Stonebranch Solutions

Indesca for SOA: JMS Connector
Proof of Concept

indesca-soa-jms-poc

Document Name	Indesca for SOA: JMS Connector Proof of Concept
Document ID	indesca-soa-jms-poc
Products	Universal Command Agent for SOA

Stonebranch, Inc.
950 North Point Parkway, Suite 200
Alpharetta, GA 30005 USA
Tel: (678) 366-7887
Fax: (678) 366-7887

SAP Certified Integration

Contents

List of Figures

Concept

Using Stonebranch's Indesca for SOA : JMS Connector to send and receive messages with an MQ Infrastructure.

Challenge

To make Indesca for SOA : JMS Connector communicate with an MQ infrastructure; specifically, the ability to read and write MQ messages to MQ queues.

The business scenario related to the request involves a three-step job initiated out of TWS, described as follows:

1. JDA Batch Job – A shell script is executed in the JDA environment. After the shell script is complete the return code is passed back to TWS via Universal Command.
2. SAP via Messaging – Upon successful completion of step 1, a message is placed on an inbound MQ Series queue that is used as an event to trigger pre-processing by an MQ Message Broker Workflow prior to delivering an iDoc to the SAP environment. Once SAP has completed its processing, a message is placed on an outbound MQ Series queue that contains the return code of the MQ Message Broker Workflow.
3. Mainframe Batch Job - Upon successful completion of step 2, a batch job is run on the mainframe.

The basic challenge is that TWS does not have a way to execute step 2 of the described job. Thus, this Proof of Concept regarding the Universal Command Agent for SOA and its possible use in this scenario.

Solution

There are two possible solutions for this scenario:

1. Use the Universal Command Agent for SOA : MQ Connector.
   This would solve the communication challenge with the MQ Message Broker environment.
2. Use the Universal Command Agent for SOA : JMS Connector.
   The challenge is that JMS to MQ communication requires some specific knowledge about the MQ Series infrastructure and how it can work with JMS.

It was decided to implement a proof-of-concept to provide exact details on how this solution would work in a specific environment, utilizing the following Indesca components:

- Universal Command
- Universal Command Agent for SOA

POC Environment

The POC environment was set up using the following components:

- WebSphere MQ v6.0 installed on AIX 6.1
- Universal Command Manager v3.2 installed on MVS
- Universal Command Agent v3.2 installed on Linux
- Universal Command Agent for SOA : JMS Connector v3.2 on Linux
- IBM Client Jar Files for JMS to MQ Operation
- Quasar SOA Test Workbench

The deployment topology is illustrated in the following diagram.

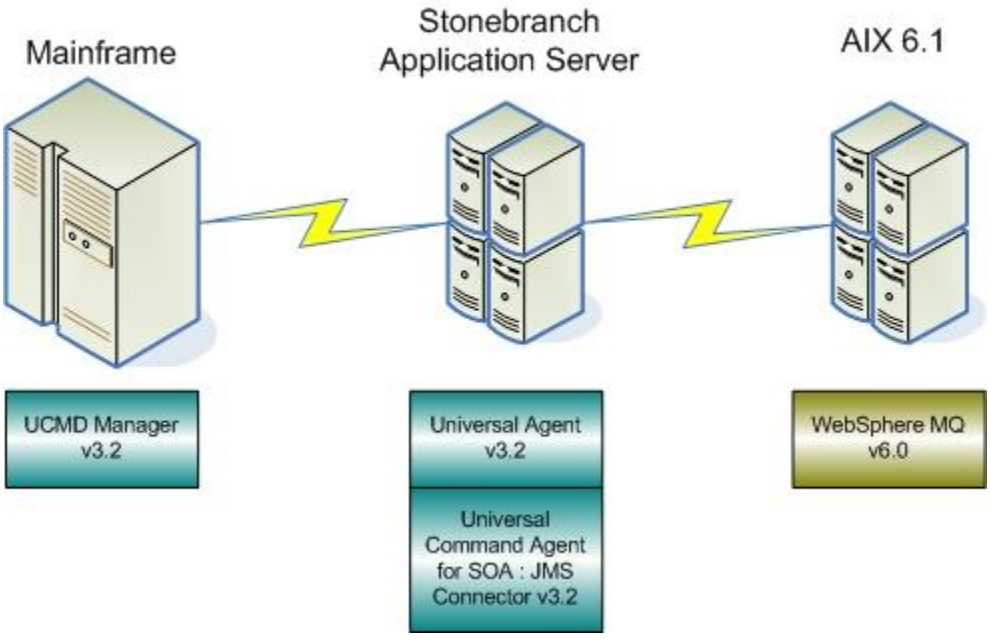

Figure 1 POC Deployment Topology

POC Process Workflow

The workflow for the POC includes the following steps:

1. A job is manually submitted on MVS that invokes Universal Command.
2. Universal Command validates request and sends it to the Universal Agent.
3. The Universal Agent starts a UCMD Server that submits the command options and payload for the JMS message to the Universal Command Agent for SOA : JMS Connector.
4. The JMS Connector connects to the MQ Broker in a synchronous, request/reply operation, and writes the message to the MQ Request queue.
5. The workflow process, in this case the Quasar SOA Test Workbench, reads the message from the MQ Request queue and writes a separate message to the MQ Reply queue. This represents the MQ Message Broker workflow that reads the message off the request queue, starts its processes, then places a message on the reply queue.
6. The JMS Connector reads the message from the MQ Reply queue and returns it back to the UCMD Server.
7. The UCMD Server passes the message back to Universal Command which writes it to UNVOUT where it is available for use by subsequent MVS jobs.

The POC workflow is illustrated in the following diagram.

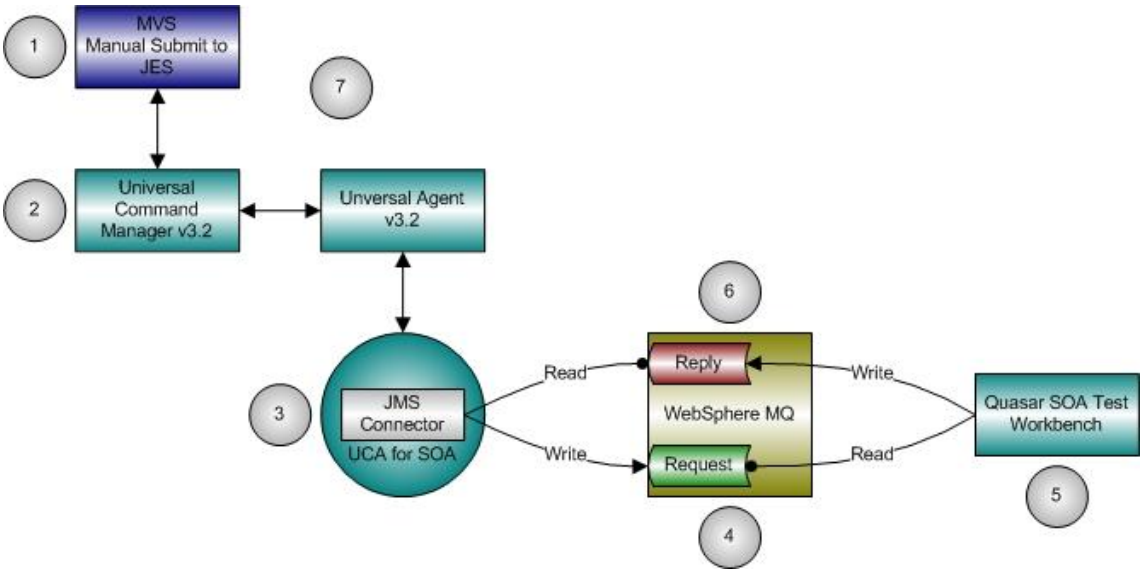

Figure 2 POC Workflow

POC Installation

The installation process for this POC is:

1. If not already installed, install WebSphere MQ v6.0 or v7.0. Please reference the WebSphere MQ installation guides.
2. Install Stonebranch Solutions . Note that the Universal Command Agent for SOA is available on three platforms (Linux, AIX, and Windows), so be sure to install Stonebranch Solutions on the same platform where you will be installing the Universal Command Agent for SOA. Please reference the Stonebranch Solutions Installation Guide.
3. Install Universal Command Agent for SOA on the same server that you installed the Stonebranch Solutions package. If you have not installed the Stonebranch Solutions package, then you will not be able to install the Universal Command Agent for SOA package, as there is a dependency.

   Please note that for both steps 1 and 2, you can deploy the packages to the server that the target workload is installed on (in this case WebSphere MQ) or you can choose an application server approach and install the two products on a Linux, AIX, or Windows server that has network access to both the Universal Command Manager host and the target workload.

MQ Setup and Configuration

You must configure the WebSphere MQ environment to support JMS operations. This involves defining queues and channels for the WebSphere MQ class for Java and defining the specific Java information for the WebSphere MQ environment.

This process involves two procedures:

1. Create Queues and Channels
2. Set up JMS using JMSAdmin

Create Queues and Channels

1. Create the queues and channels associated with the WebSphere MQ class for Java.

   Note: If you have existing queues you can skip this step, although you will need to define the java channel. Create a configuration file with a name of your choosing (MyQueueManager.conf works) and the following commands:

   DEFINE QLOCAL (‘MyRequestQ’) + REPLACE
   DEF CHL(‘JAVA.CHANNEL’) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   MCAUSER(‘mqm’) +
   DESCR(‘Sample channel for WebSphere MQ class for Java’) + REPLACE

   DEFINE QLOCAL (‘MyReplyQ’) + REPLACE
   DEF CHL(‘JAVA.CHANNEL’) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   MCAUSER(‘mqm’) +
   DESCR(‘Sample channel for WebSphere MQ class for Java’) + REPLACE

2. Once you have the file created, you need to run the following command from the prompt:
   runmqsc MyQueueManager < MyQueueManager.conf > qcreate.log

Set up JMS using JMSAdmin

Set up JMS using the JMSAdmin command line utility.

1. Modify the JMSAdmin.config in /usr/mqm/java/bin so that:

   INITIAL_CONTEXT_FACTORY=com.sun.jndi.fcontext.RefFSContextFactory
   PROVIDER_URL=file:/opt/tmp

   Note that the initial context factory variable exists and just needs to be uncommented. Be sure to comment out the default initial context factory value.

2. Make sure you are logged in as the mqm user.
   su – mqm
   cd /usr/mqm/java/bin

3. Run the jmsenvenv script at the prompt as follows (your syntax may be different and note the space between the dot and the script name):
   . setjmsenv

4. Run the JMSAdmin tool as follows (note that your syntax may be different):
   JMSAdmin

5. You will now have a new prompt associated with the JMSAdmin tool. At that prompt run the following commands (note that each def is it's own line and there are only three lines):
   def qcf(ConnectionFactory) hostname(yourhostname) port(1414) channel(JAVA.CHANNEL) transport(CLIENT) qmanager(MyQueueManager)
   def q(MyJMSRequestQ) queue(MyRequestQ) qmanager(MyQueueManager)
   def q(MyJMSReplyQ) queue(MyReplyQ) qmanager(MyQueueManager)

6. At the command prompt start the listener (don't forget to background the task):
   runmqlsr –m MyQueueManager –t tcp &

Universal Command Agent for SOA Set-up

You must configure the Universal Command Agent for SOA to work with WebSphere MQ. This includes copying a specific set of MQ jar files, copying the MQ .bindings file, setting up the properties file for the reply-to address, and starting the components.

The process is:

1. Copy the following list of jar files from /usr/mqm/java/lib to /opt/universal/uac/container/webapps/axis2/WEB-INF/lib. Verify the file sizes once the copy is complete. If there are any differences, recopy the files or things will not go so well in subsequent steps.
   - com.ibm.mq.jar
   - com.ibm.mqjms.jar
   - commonservices.jar
   - connector.jar
   - dhbcore.jar
   - fscontext.jar
   - jms.jar
   - jta.jar
   - providerutil.jar
2. Copy the .bindings from /opt/tmp from the WebSphere MQ host to /opt/universal/uai. This is a hidden file so you will need to use the command ls –lsa to see the file. Note that this file was generated when you ran the JMSAdmin utility based on the PROVIDER_URL=/opt/tmp option.
3. In order for the Universal Command Agent for SOA to retrieve the reply, it needs to know what the name of the reply-to queue is. This is handled by creating a simple XML file with the reply-to queue value in it and specifying the path and filename of this file using the –jmspropertiesfile option.

   Here is an example of this file:

```
<?xml version="1.0" encoding="UTF-8"?>
<sb:JMSProperties
xmlns:sb="http://com.stonebranch/UAC/JMSProperties"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://com.stonebranch/UAC/JMSProperties
JMSProperties.xsd ">
<sb:Property>
<sb:Name>jms.header.JMSReplyTo</sb:Name>
<sb:Value>MyJMSReplyQ</sb:Value>
</sb:Property>
</sb:JMSProperties>
```

4. Start the Universal Broker. This will start the Universal Command Agent for SOA. Use uquery to validate that the Universal Broker and Universal Command Agent for SOA have started successfully.

Running the Job

Now you can run the job. Create the control JCL, script, and options file for the host you are using.

The following examples are provided, which should work on most MVS/zOS hosts.

Control JCL UACJMS1

```
//UACJMS1  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//*************************************************************
//*MQ queue test for Publish
//*UCMD is the proc that calls UC Manager
//*LOGON is the DD with userid and passwd (can use encrypted)
//*SCR is the script that contains the JMSConnector information
//* to connect to Websphere job scheduler
//*UNVIN provides the payload for the SCRIPT in SCR
//*************************************************************
//*
//*        JCLLIB ORDER=SBI.UNV.SUNVSAMP
//*
//UCMD     EXEC UCMDPRC
//LOGON    DD  DISP=SHR,DSN=SUPPORT.UAC.LOGON(USER1)
//SCR      DD  DISP=SHR,DSN=SUPPORT.UAC.SCRIPTS(SCRJMSRR)
//UNVIN    DD  DISP=SHR,DSN=SUPPORT.UAC.SCRIPTS(PYLJMS1)
//UNVOUT   DD  DISP=SHR,DSN=SUPPORT.UAC.JOBLOGS(JOB1)
//UNVERR   DD  DISP=SHR,DSN=SUPPORT.UAC.JOBLOGS(JOB2)
//SYSIN    DD  *
-s scr
-script_type SERVICE
-i 123.45.67.890 -f logon
```

Script SCRJMSRR

```
This script contains the options that will be passed to the
Universal Command Agent for SOA.

# protocol is what protocol you are choosing (JMS)

# mep is the type of message.  In this case Request for a
request/reply operation

# serviceurl is the path to the .bindings file in this example

# jmsdestination is the Queue alias setup in step 5 of the MQ setup
section

# jmsconnectionfactoryname is connection class

# jmscontextfactoryname is classname of initial context

# jmspropertiesfiles is the path and name to the file that contains
the reply-to

# queue information.

#*********************************************

-protocol JMS

-mep Request

-serviceurl file:///opt/universal/uai

-jmsdestination MyJMSRequestQ

-jmsconnectionfactoryname ConnectionFactory

-jmscontextfactoryname com.sun.jndi.fscontext.RefFSContextFactory

-jmspropertiesfile /opt/universal/uai/MQReply.properties.xml
```

Payload PYLJMS1

If the message being placed on the MQ request queue is only acting as an event then technically you don't need to include a payload file. If there is data required by the MQ Message Broker workflow then you will need to create a payload file whose contents will then be attached to the JMS message. This can be any type of text based file such as XML, csv, or plain text. See the Universal Command Agent for SOA Reference Guide for reference. The following example shows a plain text payload file.

```
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000001 Test of configuration for POC
```

950 North Point Parkway, Suite 200
Alpharetta, Georgia 30005
U.S.A.