JMS Messaging in WebSphere Application Server Version 5

Jamie Roots

January 13, 2003

1 Introduction: Messaging, Application Servers, JMS, and J2EE

In Version 5, WebSphere Application Server includes for the first time a built-in implementation of the Java Message Server (JMS) 1.0.2 API, as part of its support for Java 2 Enterprise Edition (J2EE) 1.4. The JMS specification defines the standard API for Java applications to use to perform messaging. An implementation of the API is termed a "JMS Provider", and products that claim J2EE 1.4 certification, such as WebSphere Application Server, must include a JMS Provider as part of the product.

The purpose of this paper is to help you choose from the JMS messaging options available from IBM, according to the needs of your J2EE applications. In particular, the messaging technology built into WebSphere Application Server Version 5 is compared with WebSphere MQ and WebSphere MQ Event Broker.

People have been building systems using application server (or formerly "transaction monitor") and messaging technologies for many years. Most uses of messaging within an application server environment fall into one of these categories:

- To provide *asynchronous communication* between applications or application components. The sending of a message between applications represents some sort of event, and the content of the message represents data associated with or describing the event. In point-to-point messaging, the message is directed as a specific target application. In publish/subscribe messaging, receiving applications register interest in different event types by subscribing to one or more topics, and each message published to a topic is distributed to all of the applications that have subscribed to that topic. These uses of messaging are forms of asynchronous communication, because the message merely carries information, and does not imply that flow-of-control has passed from the sending to the receiving application, as with other forms of distributed communication, such as RPC.

  In J2EE, messages corresponding to asynchronous events can be received into the J2EE server using Message Driven Beans (MDBs), which are a type of Enterprise Java Bean (EJB). When an MDB is deployed, the destination, that is, queue or topic, from which it is to receive messages must be specified. The application server is notified when a message is sent to that destination (event) by the JMS Provider, and calls the MDB, passing it the message content.

- To provide *heterogeneous integration*. It is frequently the case that applications that run on different hardware, with different operating system software, in different application server environments, have no other communications technology in common, and must use messaging in order to communicate. Conversely, it is rarely the case that two applications,

however different their environments, are unable to exploit messaging in order to exchange information. When messaging is used for this purpose, the style of communication may be asynchronous, synchronous (RPC-style), or even batch. In all cases the messaging transport itself is by nature asynchronous, and the other styles of communication are built out of the asynchronous messaging primitives using well-established design patterns.

J2EE is a very popular choice of technology for implementing new applications, and Web Services is becoming more and more important as an integration techology. Nevertheless, the number of J2EE applications that need to be able to integrate with systems that use neither J2EE nor Web Services, continues to grow. The availability of heteregenous messaging to accomplish that integration, exposed to J2EE applications through the JMS API, is extremely important in most enterprises that have selected J2EE technology. WebSphere Application Server, in common with other J2EE Application Server products, relies on WebSphere MQ or another external JMS product for heterogeneous messaging.

- To provide *temporary data storage*. In transaction monitor environments such as CICS, specific facilities were provided for storing temporary data on queues. By writing the data to a queue, it could be recovered in the event of system failure, but unlike when using a database, the schema of the data did not have to be predefined. Many other sorts of applications have also used queues as a convenient place to store data for a while.

  A large part of the purpose of J2EE is to enable application developers to write application components by focusing on their functional effect ("business logic"), without having to explicitly code for transactions, concurrency, and persistence. Several facilities—container managed persistence, stateful session bean passivation, HttpSession management—are provided which, properly exploited, can and should eliminate the need for applications to explicitly code for the temporary storage of data on queues.

The rest of this document describes the JMS technologies that IBM offers. We begin by looking at the standalone messaging products, WebSphere MQ and WebSphere MQ Event Broker. Next we look at the built-in JMS Provider included in WebSphere Application Server Version 5. We conclude by looking at the use of WebSphere MQ and WebSphere MQ Event Broker with WebSphere Application Server Version 5.

2 The WebSphere MQ Products

WebSphere MQ Version 5.3 is the current release of IBM's market-leading standalone messaging product (formerly called "MQSeries"), used for point-to-point messaging from Java, C, C++, and COBOL applications, and on over 35 hardware platforms. It allows Java applications to use JMS to exchange messages with Java or non-Java applications that use WebSphere MQ's own API, called "MQI". Three-quarters of customers who buy inter-application messaging systems buy WebSphere MQ, and in the largest deployment, over 250 million messages a day are transmitted. WebSphere MQ (sometimes called just "MQ") provides support for both persistent and non-persistent messaging. With persistent messaging, messages can be recovered following a system failure; once an application has sent the message, the message can only be lost if the disk on which it is stored is damaged or lost and there are no backups or mirrored copies. With non-persistent messages, message recovery following failure is not possible, but message throughput is considerably higher. In a typical point-to-point configuration, the sending and receiving application each connect to their own queue manager, and the two queue managers are interconnected by means of a *message channel*. The sending application first puts the message to its local queue manager, the message

is then carried over the message channel to the receiving application, and finally the receiving application retrieves the message.

WebSphere MQ Event Broker adds support for publish/subscribe messaging. The job of a broker is to match published messages to subscriptions. A broker works in conjunction with a "transport", which is used to pass the messages from the publishing application, through the broker itself, to the subscribers. WebSphere MQ Event Broker (or just "Event Broker") offers two transports for use with J2EE applications, to give a wide range of quality-of-service and performance options.[1] For fastest performance of non-persistent messaging, Event Broker has a custom transport, where each application makes a direct TCP connection to the broker, and "non-persistent" is interpreted as "best efforts", consistent with vendors' JMS products. For more reliability, WebSphere MQ message channels can be used as the transport; a copy of WebSphere MQ for use with WebSphere MQ Event Broker is included with the product. In this case, "non-persistent", is interpreted such that every effort is made not to lose messages short of persisting them so as to be recoverable following system failure. The MQ transport must always be used for persistent messaging.

Finally, WebSphere MQ Integrator Broker is IBM's high-end broker product. Like WebSphere MQ Event Broker, it adds publish/subscribe function on top of WebSphere MQ. However, it also provides a rich environment for the transformation and routing of messages, including support for a wide range of XML and non-XML message formats, and the integration of database access into message flows.

3 WebSphere Application Server Version 5 Embedded Messaging

WebSphere Application Server Version 5 includes a built-in JMS Provider (called simply the "WebSphere JMS Provider"), and this feature of WebSphere Application Server is known as "WebSphere Embedded Messaging". The purpose of WebSphere Embedded Messaging is to enable Application Server users to exploit JMS without having to obtain separate messaging technology and integrate it into the Application Server environment. Embedded Messaging is built using technology from WebSphere MQ and WebSphere MQ Event Broker. For point-to-point messaging, the underlying transport is the same as that used by the WebSphere MQ product, and for publish/subscribe, support for both the MQ and direct TCP forms of connection between application and broker is included. However, Embedded Messaging has a number of important differences from the standalone products:

- Whereas the WebSphere MQ products provide separate "queue manager" and "broker" components, Embedded Messaging provides a single server component (the "Embedded Messaging Server"), accessed via a single integrated client component (the "Embedded Messaging Client"). The two components are each optional parts of a WebSphere Application Server installation.

- Embedded Messaging is administered using the WebSphere Administrative Console, and other WebSphere Application Server administration tools. WebSphere MQ tools (such as MQ Explorer) must not be used with WebSphere Embedded Messaging.[2]

[1] This is in addition to the MQe transport, intended for mobile devices, and the SCADA transport, intended for remote telemtry devices. By using WebSphere MQ Event Broker for publish/subscribe, J2EE applications can communicate with any MQe or SCADA-enabled device.

[2] It is technically possible to discover the MQ queue manager which underlies an Embedded JMS Provider instance, and to operate on it with MQ Explorer and other MQ tools. However, doing so is highly likely to put

- Embedded Messaging is intended for use with the Web, EJB, and Client Containers of WebSphere Application Server Version 5. As with other WebSphere Application Server resources, it can also be used by "thin client" Java applications. It does not interoperate with WebSphere MQ or WebSphere MQ Event Broker, and non-Java applications or Java applications that do not have access to WebSphere Application Server resources via JNDI, cannot connect to the WebSphere JMS Provider.

- Embedded Messaging provides a specific messaging topology: If the Embedded Messaging Server is installed onto a node, then that node will host a single JMS Server. For two WebSphere Applications to be able to communicate using JMS, they must be running in the same "cell", which is the name given to a cluster of Application Server instances that are administered together. In order to exchange JMS messages, the two applications must both connect to the same JMS Server instance, i.e. they must both choose the same node whose JMS Server they are going to use. Each node that hosts applications using JMS must have the Embedded Messaging Client installed; only nodes hosting a JMS Server required the Embedded Messaging Server component.

- Embedded Messaging is integrated with WebSphere Application Server security. Userids are authenticated, using the password supplied by the application or administrative configuration, against the user registry that the Application Server has been configured to use (for which there are several options, including the underlying operating system and LDAP). Access to queues (for point-to-point messaging) and topics (for publish/subscribe) is controlled from the Application Server using an XML-format JMS authorizations file.

- The components of the Embedded Messaging JMS Server are integrated so as to form a single logical runtime server. So, for example, starting the JMS Server causes all the runtime components associated with the underlying MQ technology to be started automatically. The JMS Server is capable of both point-to-point and publish/subscribe messaging, and has no visible separation into queue manager and broker components.

An important consequence of the fact that communicating applications must use the same JMS Server is that much of the time, those applications will be making remote connections. Previously, WebSphere MQ could only support transactional (or "XA") connections from applications running locally to the queue manager. Available for the first time in WebSphere Application Server Version 5, the WebSphere MQ technology which comprises WebSphere Embedded Messaging has been enhanced to support remote transactional connections, so that applications may connect remotely to a JMS Server without reduction in quality-of-service.

WebSphere Embedded Messaging is recommended for use as in functional verification test, system test, and small-scale pilot production deployments. In these environments, the integration of the messaging technology into the application server means the administrative and operational cost of exploiting messaging for J2EE applications is minimized. For larger production deployments, where throughput or availability considerations suggest the use of a more complex messaging topology, then the use of the WebSphere MQ and/or WebSphere MQ Event Broker is recommended. The MQ products are also required if messaging between J2EE and non-J2EE applications is to be used.

---

the queue manager into a state which is inconsistent with the view of it that the Application Server has, which can result in undelivered messages.

4 Using WebSphere MQ and WebSphere MQ Event Broker with WebSphere Application Server Version 5

By choosing to use WebSphere MQ and/or WebSphere MQ Event Broker with WebSphere Application Server you are able to exploit all of the features of those products with your J2EE applications; the cost is that you must separately configure the messaging system itself, using the MQ and Event Broker tools (for example runmqsc or MQ Explorer), and the connections between the Application Server and MQ or Event Broker (for example using the WebSphere Administrative Console). Within the Application Server tools, MQ and Event Broker are referred to collectively as the "WebSphere MQ JMS Provider".[3]

To use the "MQ Provider", you install and configure either WebSphere MQ or WebSphere MQ Event Broker, and then configure the Application Server to connect to it instead of the Embedded Provider. The MQ products may be installed onto the same machine as the Application Server, or on a different machine. It is not necessary to separately install the MQ JMS client into the Application Server environment; the Embedded Messaging Client component of the Application Server install provides all that is needed to make transactional connections both to the Embedded Provider and the MQ Provider.[4]

If your J2EE applications are using only point-to-point messaging, then WebSphere MQ is sufficient; if they are using publish/subscribe, then WebSphere MQ Event Broker is needed. WebSphere Application Server Version 5 Enterprise includes a copy of WebSphere MQ (but not WebSphere MQ Event Broker) with licencing restricting its use to WebSphere Application Server environments; for all other cases the appropriate WebSphere MQ or WebSphere MQ Event Broker licence entitlement for the messaging server in use must be obtained.

Here are some examples of requirements for which the use of the additional capabilities of WebSphere MQ, WebSphere MQ Event Broker, or WebSphere MQ Integrator Broker would be appropriate:

- A requirement to connect applications running in WebSphere Application Server with applications that use a wide selection of other language environments, runtime environments, and/or hardware platforms, or to connect to a large range of packaged applications that have either native a MQ interface, or for which an adapter is available.
- A requirement to support high message volumes (measured as a function of both message size and number of messages). With WebSphere MQ, Queue Manager Clustering can be used to distribute messaging workload across multiple queue managers.
- A requirement to decouple sending and receiving application environments, both from one another and from the underlying network that provides connectivity between them. WebSphere MQ message channels can allow the sending application to continue processing when the receiving application or its hardware is unavailable, and vice versa, and both applications may be able to continue operating when the network link is down.
- A requirement to support a large number of independent subscribers. With WebSphere MQ Event Broker, multiple brokers can be interconnected so as to form a graph structure. This allows publications to be "fanned out" and distributed across an arbitrarily large number of subscribing applications.

[3] WebSphere Application Server also allows you to configure a "Generic Provider". This is primarily intended for configuring WebSphere Application Server to work with JMS Providers from other vendors.

[4] For non-Application Server environments, remote transactional connections require the installation (and licensing) of a special WebSphere MQ component; without this component, only local connections or non-transactional remote connections can be made. For applications running in WebSphere Version 5 Application Server, this component is part of the Embedded Messaging Client, and no additional MQ client install (or licensing) is necessary.

- A requirement to reuse existing WebSphere MQ or Event Broker infrastructure. Many organizations already have an MQ network in place, and may wish to reuse their existing investment. For example, if an MQ queue manager has been configured in an HA environment (such as HACMP or Microsoft Cluster Server), and has spare capacity, then applications running in WebSphere Application Server can also connect to this queue manager and benefit from its high availability. Or, the existing MQ infrastructure may have monitoring and management tools and procedures in place, that are just as relevant to J2EE applications as to existing MQ applications.

5 Tooling Support: Messaging in WebSphere Studio Application Developer

WebSphere Studio Application Developer Version 5.0 is the "build-time" counterpart to WebSphere Application Server Version 5.0: it provides the tooling necessary to develop, deploy, and test WebSphere Application Server applications. Applications are tested within WebSphere Studio by running a copy of the Application Server within the WebSphere Studio IDE, and all deployment and configuration can be performed from within the WebSphere Studio GUI.

When testing applications that use JMS messaging, there are a number of options:

- The copy of WebSphere Application Server running within the WebSphere Studio Application Developer IDE can be configured to run its own Embedded Messaging JMS Server.
- Applications running in the Application Server inside WebSphere Studio Application Developer can connect to an Embedded Messaging JMS Server associated with another installation of WebSphere Application Server, usually on a different machine. This enables the same technique as is employed with applications that use a database: one machine is installed with a database that is dedicated to development test, to avoid having to install and operate the database on each developer workstation.
- Applications running in the Applications Server inside WebSphere Studio Application Developer can connect to a WebSphere MQ JMS Provider, just as if the Application Server were running outside of WebSphere Studio.
- WebSphere Studio Application Developer can be configured to run a special JMS Provider called "MQ for Java Developers". This is really a special case of the first option, where instead of running the normal Embedded Messaging JMS Server, a simulator for the JMS Server is executed instead. This simulator implements all of the JMS APIs, with the same functional behaviour, including both point-to-point and publish/subscribe, and the same error codes, as the regular Embedded Provider. However, it does not actually implement persistence, recovery logging for transactions, or security checks. MQ for Java Developers is ideal for unit testing of application components, as it allows the Application Server to be started and stopped very quickly—important when making lots of configuration changes early in the development cycle of an application. It should obviously not be used for system testing or production.

6 Conclusion

IBM provides a range of options for JMS Messaging in a J2EE environment, following the "start simple, grow big" philosophy. For development and unit test, MQ for Java Developers, part of

WebSphere Studio Application Developer, improves productivity by speeding the build-deploy-test cycle. For system test and small-scale pilot production deployments, WebSphere Embedded Messaging, part of WebSphere Application Server, provides a full JMS implementation, built using tried-and-tested technology from the WebSphere MQ family, but fully integrated with Application Server administration and security. For large-scale production, WebSphere MQ is the most widely-trusted messaging technology, with many high-volume deployments around the world. WebSphere MQ also allows your J2EE application to connect to the widest range of other platforms, environements, and applications. For publish/subscribe, the WebSphere MQ family offers WebSphere MQ Event Broker, enabling you to reuse your MQ transport infrastructure, or to use a custom TCP transport optimized specifically for highly-scalable lightweight publish/subscribe.

More information about the features and capabilities of WebSphere Application Server Version 5 can be found at http://www-3.ibm.com/software/webservers/appserv/appserv_v5.html. More information about the WebSphere MQ products can be found at http://www-3.ibm.com/software/ts/mqseries/.