, Enterprise Edition (J2EE) as an enterprise programming model. Or XML as a canonical data format. Messaging technology—particularly WebSphere MQ—complements these standards by playing an important role in building a strong, open, IT infrastructure.

This white paper examines the value of implementing an application-connectivity architecture instead of building a complex web of custom coding to integrate your business processes. It explores WebSphere MQ, and the key concepts that comprise the product in more detail. The paper discusses some of the standards that currently shape the industry – and why messaging is still a critical part of the picture. It also provides answers to some common questions about WebSphere MQ. Finally, this paper provides insight about the overall industry landscape, so you can more clearly understand WebSphere MQ, how it fits into today's IT industry and how it can work for you.

The role of messaging

To understand the benefits of messaging software, consider the alternatives. Most enterprises have several systems, applications or islands of automation that stand alone – often on different operating systems. Data usually resides in more than one place, causing duplication and synchronization issues. Employees manually enter data into several different systems. And if you develop or purchase a new application – or a merger or acquisition occurs – the situation becomes even more complex. To solve the problem, you need to connect your applications together, allowing them to share information and unlock the data distributed across your enterprise.

To achieve this level of connectivity, you may decide to write code, possibly added to your application logic, to communicate with other systems. This means your developers must write connectivity logic – and grapple with the nuts and bolts of a particularly difficult area of software development. It could involve issues like the handling of TCP/IP sockets, which can vary depending on the operating system and programming language used. And it can require a wide array of skill sets within your development team. The code must be able to handle situations where the network fails, or where your target (receiving) application is unavailable. Because each piece of connectivity logic is specific to the applications it connects, you limit the possibility of reuse and make it more difficult to add applications as your business needs change. The likely result is your IT staff writes, owns, extends and maintains a large quantity of complex, unwieldy, connectivity logic.

You can avoid this situation by using a software product designed to handle all these connectivity issues for you — message-oriented middleware. Rather than connectivity logic, let your applications talk to the application-connectivity infrastructure — through a simple, common application programming interface (API) — to deliver the data to other applications.

Key benefits to implementing messaging

When you implement messaging you take advantage of a simple and common API. You adopt industry-standard programming models and you make these available on a selection of operating systems. An effective messaging layer should be ubiquitous to maximize reuse of skills and code across your enterprise. Your application developers can simply concentrate on writing business logic without having to maintain large quantities of connectivity code.

A vital aspect of messaging middleware is assured delivery. You have to be able to control the required quality of service on data delivery. For example, it may be acceptable to send noncritical data in a fire-and-forget model, where you're aware that the data may be lost, given certain failure scenarios. However, for critical business information, like a banking transaction, you want assured once-and-once-only delivery. When the application sends this critical data to the messaging layer, the processing should continue. If messaging events — like sending and receiving data — can also act as part of a transaction, it helps ensure that actions, for example, database updates, can occur in the same unit of work as messaging operations, with coordinated commit or rollback.

Another central concept of messaging, time-independent — or asynchronous — processing means that applications don't rely on each other's availability, or the availability of the network, to send data. In a purely synchronous model, in the case of a network failure, your applications would require sophisticated retry logic and could be blocked waiting for the network to recover. Asynchronous messaging is best viewed as a delivery model — not as the opposite to synchronous messaging. Asynchronous messaging simply decouples applications from each other and from the network. It operates on a fastest-possible delivery model. If you wish to send data from application A to

application B, and the network is available, the data will be delivered almost immediately. However, if the network or the receiving application is unavailable, the sending application isn't necessarily impacted. The messaging layer will temporarily store the data if required.

Because your application-connectivity infrastructure can be the core of your enterprise, potentially handling all of your business-critical data, you must ensure that the infrastructure you choose is reliable. You have to make sure data doesn't get lost. Your messaging software must operate in a failover model. It needs built-in capabilities to help ensure high availability. You may also want features to help with workload balancing, so you can be sure your application-connectivity architecture can grow and adapt as your business requirements change.

Paradigms, acronyms and buzzwords: a messaging vocabulary lesson

A host of terms have been associated with messaging. Depending on your audience, messaging equates to a van, a bus, a pipe, a hub-and-spoke or a star. You'll also hear about paradigms, like point-to-point, fire and forget, request/reply and publish/subscribe. Or you'll hear about adapters (fat and thin), brokers and distributed brokers. Terms like J2EE, Java Message Service (JMS), Simple Open Access Protocol (SOAP) and XML are also prevalent.

Point-to-point and publish/subscribe are the two main messaging paradigms, as shown in Figure 1. A point-to-point model draws a direct line between two applications. The logic defining where you send your data resides either in the originating application, or in the messaging artifacts defined locally to this application. If you want to modify the location of the target application, you impact the source. If you want to connect n applications to each other, you have to define $n(n-1)/2$ connections in your messaging middleware – to connect each point to each other point. A good messaging product can provide some additional function to make it easier to administer this type of architecture.

Other types of point-to-point architecture include hub-and-spoke, request/reply and fire-and-forget. With the hub-and-spoke architecture, each point connects into a central piece of software. By implementing this type of architecture, you can minimize the number of connections. The request/reply

architecture enables your originating application to send data, specifying it as a request message, together with an address describing where replies are to be sent. The application can then wait for a reply or continue processing. A fire-and-forget architecture simply refers to a situation where you send data and do not expect a reply or confirmation.

Publish/subscribe allows you to abstract routing (delivery) information from the endpoints and centralize it in a broker. In this model, you send data through a broker without knowing its exact final destination. You can categorize this data using topics. Receiving applications register with the

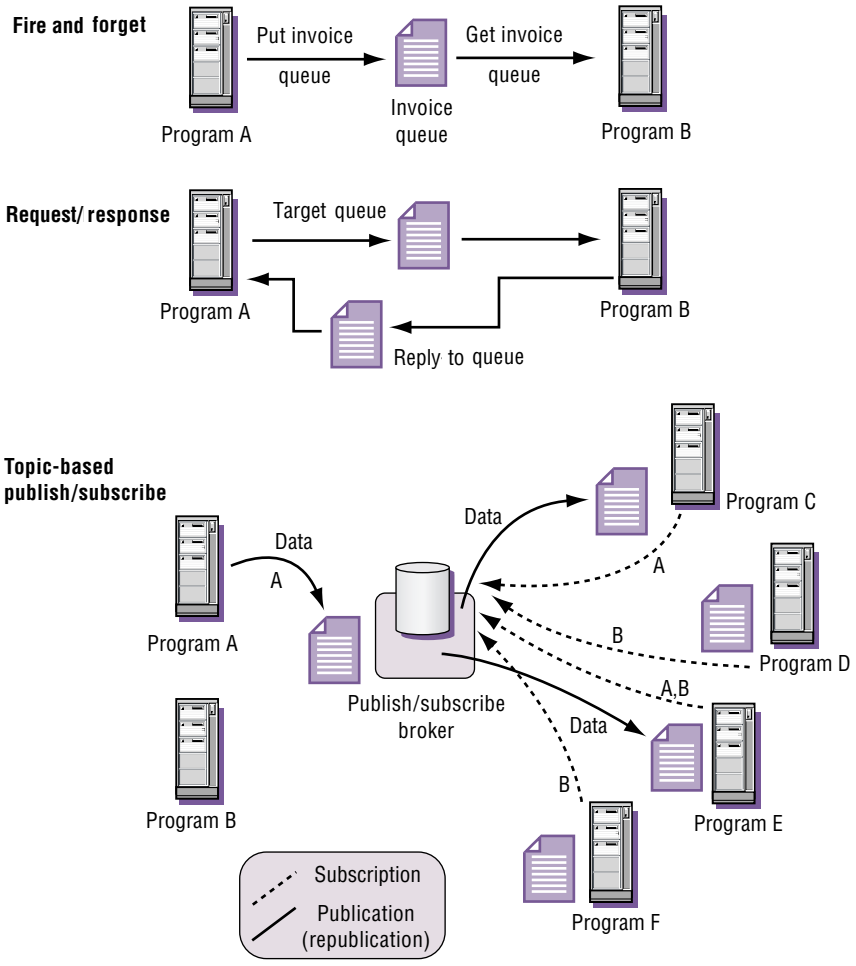

Figure 1. Messaging paradigms

broker by providing delivery information (where should I send the data), and subscribing to topics that are of interest (which data do I want to receive). The publish/subscribe model gives you more flexibility, allowing you to dynamically change the intended recipients of data with minimal impact. While this model requires the use of a publish/subscribe broker, it shouldn't impact your choice of quality of service, or your ability to use other concepts, like request/reply.

These are the fundamental concepts of messaging. Whether you decide to render this as a bus, a pipe or a hub is a subjective issue, depending on how you decide to draw the picture. Technological factors can also influence how you characterize your application-connectivity infrastructure. You may consider it a bus if it contains some intelligence regarding the routing and transformation of your data.

Messaging and the bigger integration picture

You can reap substantial benefits by employing an application-connectivity architecture—even if it's only to provide simple connectivity within a small enterprise. However, messaging is a base technology that can enable other integration capabilities, to handle more complex integration issues as your business needs grow.

In conjunction with the messaging layer, brokers and adapters can provide other application-connectivity services that can help get the right data to the right place, in the right format. Message-broker software can dramatically simplify a maze of interconnections between applications by providing the hub in a hub-and-spoke architecture. You can also implement process integration—the modeling, automation and control of business processes—allowing you to choreograph interactions across many applications and people.

Together, these elements comprise business integration. Building a business integration architecture involves strategy, and a suitable architecture for your current integration scenario may not fit the project you're planning for next year. So it's vital to retain flexibility to allow future growth, and to adapt quickly and easily to new technologies and concepts. IBM WebSphere business integration software gives you that flexibility.[1]

Messaging lies at the heart of business integration – currently the highest priority in IT spending among CIOs. However, while 43% of CIOs list a real-time, integrated view of the company as a critical goal and an additional 28% say it is a desirable goal their business units want, only 1% have actually completed this integration.[2] The remainder of this paper concentrates on the messaging aspects of WebSphere Business Integration software – and, in particular, WebSphere MQ software.

WebSphere MQ key concepts

The fundamental building blocks of a WebSphere MQ architecture are messages and queues. This section briefly discusses these and some other key components. For more in-depth information, refer to the WebSphere MQ product manuals.

Messages

A message is a string of bytes, containing the data you wish to deliver from one application to another. A message contains two parts, application data and a message header. The applications you are connecting define the application data. WebSphere MQ software doesn't distinguish what the data represents or which format you use to represent it. For example, if your data is in XML format or another kind of legacy bit stream, it would be transparent to the WebSphere MQ product.

The message header describes the message. It contains important information used in its delivery – including a unique identifier for the message, information about where it came from, the code page of the application data, the priority of the message and various other fields. The header always includes certain fields contained within the mandatory MQ Message Descriptor (MQMD). You can also use any number of optional headers specific to the type and use of your message. For example, if you want to send a message to a central broker, you can use an MQ Rules and Formats Header (MQRFH2 Header) to provide extra information about how to parse the application data.

Messages can be as small as a few hundred bytes—and as big as 100MB. WebSphere MQ provides the ability to split large messages into several logical groups or segments for transmission.[3] You can control this either by applications or with the queue manager itself. Messages can be persistent (assured once-only delivery) or nonpersistent. Persistent messages are logged to ensure recovery and delivery even in the case of a WebSphere MQ failure. Nonpersistent messages are not logged, and are delivered at most, once, ruling out the possibility of duplicate message delivery.

Queues

A queue is a data structure in which messages are stored. Applications or queue managers can put messages to, or retrieve messages from a queue. Under normal circumstances, the process of putting a message to a queue, the message traveling across the network and the message being received by the target application occurs in near real time. The queue adds value to your application-connectivity infrastructure by providing storage for messages—allowing decoupling of your applications from each other and from network availability. You can configure how messages move on and off queues, with the default being first in/first out processing.

Queues are independent from the applications that access them because they're owned by queue managers that run in their own application spaces. The most common types of queues include:

- *A local queue that physically exists on your local system.*
- *A remote queue, which is a local queue on another queue manager.*
- *A transmission queue, which is a special queue where messages reside before they move across the network. You don't manually put messages directly to a transmission queue; the queue manager handles this internally. The primary purpose of this queue is to provide storage for messages if the network is unavailable.*
- *An alias queue, which is a pointer to a local queue, providing extra direction between your applications and the physical local queues they communicate with.*

Queue managers

A queue manager provides message-queuing services to applications, and manages objects (like queues and channels). It controls administration of objects, either from the tools provided (graphical and command line) or via special command messages. It also provides services like triggering—where applications or channels can be started based on the arrival of messages on queues. Some of the other services provided by the queue manager include acting as a transaction coordinator (except on IBM z/OS® systems) and ensuring conversion between different character sets.

As owner of the various components created within it, the queue manager handles the recovery, persistence and assured delivery of messages. For persistent messaging, the queue manager logs data on the disk. For these reasons, the WebSphere MQ queue manager is often backed up in a high-availability environment.

Channels

WebSphere MQ provides intercommunication between queue managers (or between queue managers and clients) using channels. Communication between two queue managers (also known as server-server communication) requires the definition of message channels. These are unidirectional and so are often defined in pairs. Each direction will usually involve the definition of a sender channel at the source queue manager and a receiver channel at the target queue manager. The sender channel has attributes that detail how to connect to the target machine using the specified network protocol. The sender channel is also tied to a particular transmission queue.

A standard server-server communication scenario operates as follows:

1) *Application A puts a message to a remote queue.*
2) *The remote queue is a reference to a local queue in another queue manager, and is tied to a particular transmission queue.*
3) *The message goes onto the relevant transmission queue.*
4) *If the network is available, the message travels across the channel and is delivered to the target queue, where it is picked up by the target application. If the network is unavailable, the message remains on the transmission queue until the channel is running.*
5) *A reply message is often sent using the same method.*

In Figure 3, an application on System A puts a message to either a local queue (Q1) or to a remote queue, which is a pointer to Q2 on System B. The message channel agent (MCA) is the part of a queue manager that manages channels. WebSphere MQ clients connect to queue managers using a bidirectional channel. For further discussion, see a later section about clients and servers.

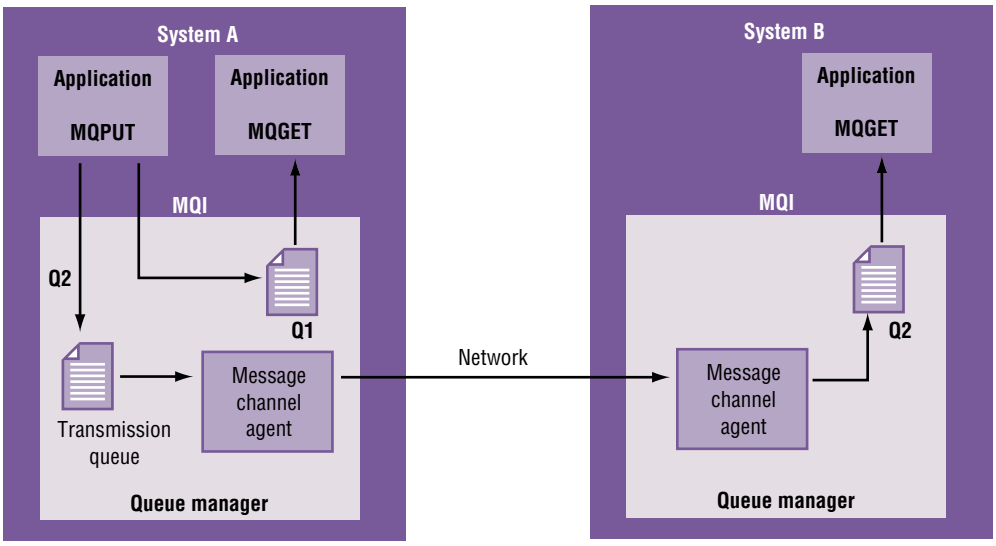

Figure 3. Distributed queuing

Clustering

Similarly, special channels connect queue managers into WebSphere MQ clusters. A cluster is a named collection of queue managers, intended to simplify setup, administration and workload balancing. A queue manager can stand alone or belong to one or several clusters.

Clusters require that at least one of the queue managers in the cluster must be defined as a repository (a place which holds the shared cluster information). Typically, two or more such repositories are designated to provide continued availability in the case of system failure. WebSphere MQ synchronizes the information in the repositories.

A queue defined as a cluster queue can be regarded as a public queue because it's freely available to other queue managers in the cluster. This contrasts with noncluster queues, which are accessible only when a local definition of them is available. Thus, a noncluster queue has the characteristics of a private queue, accessible only to the queue managers that have been configured to know about it.

Public queues with the same name in the same cluster are equal. If a message is sent to that queue name, WebSphere MQ sends it to any one of the instances, using a load-balancing algorithm. To avoid this, you can use the queue manager name and queue name in the address to force the message to be delivered to a specific queue manager. Or you can replace the load-balancing routine with a different implementation.

Each queue manager in the cluster requires only a cluster channel to communicate with the other queue managers in the cluster. This is much simpler than traditional, distributed queuing where channels are usually defined between each pair of queue managers that need to communicate. WebSphere MQ clustering should not be confused with hardware clustering, used to provide standby or failover support for high availability.

Clients and servers

Along with server-server communication, WebSphere MQ also supports the concept of client applications that can connect to a remote queue manager. The clearest distinction between a client and a server in WebSphere MQ is that a client isn't a queue manager. It doesn't have any local queues and, for this reason, can't provide any storage for messages. A client connects to a queue manager using a dedicated, bidirectional channel. If the network is unavailable, the client won't connect—and your application must manually try to connect again, or handle the situation in another way.

The decision whether to use clients or to use purely server-server communication depends on your requirements. If you need assured delivery, or your network is prone to failure, a client may not provide the necessary quality of service.

You can download most clients from the WebSphere Business Integration Support Pac Web site. These clients are generally fully supported, and are often updated and improved. Recent additions include transactional clients and 64-bit support on certain systems. Some other clients are available from third-party vendors.

Security

WebSphere MQ contains several built-in security features. WebSphere MQ, Version 5.3 introduces Secure Sockets Layer (SSL) support. This support allows a strong authentication of one channel to the other before any data is passed—protecting against malicious attacks to access production systems. You can also use SSL to provide bulk, Virtual Private Network (VPN)-like encryption for all data flowing across a channel.

With the introduction of IBM WebSphere MQ Extended Security Edition, Version 5.3[4], you can now choose the appropriate level of security to meet your specific business needs. This new offering combines WebSphere MQ with IBM Tivoli® Access Manager for Business Integration to deliver a security-rich environment to build business-critical applications. By extending the base security features of WebSphere MQ, WebSphere MQ Extended Security Edition provides individual, message-level, security encapsulation—giving you true, end-to-end, data protection. You can set security policies that enforce end-to-end integrity and privacy protection for sensitive business data without requiring security-specific coding in each application. Remotely manage both access control and data-protection security policy using a Web browser interface. And deploy WebSphere MQ Extended Security Edition, along with preexisting WebSphere MQ environments.

z/OS platform characteristics

Most of the function of WebSphere MQ extends across the majority of supported operating systems. However, the mainframe (z/OS, IBM OS/390®) is different. WebSphere MQ includes some extra features that use the unique capabilities of this operating system.

Shared queues enable multiple queue managers within an IBM Parallel Sysplex® to access the same queue – improving availability for old, new, persistent and nonpersistent messages. It also provides scalable capacity, low-cost messaging within a Parallel Sysplex and easier administration.

You can also implement pull workload balancing. By defining an application's input queue as a shared queue, you make any message put to that queue retrievable by any queue manager in the queue-sharing group. You can configure a queue manager on each z/OS image in the Parallel Sysplex. And by connecting them all to the same queue-sharing group, you can let any one of them access messages on the application's input queue.

When a queue manager connected to a shared queue fails, other queue managers in the queue-sharing group can undertake peer recovery. This ensures that in-flight messages and transactions are recovered and not lost or cancelled. Queue-sharing groups and shared channels lead to very high-availability messaging on the z/OS platform.

Tight integration from WebSphere MQ on the mainframe with IBM CICS® and IBM IMS™ using the IBM MQSeries-CICS Dynamic Program Link (DPL) Bridge, IBM MQSeries-CICS 3270 Bridge and IBM MQSeries-IMS Bridge.

Ubiquitous platforms and APIs

One of the key attributes of any successful messaging product is ubiquity. A solution that can only connect two or three different operating systems quickly becomes obsolete within your enterprise.

WebSphere MQ supports virtually all popular commercial platforms.[5] Supported server (queue manager) platforms include:

- *IBM AIX®.*
- *IBM @server iSeries™ (IBM OS/400®).*
- *HP-UX.*
- *Linux on Intel®.*
- *Linux on IBM @server zSeries™.*
- *Sun Solaris operating environment.*
- *Microsoft® Windows NT®, Windows® 2000, Windows XP.*
- *z/OS (OS/390).*

WebSphere MQ software is recognized industry-wide as the de facto messaging standard. Businesses all over the world are taking advantage of this leading-edge technology, including:

- *10,000 customer sites.*
- *Two-thirds of the top 100 North American and European banks.*
- *550 independent vendors.*

Supported client platforms include:

- *Apple Macintosh operating system.*
- *Data General DG/UX.*
- *DOS.*
- *IBM 4690 operating system.*
- *Stratus VOS.*
- *Windows NT, Windows 2000, Windows XP.*
- *zVM and VM/VSE.*

Ubiquity also means being able to use different programming languages, models and paradigms. All of the interfaces discussed here can provide both local (server) and remote (client) access to queue managers.

Message queuing interface

WebSphere MQ uses the message queuing interface (MQI) as its native programming interface. It allows you to:

- *Connect programs to, and disconnect programs from, a queue manager.*
- *Open and close objects—like queues, queue managers, name lists and processes.*
- *Put messages on queues.*
- *Receive messages from a queue, or browse them (leaving them on the queue).*
- *Inquire about the attributes (or properties) of MQSeries objects, and set some of the attributes of queues.*
- *Commit and back out changes made within a unit of work, in environments where there is no natural sync-point support, such as for IBM OS/2® and UNIX® systems.*
- *Coordinate queue manager updates and updates made by other resource managers.*

MQI provides structures (groups of fields) that you can use to supply input to, and get output from, calls. It also provides a large set of named constants to help you supply options in the parameters of the calls. Data definition files supply the definitions of the calls, structures and named constants for each of the supported programming languages. Default values are set within the MQI calls.

Depending on the operating system being used, you can use C, C++, Java, PL/I, COBOL, Visual Basic, ActiveX/COM, Assembler, RPG or TAL to program the MQI. Support for other languages, like Perl, are available as downloads.

Java Message Service

An industry-standard API for accessing messaging services from Java applications, JMS is prevalent in situations where you use messaging to connect modern, Java technology-based applications (often running in an application server) to each other, legacy applications or to an integration hub.

Using JMS as your API provides portability at the application layer—an important advantage. The theory here is that you can write an application that uses JMS to provide messaging services. These JMS calls are provider-neutral, i.e. they contain nothing specific to WebSphere MQ, or to any other messaging product on the market. An extra level of indirection exists between the messaging code and the underlying messaging product. Administered objects fill the gap. These objects are defined in a namespace and allow your applications to resolve the provider-neutral JMS code to provider-specific objects at run time. For example, your application may put a message on a queue called Q1. The application has no knowledge of whether this is a WebSphere MQ queue, or an analogous component in another messaging product. The Q1 is defined in a namespace (maybe Java Naming and Directory Interface [JNDI]) where it has a number of attributes that map this abstract definition to a physical WebSphere MQ queue. Therefore, you could remove the physical messaging product, and plug in an alternative JMS provider, without impacting your application.

However, JMS is an API—not a messaging product, and the JMS specification makes no stipulation about the technology used for the underlying data delivery. You shouldn't expect applications using different JMS implementations to communicate.[6]

Other protocols

Besides the protocols and quality of service offered within the WebSphere MQ product, other technologies are available within the WebSphere Business Integration portfolio to extend integration to new devices and industries. IBM WebSphere MQ Everyplace® applies the core concepts of commercial messaging to handheld devices and pervasive computing. Optimized for use on hardware with limited resources and for transmission over unreliable networks, this highly configurable product interoperates seamlessly with WebSphere MQ. IBM WebSphere MQ Telemetry Transport allows a WebSphere Business Integration hub to communicate with very small (and embedded) devices, like sensors, valves and meters. IBM WebSphere MQ Real-time and Multicast Transports provide very high-performance, publish/subscribe data distribution through IBM WebSphere Business Integration Broker. IBM WebSphere MQ Web Services Transport provides additional Web services support through built-in support to the brokers.

Messaging and the future

As the IT industry moves forward, enterprises will start to adopt emerging standards and consolidate their strategies through a preference for particular technologies, platforms or architectures. For example, your company may have a policy about hosting all critical applications on z/OS mainframes. Maybe you develop all new applications using Microsoft .NET technologies. Or use Oracle databases at the core of your enterprise. Regardless of your preference, you'll still require messaging whenever you need assured, once-and-once-only delivery of data between applications. WebSphere MQ offers an ideal solution for this — no matter what your company's IT preferences are. Why? Ubiquity. WebSphere MQ can run on all of these operating systems and platforms. It includes bindings into the .NET environment and, through its simple API, can easily connect into any programming model.

You may also want to adopt standards to provide protection and to ensure vendor neutrality for your IT development and purchasing. In particular, J2EE technology is becoming fundamental to the IT strategies of many enterprises. And Web services concepts are rapidly being adopted to provide a framework for service-oriented architectures. Messaging — along with WebSphere MQ — continues to play a vital role in these new areas.

Messaging and J2EE

J2EE has emerged as a common framework, underpinning the IT strategy of many companies. It dictates standards for various aspects of application development, deployment and hosting. The proliferation of application servers that can host applications according to the new standards goes hand in hand with the rise in J2EE popularity.

The overriding premise of J2EE architectures is vendor-neutrality—the ability to develop your enterprise's IT infrastructure without using proprietary solutions from any one vendor. J2EE (and the various standards that it contains) isn't software, but a specification, containing guidelines, rules and APIs that you must follow. For example, to connect to a database from within a J2EE application, most programmers use Java Database Connectivity (JDBC) technology. As an API, the JDBC specification doesn't specify what type of database you want to connect to. If database vendors want you to connect to their databases using the JDBC specification, they must provide a JDBC driver that obeys all the rules of the JDBC specification, which is part of the J2EE specification.

Similarly, with messaging, the J2EE specification contains JMS. For a messaging provider to support the JMS API (to allow you to use JMS to communicate with its messaging product), it must be J2EE technology-compliant. The resulting vendor neutrality means that your application doesn't need to know which messaging product you're using. You can replace your JMS provider with another without impacting your application. So, if you require messaging services from within a J2EE environment to other J2EE applications, or to legacy applications, you can use JMS. This is the preferred solution if you want to decouple your applications or have assured once-and-once-only data delivery.

WebSphere MQ has an award-winning JMS implementation. WebSphere MQ Everyplace and WebSphere MQ Real-time Transport each include JMS implementations—allowing you to plug in the IBM messaging transport of your choice. With the interoperability of WebSphere MQ and WebSphere MQ Everyplace JMS implementations, you can send a WebSphere MQ Everyplace message using the JMS specification. Then pick it up on WebSphere MQ (using JMS) seamlessly—function above and beyond that dictated by the JMS specification.

IBM WebSphere MQ,
Best Java Messaging Middleware,
– Java Pro Magazine 2003 Readers' Choice Awards

Messaging and Web services

Web services technology defines standards for classifying, externalizing, finding and invoking services. These services could range from operations residing within your existing, legacy applications – that can be made available to the rest of your enterprise – to new J2EE components sitting within an application server. And as Web services standards develop, concepts like Web Services-Reliable Messaging are emerging.[7]

Web Services Description Language (WSDL) defines the standards for Web services. Universal Description, Discovery and Integration (UDDI) provides directory and search capabilities. SOAP is the data format used when communicating with a Web service. However, a predicated standard for data transmission doesn't exist. You can send your SOAP messages using whatever transport suits you. One common method is to send SOAP messages over HTTP or HTTP/S. WebSphere MQ Web Services Transport allows you to send SOAP over HTTP into and out from WebSphere Business Integration Message Broker, Version 5.0.

If you want your data (probably in SOAP format) to be delivered with the quality of service and benefits of messaging, you must use a messaging product for the underlying data transfer. You can use WebSphere MQ to send and receive SOAP data within a Web services implementation – most common within the J2EE environment – and often referred to as SOAP over JMS. A currently available support pack can help you set this up.

Exploding some myths

Many common misconceptions exist about the benefits of messaging to support application connectivity. This section addresses some of these questions.

Does asynchronous mean slow?

No. An asynchronous delivery mechanism simply decouples applications. If everything works smoothly, i.e. your applications are running and the network is up – the system delivers your data just as quickly as with synchronous communication. Asynchronous messaging enables the messaging layer to cope with situations where parts of the architecture are unavailable.

Can I use WebSphere MQ in a real-time scenario?

Yes. Because you use an asynchronous delivery mechanism doesn't mean you can't use WebSphere MQ in synchronous architectures. Using asynchronous messaging, your originating application can continue processing after putting a message to a queue, even if it's waiting for a response – but it doesn't have to. If you want your application to mimic synchronous processing, it is perfectly feasible to program it to do so. An example of how this works follows:

1. *A user logs on to an online, banking Web site and makes a request.*
2. *The application serving this request runs as an Enterprise JavaBeans (EJB) component in a J2EE technology-compliant application server.*
3. *Using JMS, the application sends the request (with WebSphere MQ as the underlying messaging product) to an integration hub for processing. A request message is generated.*
4. *The application waits for a reply from a specified queue.*
5. *If all parts of the enterprise work correctly, the integration hub constructs the reply message, using data from within the enterprise, and using the information you've specified in your initial request metadata (for example, where to send replies and what the ID of the initial message was).*
6. *WebSphere MQ sends this message back to your EJB, where it's picked up, and a reply is returned to the user.*
7. *Also when the EJB (using JMS code) component waits on the queue, you can also specify a time-out.*
8. *If a reply isn't received within the specified time-out period, WebSphere MQ can inform the user that the action could not be performed and send a back-out message to the integration hub. Or the user may choose to receive an e-mail when the action completes, for example.*

Does queuing mean my data is always waiting?

No. The term queue can be somewhat of a misnomer. The default mechanism for getting data in and out of WebSphere MQ is first-in/first-out. The action is analogous to queuing, where the newest member joins the back of the queue. But the analogy ends there.

In a well-crafted WebSphere MQ installation, data doesn't usually wait for long periods of time. Messages should only stack up in queues if they're waiting for a network connection to become active, or for an application to start up. Basically, a queue isn't a database. Queues store messages only if they can't be delivered. You should monitor the depth of queues (the number of messages) and take action if it gets too large. Large queues normally indicate a problem, and imply that a channel or application isn't operating properly.

Other performance information

The WebSphere Business Integration Support Pac Web site[8] contains comprehensive performance reports for WebSphere MQ about a number of operating systems. You can find them in the drop-down menus on the Web site and download them at no charge. The performance information includes both detailed throughput analysis and simpler headline numbers. They also provide guidance on capacity planning and performance tuning.

Summary

Although WebSphere MQ is a mature product, with a strong reputation and market presence in the established message-oriented middleware space, it continues to evolve—anticipating any potential paradigm shifts in the IT industry. As J2EE technology grew in importance, WebSphere MQ was one of the first messaging products to support the JMS standard. Today, as Web services gain momentum, IBM stands at the vanguard of standards definition in this area. And WebSphere MQ plays a key role as these exciting technologies develop.

At the same time, the value of messaging remains current for your enterprise—regardless of its size. If you're a large business interested in implementing a business integration architecture, WebSphere MQ can help you form a fundamental connectivity layer to build on. If you're a small or midsize business requiring only a handful of WebSphere MQ components, the benefits of implementing a messaging solution to solve basic connectivity issues remain substantial. Whatever business you're in, you can feel confident in the knowledge that IBM continues to invest in industry-leading WebSphere MQ software. IBM will continue to add new function and support new technologies, while improving existing function.

For more information

WebSphere MQ software isn't just a product with a distinguished past. It's also a product with a long and exciting future dedicated to solving the fundamental connectivity problems faced within the modern IT industry. To learn more, visit:

ibm.com/software/integration/wmq/

To learn more about WebSphere Business Integration software, visit:

ibm.com/software/integration

IBM Corporation
Hursley Park
Winchester
Hampshire
SO21 2JN
United Kingdom

Produced in the United States of America
09-03

[1] To learn more, visit the IBM WebSphere Business Integration Web site at **ibm.com**/software/integration/library/.

[2] Morgan Stanley CIO Survey Series, Release 3.4, June14, 2002.

[3] Message segmentation is not available on z/OS.

[4] You can upgrade your existing WebSphere MQ, Version 5.3 licenses to the full function of WebSphere MQ Extended Security Edition, Version 5.3 by licensing Tivoli Access Manager for Business Integration, Version 4.1 for these systems.

[5] For a full listing, visit **ibm.com**/software/integration/platforms.

[6] However, there is interoperability between IBM JMS implementations for WebSphere MQ and WebSphere MQ Everyplace.

[7] To learn more, read “Implementation strategies for WS-Reliable Messaging” on the IBM developerWorks Web site at **ibm.com**/developerworks/webservices/library/ws-rmimp/

[8] To visit the WebSphere Business Integration Support Pacs Web site, go to **ibm.com**/software/ts/mqseries/txppacs/txpsumm.html.

All statements regarding IBM future direction or intent are subject to change or withdrawal without notice and represent goals and objectives only.

G325-2390-00