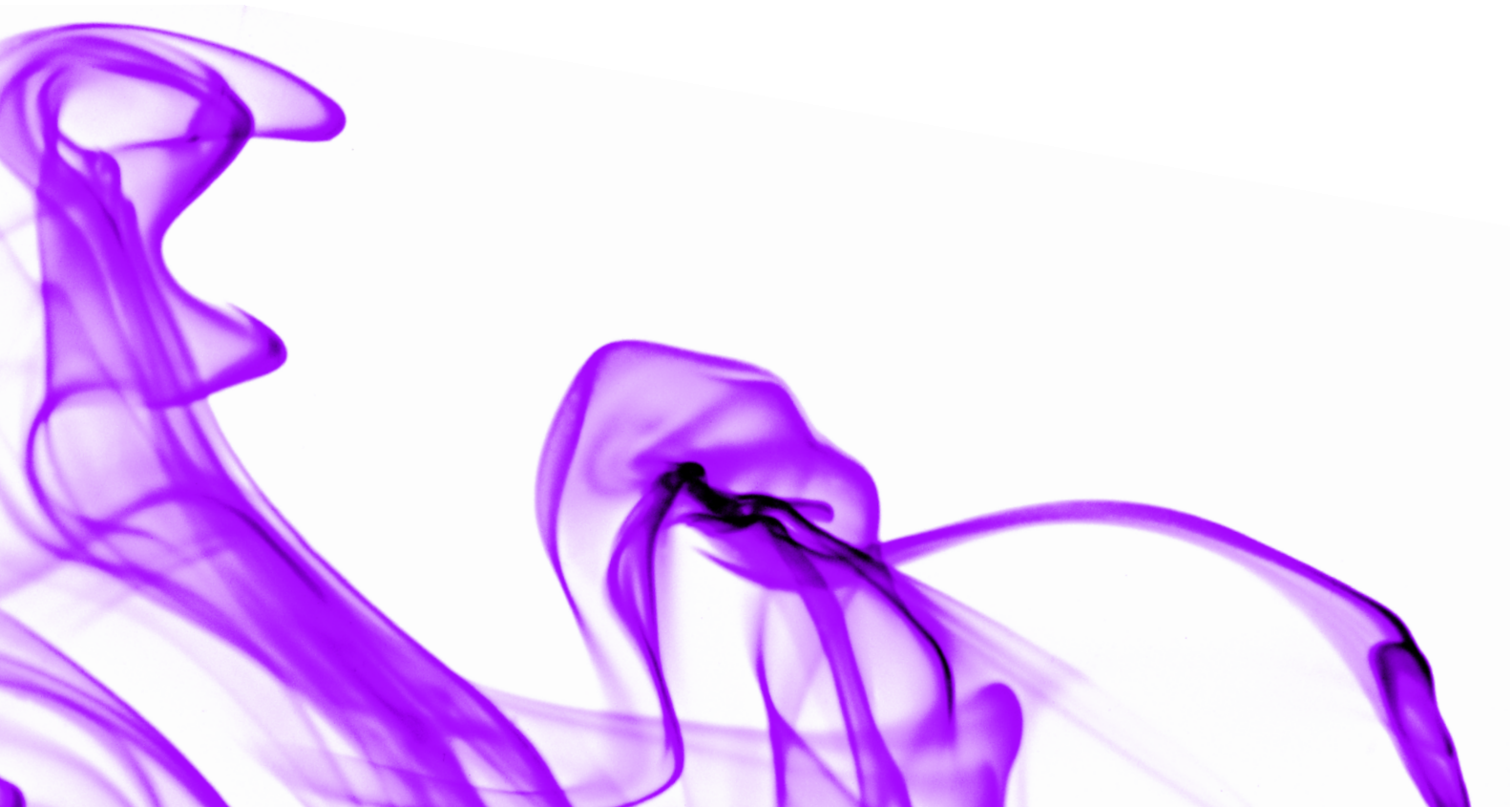




BEA White Paper

BEA AquaLogic™ Service Bus and WebSphere MQ in Service-Oriented Architectures

Integrating a Clustered BEA AquaLogic Service
Bus Domain with a Clustered IBM WebSphere MQ



Copyright

Copyright © 1995–2006 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software is protected by copyright, and may be protected by patent laws. No copying or other use of this software is permitted unless you have entered into a license agreement with BEA authorizing such use. This document is protected by copyright and may not be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form, in whole or in part, without prior consent, in writing, from BEA Systems, Inc. Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE DOCUMENTATION IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA SYSTEMS DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE DOCUMENT IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks and Service Marks

Copyright © 1995–2006 BEA Systems, Inc. All Rights Reserved. BEA, BEA JRockit, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop, Built on BEA, Jolt, JoltBeans, SteelThread, Top End, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA AquaLogic, BEA AquaLogic Data Services Platform, BEA AquaLogic Enterprise Security, BEA AquaLogic Service Bus, BEA AquaLogic Service Registry, BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA MessageQ, BEA WebLogic Commerce Server, BEA WebLogic Communications Platform, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Enterprise Security, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Java Adapter for Mainframe, BEA WebLogic JDriver, BEA WebLogic Log Central, BEA WebLogic Network Gatekeeper, BEA WebLogic Personalization Server, BEA WebLogic Personal Messaging API, BEA WebLogic Platform, BEA WebLogic Portlets for Groupware Integration, BEA WebLogic Server Process Edition, BEA WebLogic SIP Server, BEA WebLogic WorkGroup Edition, Dev2Dev, Liquid Computing, and Think Liquid are trademarks of BEA Systems, Inc. BEA Mission Critical Support, BEA Mission Critical Support Continuum, and BEA SOA Self Assessment are service marks of BEA Systems, Inc. All other names and marks are property of their respective owners.

Table of Contents

Introduction	1
Background	1
How to establish messaging communication between BEA AquaLogic Service Bus and WebSphere MQ	2
Design and configuration	2
Distributed transactional integrity between BEA AquaLogic Service Bus and WebSphere MQ	2
BEA AquaLogic Service Bus distributed transactional support	3
Clustered MQ setting	3
Setting up and configuring transparent messaging request/response communication between the clustered BEA AquaLogic Service Bus JMS and clustered WebSphere MQ	4
Sample installation, set-up, and configuration of a WebSphere MQ server and an extended transactional client	5
Configuring JMS—Foreign JMS providers	8
Configuring JMS—WebSphere MQ JMS	9
Configuring JMS—BEA AquaLogic Service Bus JMS	10
Testing the system	13
Conclusion	15
Appendix: References and related documents	16
About BEA	17

Introduction

Service-Oriented Architecture (SOA) has become a widely accepted paradigm in the industry. The increased demand for asynchronous exchange of business data has driven the need for SOAs. Different messaging middleware employed by enterprises means that BEA needs to provide a way to integrate this middleware with the BEA family of products. In particular, customers should be able to use the integrated BEA AquaLogic Service Bus/WebSphere MQ system to provide support for messaging interoperability, load balancing, high availability, high performance, failover, and exactly-once quality-of-messaging service (with regard to 'request message').

This paper discusses the setup and configuration of clustered BEA AquaLogic Service Bus 2.1 and WebSphere MQ (version 5.3 and higher) with Extended Transactional Client. It focuses on distributed transactional request/response messaging communication between these products. The example provided allows customers to achieve load balancing and failover in a clustered BEA AquaLogic Service Bus domain, as well as load balancing in an IBM WebSphere MQ cluster.

Background

BEA AquaLogic Service Bus Java Messaging Service (JMS) is an enterprise-class messaging system that fully supports the JMS specification, and provides numerous extensions to the standard JMS APIs. It is tightly integrated into the BEA AquaLogic Service Bus and BEA WebLogic Server® so customers can build highly secure Java 2 Enterprise Edition (J2EE) applications that are easily monitored and administered through the BEA AquaLogic Service Bus console. In addition to fully supporting extended architecture (XA) transactions, BEA AquaLogic Service Bus JMS features high availability through its clustering and service migration features, and also provides seamless interoperability with other versions of BEA AquaLogic Service Bus and third-party messaging vendors.

When deciding to implement the integrated system, it is worth weighing the trade-offs between using a BEA AquaLogic Service Bus/WebSphere MQ messaging conglomerate and using BEA AquaLogic Service Bus JMS. Usually the main reason for using WebSphere MQ with BEA AquaLogic Service Bus is to support a legacy messaging system that is already in place and not easily replaceable.

Customers may instead want to use the BEA AquaLogic Service Bus JMS, since it is easy to use, configure, and monitor, and it provides performance and features that meet or exceed those of other JMS vendors. Using BEA AquaLogic Service Bus JMS offers the following benefits:

- It runs as an integral part of BEA WebLogic Server, whereas a foreign JMS provider must be configured, started, stopped, and monitored using a separate set of tools.
- It is integrated with the built-in Java Naming and Directory Interface (JNDI) and clustering support of BEA WebLogic Server, so there is no need to configure and manage a separate JNDI infrastructure, whether it is a collection of files or a Lightweight Directory Access Protocol (LDAP) server.
- It provides strict conformance with the JMS specification, so that applications may be written in a portable way.

- It provides excellent performance and scalability.
- It provides more efficient transaction management for JMS. When a foreign JMS provider is used, both BEA AquaLogic Service Bus and the foreign provider must undergo the extra overhead of two-phase commit between BEA AquaLogic Service Bus and the JMS provider.

How to establish messaging communication between BEA AquaLogic Service Bus and WebSphere MQ

The goal is to connect the WebSphere MQ messaging system to BEA AquaLogic Service Bus using the WebSphere MQ JMS interface. The user will then typically use the WebSphere MQ-native protocol for business processes. Distributed transactional integrity, reliability, scalability, and failover need to be ensured in the integrated system, and the use of clustered BEA AquaLogic Service Bus and MQ architectures should be optimized.

Design and configuration of clustered BEA AquaLogic Service Bus and WebSphere MQ

The following sections of this paper outline the features directly related to the BEA AquaLogic Service Bus/WebSphere MQ inter-cluster communication. For more information about clustering of BEA WebLogic Servers, see <http://e-docs.bea.com/wls/docs91/cluster/features.html>. For full documentation on using BEA WebLogic Server clusters, see <http://e-docs.bea.com/wls/docs91/cluster/index.html>, and full documentation on WebSphere MQ Queue Manager clusters is at <http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp>

Distributed transactional integrity between BEA AquaLogic Service Bus and WebSphere MQ

IBM WebSphere MQ version 5.3 and higher offer support for XA transaction management in conjunction with an external syncpoint coordinator. This support is available in both the JMS and C clients. The WebSphere MQ Extended Transactional Client lets applications participate within a unit of work with other local resource managers, under the control of an external syncpoint coordinator. Until the release of WebSphere MQ Extended Transactional Client in February 2003, this was only possible by using a local MQ server. For the purposes of pricing, the Transactional Client is treated as if it were an MQ server, while the current non-transactional client continues to be provided free of charge.

When a WebSphere MQ Extended Transactional Client is co-located with each BEA AquaLogic Service Bus cluster server, distributed transactional messaging is possible between BEA AquaLogic Service Bus and remote WebSphere MQ servers.

Terminology	Explanation
Resource manager	A computer subsystem that owns and manages resources that can be accessed and updated by applications (i.e., WMQ queue manager—queues are its resources, or DB2 database—tables are its resources).
Unit of work	When an application updates the resources of one or more resource managers, it is often vital that all updates complete successfully as a group, or none of them will complete. Updates that complete this way are said to occur within a “unit of work” or “transaction.”
Syncpoint coordinator	A syncpoint is the point in time when all updates within a unit of work are either committed or backed out. The computer subsystem that manages units of work is called a syncpoint coordinator.
Extended architecture (XA)	For a transaction manager to manage a unit of work, an architected interface to the resource manager must be present. One such interface is the XA interface, which is published by the X/Open Company Limited.

BEA AquaLogic Service Bus distributed transactional support

When Java transactional APIs are distributed via BEA WebLogic Server, transparent request/response messaging is enabled between clustered BEA AquaLogic Service Bus JMS and clustered WebSphere MQ.

Clustered WebSphere MQ setting

If a WebSphere MQ cluster node that serves as a remote message forwarder goes down and is restarted, it is possible to keep request/response messaging interoperability uninterrupted and transparent between clustered BEA AquaLogic Service Bus JMS and clustered WebSphere MQ, by using a clustered MQ setting. This is because the queue manager that holds the local references to shared queues stays alive. BEA AquaLogic Service Bus can still put messages to the request queue and get messages from the response queue. A business application could still get messages from the request queue and put messages to the response queue.

However, this is not the case if the queue manager that holds the local references to shared request and response queues goes down. In that case, messages can still be put to the queues, but not retrieved from them.

Setting up and configuring transparent messaging request/response communication between the clustered BEA AquaLogic Service Bus JMS and clustered WebSphere MQ

In general, the details of the installation and configuration depend on the operating system, the number of machines, and the desired cluster sizes of BEA AquaLogic Service Bus and WebSphere MQ. In a production environment, each BEA AquaLogic Service Bus cluster server is usually hosted on a separate machine. In addition, each WebSphere MQ cluster node may be hosted on a separate hardware system.

The following section offers an example set-up and configuration for transparent request/response messaging between clustered BEA AquaLogic Service Bus JMS and clustered WebSphere MQ. Minimal hardware is used—two systems, each running Microsoft Windows operating systems. The first machine hosts the BEA AquaLogic Service Bus domain and the WebSphere MQ Extended Transactional Client. The second machine hosts a cluster of WebSphere MQ servers.

The BEA AquaLogic Service Bus domain includes an administration server and a cluster of two managed servers. The MQ cluster includes two nodes. The example below outlines common configuration steps that can be extrapolated to suit production requirements.

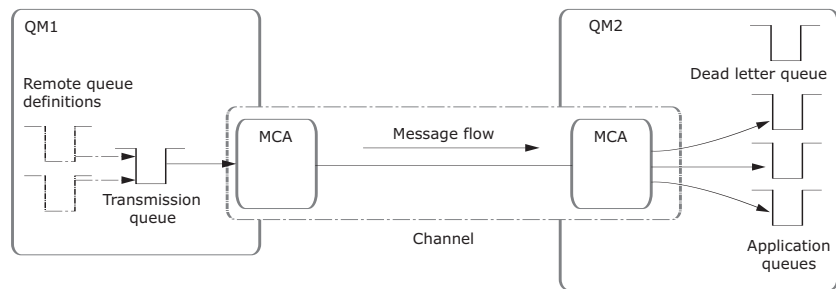
Sample installation, set-up, and configuration of a WebSphere MQ server and an Extended Transactional Client

To install and configure a WebSphere MQ server and an Extended Transactional Client, begin by installing WebSphere MQ 5.3 or 6.0. Follow the step-by-step installation instructions at the time of installation, and consult the IBM WebSphere MQ information center cited in the References section at the end of this document if needed.

Next, create two (or more) queue managers in WebSphere MQ. Setting queue managers as cluster members offers advantages over distributed queuing. When clustering is not used, the queue managers are independent and communicate using distributed queuing. In that case, if one queue manager needs to send messages to another it must define both a transmission queue, and a channel to the remote queue manager.

Figure 1 shows the components required for distributed queuing.

Figure 1
Distributed queuing.



However, if queue managers are grouped in a cluster, the queue managers can make the queues that they host available to every other queue manager in the cluster. Any queue manager can send a message to any other queue manager in the same cluster without explicit channel definitions, remote-queue definitions, or transmission queues for each destination. Every queue manager in a cluster has a single transmission queue from which it can transmit messages to any other queue manager in the cluster. Each queue manager in a cluster only needs to define one cluster-receiver channel on which to receive messages, and one cluster-sender channel with which it introduces itself and learns about the cluster. Figure 2 shows a small cluster of two queue managers.

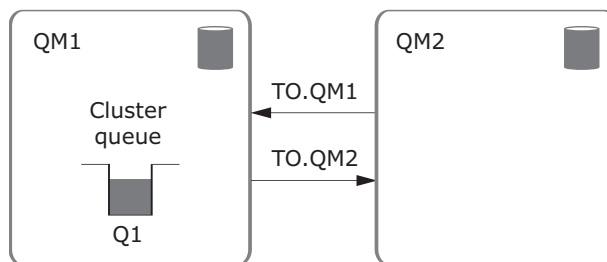
Next, create a cluster and include the queue managers as cluster nodes. Then specify both queue managers as hosting repositories.

The *full repository queue managers*, QM1 and QM2, host repositories of information about the queue managers in the cluster. (The repositories are represented in Figure 2 by the shaded cylinders.) QM1 hosts some queues that are accessible to any other queue manager in the cluster. These are called *cluster queues*. (The cluster queues are represented in Figure 2 by the shaded queues.)

As with distributed queuing, applications use the MQPUT call to put a message on a cluster queue at *any* queue manager. Applications use the MQGET call to retrieve messages from a cluster queue on the *local* queue manager.

Including queue managers in the cluster involves creation of system channels between them. Each queue manager has a definition for the receiving end of a channel called *TO.qmgr* on which it can receive messages. This is a *cluster-receiver channel*. A cluster-receiver channel is similar to a receiver channel used in distributed queuing, but in addition to carrying messages, this channel can also carry information about the cluster.

Figure 2
A small cluster of two queue managers.



Each queue manager also has a definition for the sending end of a channel, which connects to the cluster-receiver channel of one of the full repository queue managers. This is a *cluster-sender channel*. In Figure 2, QM1 and QM2 have cluster-sender channels that connect to TO.QM2. QM2 has a cluster-sender channel that connects to TO.QM1. A cluster-sender channel is similar to a sender channel used in distributed queuing, but in addition to carrying messages, this channel can also carry information about the cluster.

Once both the cluster-receiver end and the cluster-sender end of a channel have been defined, the channel starts automatically.

On Windows systems, creating queue managers and a cluster, and including queue managers in the cluster, can be done from the WebSphere MQ Explorer interface or by using the command line interface tool (`runmqsc`) found in the `.../WebSphere MQ/bin` directory. On UNIX systems, use the command line interface tool `runmqsc`.

Example for the Windows operating system:

```
>runmqsc.exe QM_IR02.1416_ALSB1
ALTER QMGR REPOS(ALSB1)

DEFINE CHANNEL(TO.QM_IR02.1416_ALSB1) CHLTYPE(CLUSRCVR)
TRPTYPE(TCP) CONNAME('ir02.bea.com(1416)') CLUSTER(ALSB1)

DEFINE CHANNEL(TO.QM_IR02.1417_ALSB1) CHLTYPE(CLUSSDR)
TRPTYPE(TCP) CONNAME('ir02.bea.com(1417)') CLUSTER(ALSB1)

>runmqsc.exe QM_IR02.1417_ALSB1
ALTER QMGR REPOS(ALSB1)
```

```
DEFINE CHANNEL(TO.QM_IR02.1417_ALSB1) CHLTYPE(CLUSRCVR)
TRPTYPE(TCP) CONNAME('ir02.bea.com(1417)') CLUSTER(ALSB1)

DEFINE CHANNEL(TO.QM_IR02.1416_ALSB1) CHLTYPE(CLUSSDR)
TRPTYPE(TCP) CONNAME('ir02.bea.com(1416)') CLUSTER(ALSB1)
```

The next step is to create two local queues (request and response) on one of the queue managers using the WebSphere MQ Explorer interface or `runmqsc`, and share them on the cluster.

Example:

```
DEFINE QLOCAL (REQUESTQUEUE1) CLUSTER (ALSB1)

DEFINE QLOCAL (RESPONSEQUEUE1) CLUSTER (ALSB1)
```

Next, install MQ Extended Transactional Client 5.3 on a second machine. MQ Extended Transactional Client 5.3 comes as a separate installation package for version 5.3. It enables global transactions between BEA WebLogic Server and WebSphere MQ hosted on separate machines. The references section at the end of this document points to a complete explanation.

Then install the BEA AquaLogic Service Bus clustered domain on the same machine where MQ Extended Transactional Client 5.3 is installed.

The above set-up is somewhat simplified, because in production, each managed server will be on a separate machine and has to be co-located with its own installation of WebSphere MQ Extended Transactional Client.

Configuring JMS—Foreign JMS providers

To explain the configuration of JMS using JNDI, this paper uses information from the “Using Foreign JMS Providers with WebLogic Server” document, which can be found at <http://ftpna2.bea.com/pub/downloads/jmsproviders.pdf>.

A JMS client needs to look up an initial “Connection Factory” object and destination objects—queues and topics—using JNDI. Therefore, a JMS client can use a WebSphere MQ JMS provider when two key settings are implemented:

- JNDI binding of the MQ JMS objects using MQ command-line utility. Besides connection factories and destinations, the MQ JMS provider uses JNDI to store its own configuration information, such as the IP address and port of the server, type of transport, destination attributes, and other special options.
- Configuration of the “Foreign JMS provider” for MQ using the BEA WebLogic Server administration console.

This configuration supplies the following information to the BEA AquaLogic Service Bus server:

Initial context factory: This is the name of the JNDI class that will be created to perform the lookup. This JNDI class may be part of the JDK, it may be provided by Sun Microsystems, or it may be provided by the JMS vendor.

Provider URL: This URL tells the initial context factory where to find the directory information. It may refer to a server using a protocol such as LDAP, it may refer to a file in the file system, or to something else.

Connection factory JNDI name: This is the name of the connection factory object that is stored in JNDI. Each JMS provider creates its own tool to create this object and store it in JNDI. Once it is stored there, any JMS client program (including BEA AquaLogic Service Bus and BEA WebLogic Server) can look up the object.

Destination JNDI name: This object represents a JMS destination, a queue or a topic, as stored in JNDI. Like the connection factory, each JMS provider will provide its own tool to create the object and place it in JNDI.

Configuring JMS—WebSphere MQ JMS

To configure WebSphere MQ JMS, perform identical JNDI binding of the MQ JMS objects on both machines—with MQ server and MQ Extended Transactional Client installations. Alternatively, the shared data structure can be used.

First, select the JNDI service provider and the URL of the service provider's initial context by un-commenting them in the following file: `.../Java/bin/JMSAdmin.config`.

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.ReffSContextFactory
PROVIDER_URL=file:/C:/JNDI-Directory
```

Second, bind JMS objects into the JNDI tree using the JMS admin file `.../Java/bin/JMSAdmin.bat`. The bindings will be stored in the file `C:\JNDI-Directory\binding`. Namely:

- Bind two WebSphere MQ XA connection factories (one for each WebSphere MQ server node) indicating the name of the factory, the name of the queue manager, host name, port, and client transport. For example:

```
def xaqcf(XAQCF0) qmanager(QM_IR02.1416_ALSB1) HOSTNAME(ir02)
PORT(1416) TRANSPORT(CLIENT)

def xaqcf(XAQCF1) qmanager(QM_IR02.1417_ALSB1) HOSTNAME(ir02)
PORT(1417) TRANSPORT(CLIENT)
```

- Bind two MQ queues (request and response) indicating the name of the queue, and the name of the queue manager holding the local reference to the queue and persistence mode. If the client which de-queues the queue will be a native WebSphere MQ client, the `TARGCLIENT(MQ)` option must be specified. This option ensures that the client will be able to retrieve proper WebSphere MQ headers for the message. For example:

```
def Q(REQUESTQUEUE1) queue(REQUESTQUEUE1)
qmanager(QM_IR02.1416_ALSB1) PERSISTENCE(PERS) TARGCLIENT(MQ)

def Q(RESPONSEQUEUE1) queue(RESPONSEQUEUE1)
qmanager(QM_IR02.1416_ALSB1) PERSISTENCE(PERS)
```

Finally, to see the resulting WebSphere MQ JMS bindings, give the command *dis ctx*:

```
InitCtx> dis ctx

Contents of InitCtx

      .bindings                                java.io.File
a  RESPONSEQUEUE1                            com.ibm.mq.jms.MQQueue
a  REQUESTQUEUE1                             com.ibm.mq.jms.MQQueue
a  XAQCFO
com.ibm.mq.jms.MQXAQueueConnectionFactory
```

```
a  XAOCF1
com.ibm.mq.jms.MQXAQueueConnectionFactory
5 Object(s)
0 Context(s)
5 Binding(s), 4 Administered
```

Configuring JMS—BEA AquaLogic Service Bus JMS

The simplest setting is to configure the BEA AquaLogic Service Bus clustered domain in production mode with an Oracle database and with two co-located managed servers.

Edit `setDomainEnv.cmd` or `setDomainEnv.sh`.

ADD EXTENSION TO PRE_CLASSPATH.

For example:

```
set IBM_MQ_DIR=C:\Program Files\IBM\WebSphere MQ\Java\lib
set IBM_MQ_ETC_DIR=C:\Program
Files\IBM\Source\C48UVML\Windows\MSI\Program Files\IBM\WebSphere
MQ\Java\lib
set
EXT_PRE_CLASSPATH=%IBM_MQ_DIR%\com.ibm.mqjms.jar;%IBM_MQ_DIR%\fscontext.jar;
%IBM_MQ_DIR%\com.ibm.mq.jar;%IBM_MQ_DIR%\com.ibm.mqbind.
jar;%IBM_MQ_DIR%\providerutil.jar;%IBM_MQ_ETC_DIR%\com.ibm.mqetclient.jar;
%EXT_PRE_CLASSPATH%
```

Optionally, for demo purposes, to reduce the memory consumption, change the default memory settings. For example:

```
set MEM_ARGS=-Xms256m -Xmx256m
```

Start the administration and clustered (managed) servers.

To have all relevant JMS objects in a separate JMS system module, use the BEA WebLogic Server Administration Console to create a new JMS system module that is deployed on the cluster.

In this JMS system module, create:

- A foreign JMS server deployed on the cluster with two WebSphere MQ XA connection factories corresponding to two MQ cluster nodes—*XAQCF0* and *XAQCF1*—and two queues, the WebSphere MQ request and response queues *REQUESTQUEUE1* and *RESPONSEQUEUE1*
- An XA connection factory with default delivery mode set to persistent
- A proxy request Gateway Uniform Distributed Queue (UDQ) deployed on the cluster
- A proxy response UDQ deployed on the cluster.

Optionally, if using a POJO JMS client to send request messages for testing the system, create a local queue that is deployed on one of the managed servers as an entry point. The purpose behind creating the local queue on which the first proxy service will be deployed is to route the message to the second proxy service deployed on the UDQ. This is purely optional, but can be done to see all cluster servers that participate in sending messages to the WebSphere MQ server.

Next, verify that the JNDI panel in the BEA WebLogic Server Administration Console shows the JNDI tree with all foreign JMS server objects (MQ connection factories and queues) and the newly created BEA AquaLogic Service Bus JMS connection factory and queues.

Using the BEA AquaLogic Service Bus Console (sbconsole):

- Configure the first JMS proxy service with the local queue as JMS end point (MDB will be deployed on that local queue). This is optional if using POJO JMS client to send the initial message.
- Configure the second JMS proxy service with Gateway UDQ as JMS end point (MDB deployed on the Gateway UDQ). This JMS proxy service requires a response to the response UDQ.
- Configure the router from the first JMS proxy service to the second one.
- Configure the JMS business service deployed on the MQ request queue that requires response to the MQ response queue. This business service has two end points, each pointing to one of the two MQ queue managers for MQ cluster load balancing.
- Configure the pipeline between the second JMS proxy service and the JMS business service with the router that will route messages from the second JMS proxy service to the JMS business service.
- Configure the Request and Response Logging Actions for logging messaging headers. The logging will show how many request and response messages would be passing the request and response pipelines.

Testing the system

The next step is testing the newly integrated system. Verifying the messaging between the clustered BEA AquaLogic Service Bus and clustered WebSphere MQ with BEA AquaLogic Service Bus XA transactional managers and MQ XA resource manager requires:

- XA connection factories configured in WebSphere MQ and BEA AquaLogic Service Bus
- Persistent delivery mode
- A JMS client that sends a request message to the BEA AquaLogic Service Bus proxy service
- A business service based on WebSphere MQ APIs; it receives the request message and sends a response message
- A JMS client that receives the response message.

The business service can be written as a class, based on WebSphere MQ Java APIs; the service gets the request message from the request queue and puts the response message in the response queue. If the WebSphere MQ class is hosted remotely to the MQ server, a server connection channel and a corresponding client channel must be created on the MQ server to which the MQ client connects.

The WebSphere MQ class takes care of setting environment variables, initialization of WebSphere MQ resources, measuring the current depth of the request queue, looping through, and getting the messages from the request queue. It sets the correlation ID on each response message to the request correlation ID, puts the response messages in the response queue, and releases the WebSphere MQ resources.

If and when the common messaging request/response pattern is supported in the BEA AquaLogic Service Bus JMS request/response implementation, the WebSphere MQ class will be setting the correlation ID on the response message to the request message ID.

It is possible to write POJO JMS clients. One POJO JMS client will send the request (for example, an XML file) to the first proxy service. Each of the two additional JMS POJO clients will listen to one of the two members of the proxy service's response UDQ.

These are the settings:

- The client sends request messages to WebSphere MQ via BEA AquaLogic Service Bus. It uses the standard JMS API protocol, or enhanced WebLogic JMS APIs. A BEA AquaLogic Service Bus proxy service receives the client's messages and dispatches them to the business service deployed on WebSphere MQ.
- BEA AquaLogic Service Bus uses the underlying BEA WebLogic JMS, which is an enterprise-class messaging system that fully supports the JMS 1.1 specification and also provides important and useful extensions of the standard JMS APIs.
- The business service uses native WebSphere MQ as the transport protocol.
- A WebSphere MQ cluster queue was created by creating a local queue on one of the queue managers and sharing it with a second cluster member. The second cluster member will be a remote forwarder to the shared queue. WebSphere MQ is essentially a store-and-forward message-passing middleware protocol.

- Requests from BEA AquaLogic Service Bus can be load-balanced by sending messages in a round-robin fashion to the members of the cluster. However, getting the response message (de-queue the response queue which is shared on a cluster) is only possible by accessing the MQ server node that holds the local reference to the queue. In other words, when sending a request message, BEA AquaLogic Service Bus can iterate through a list of the JMS destinations. But it can receive a response message only from one JMS destination.

First, test the scenario in which all of the managed servers in the BEA AquaLogic Service Bus cluster are alive and connected, and all MQ cluster servers are also alive and connected.

- Messages can be sent directly to the second BEA AquaLogic Service Bus JMS proxy service deployed on Gateway UDQ if the sender is capable of distributing the messages between the members of the distributed queue. Otherwise, use a POJO JMS client to send the predefined amount of request messages (perhaps reading them from XML files) to the first JMS proxy service.
- The first JMS proxy service routes the request message to the second JMS proxy service.
- The second JMS proxy service is deployed on the Gateway UDQ and requires response to the response UDQ.
- The second JMS proxy service routes the request message to the JMS business service.
- The JMS business service is deployed on the MQ request queue and requires response to the MQ response queue.
- The JMS business service has two end points, each on one of the two MQ queue managers. The “round robin” algorithm is selected for the load balancing between these two end points.
- A Java client based on the WebSphere MQ APIs measures the current depth of the request queue and loops through the messages to get the request message from the request queue and put the response message in the response queue. As required for BEA AquaLogic Service Bus 2.1, it reads the request correlation ID as byte[] and sets it as byte[] correlation ID on the response message.
- To find out whether a traditional EAI messaging request/response pattern is supported, consult the documentation for BEA AquaLogic Service Bus 2.5. If so, in BEA AquaLogic Service Bus release 2.5, the business service application should read request message ID as byte[] and set it as byte[] correlation ID on the response message.
- A response MDB listens on the WebSphere MQ response queue, processes response messages, and puts them in the response pipeline to be delivered to the response UDQ.
- Two JMS POJO clients listen, each to one of the two members of the response UDQ. They receive messages, read their correlation ID, and log it along with the current count of the received messages.

Second, test the scenario in which all managed servers in the BEA AquaLogic Service Bus cluster are alive, but one of the MQ cluster servers is stopped and restarted. To implement it:

- Start the BEA AquaLogic Service Bus administration server and two managed servers.
- Stop one of the MQ queue managers of the cluster (on which local request and response queues were configured).
- Send a predefined number of messages (XML files) to the first JMS proxy service using POJO JMS client.

- Start the MQ queue manager that was stopped.
- Observe that all messages that were sent to MQ were successfully delivered to the MQ request queue—through the second MQ cluster node that stayed alive.
- The Java client based on MQ APIs measures the current depth of the request queue and loops through the messages to move them from the request queue to the response queue. It reads the request correlation ID as byte and sets it as byte[] correlation ID on the response message.
- A response MDB listens on the MQ response queue, processes response messages, and puts them in the response pipeline to be delivered to the proxy service response UDQ.
- Two JMS POJO clients each listen to one of the two members of the proxy service response UDQ. They receive messages, read their correlation ID, and log it and the current count of the received messages.

Third, test the scenario in which all managed servers in the BEA AquaLogic Service Bus cluster are alive, but the MQ cluster is stopped and restarted. To implement it:

- Start the BEA AquaLogic Service Bus administration server and two managed servers.
- Stop MQ services on the remote machine. Expect to receive repeated disconnect-related exceptions on both managed servers.
- Send a predefined number of messages (XML files) to the first JMS proxy service using a POJO JMS client.
- Start the MQ cluster.
- Observe that all messages that were sent to MQ were successfully delivered to the MQ request queue by BEA AquaLogic Service Bus.

Conclusion

This white paper describes an example of the configuration of the messaging communication between BEA AquaLogic Service Bus and WebSphere MQ queue manager clusters. It is an initial attempt to find the best way of configuring such a system; descriptions of real-life installations of WebSphere MQ clusters from field engineers would be much appreciated (so they can be reproduced at BEA). Please send descriptions to alsb-wp@bea.com.

If those configurations are significantly different from the type of WebSphere MQ cluster described in this paper, the additional information can be appended to the paper.

For the BEA AquaLogic Service Bus 2.1 release, users can configure request and response queues that are shared on the WebSphere MQ cluster and separate XA connection factories for each WebSphere MQ server, and send messages to those MQ servers in a round-robin manner. In this way, WebSphere MQ load balancing is driven by BEA AquaLogic Service Bus.

Users can verify that the request messages are delivered to different servers. When a message arrives to the WebSphere MQ server that holds the local reference at the cluster request queue, it stays there. When a message arrives at the remote forwarder, the remote forwarder immediately forwards it to the server that holds the local reference to the cluster request queue. Browsing that cluster queue from the local queue manager shows all messages that were sent to the cluster.

A JMS service in BEA AquaLogic Service Bus 2.1 can get a response message from only one destination—the cluster response queue on its local queue manager.

In a request/response XA transactional clustered BEA AquaLogic Service Bus/clustered WebSphere MQ scenario, messages are not lost when one of the WebSphere MQ cluster servers is shut down and restarted. In fact, if the server that is shut down is a remote forwarder, operations continue smoothly because messages continue to arrive to the local queue manager of the cluster request queue. In addition, the response messages are retrieved from its cluster response queue. The messages are not lost when the entire WebSphere MQ cluster is shut down and restarted.

Appendix: References and related documents

Messaging (JMS) for BEA WebLogic Server 9.1

<http://e-docs.bea.com/wls/docs91/messaging.html>

Configuring Foreign Servers

http://e-docs.bea.com/wls/docs91/ConsoleHelp/taskhelp/jms_modules/foreign_servers/ConfigureForeignServers.html

Accessing Foreign Server Providers

http://e-docs.bea.com/wls/docs91/jms_admin/advance_config.html#1075917

Messaging (JMS) for BEA WebLogic Server 9.0

<http://e-docs.bea.com/wls/docs90/messaging.html>

- Using Foreign JMS Providers with BEA WebLogic Server (white paper)

FAQ: Integrating Remote JMS Providers

<http://e-docs.bea.com/wls/docs81/faq/interop.html>

Using Foreign JMS Providers with BEA WebLogic Server

<http://ftpna2.bea.com/pub/downloads/jmsproviders.pdf>

Using BEA WebLogic Server Clusters

<http://e-docs.bea.com/wls/docs91/cluster/index.html>

Communications in a Cluster

<http://e-docs.bea.com/wls/docs91/cluster/features.html>

IBM WebSphere MQ Information Center

<http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp>

WebSphere MQ: Information Roadmap for All Users

<http://www-128.ibm.com/developerworks/websphere/zones/businessintegration/roadmaps/wmq/>

WebSphere MQ Queue Manager Clusters

<http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp>

IBM WebSphere MQ Extended Transactional Client

<http://www-306.ibm.com/software/integration/wmq/transclient.html>

Meet the Experts: Mark Taylor on WebSphere MQ

http://www-128.ibm.com/developerworks/websphere/library/techarticles/0302_taylor/taylor.html

About BEA

BEA Systems, Inc. (NASDAQ: BEAS) is a world leader in enterprise infrastructure software, providing standards-based platforms to accelerate the secure flow of information and services. BEA product lines—WebLogic®, Tuxedo®, JRockit®, and the new AquaLogic™ family of Service Infrastructure—help customers reduce IT complexity and successfully deploy Service-Oriented Architectures to improve business agility and efficiency. For more information please visit *bea.com*.



BEA Systems, Inc.
2315 North First Street
San Jose, CA 95131
+1.800.817.4232
+1.408.570.8000
bea.com

CWPxxxxE0306-1A