WebSphere Message Broker

SupportPac IAM9
WebSphere Decision Service Node

Jochen Benke, IT-Specialist, Software Services for WebSphere (ISSW)

Take Note!
Before using this report, be sure to read the general information under "Notices".
First Edition, February 2012

This edition applies to WebSphere Message Broker V7.0 and V8.0 – WebSphere Decision Service Node V1.1 and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

Notices

Trademarks and service marks

Document Change History

Document Number Date	**Date**	**Description of Changes**
1.0.0	02/17/11	Initial Release
1.1.0	04/25/11	• Support WODM v8, WODM v7.5 as well as ILOG BRMS v7.1 • New input/output document structure • XSD to support input/output document validation in message flows

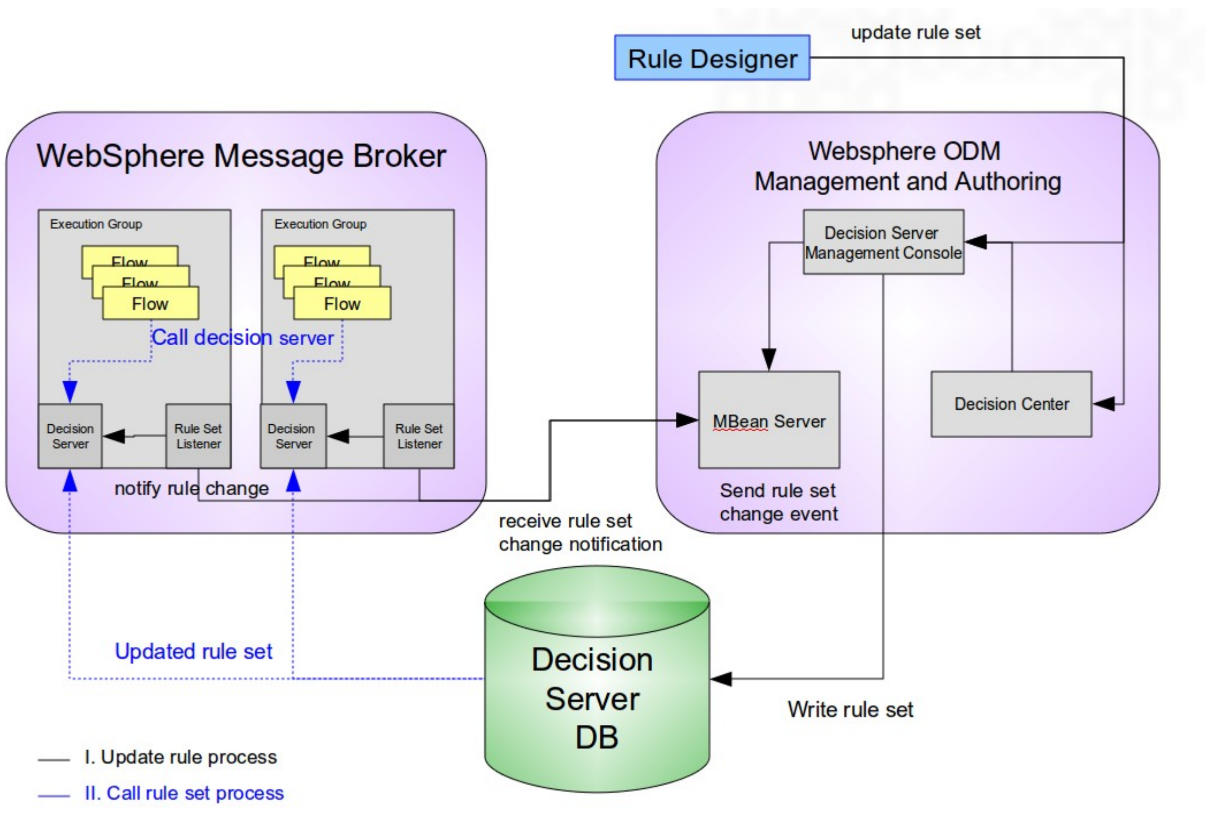
1.Introduction

The WebSphere Decision Service Node in WebSphere Message Broker runs a J2SE runtime business rules engine which can process rules based on the business content from messages within message broker flows. The runtime rules engine can come from the WebSphere Operational Decision Management (WODM) v7.5 product, or its previous version called WebSphere ILOG BRMS v7.1. The major benefits of doing this is it provides a rule authoring language understandable to both Business and IT users, runs tightly coupled for high performance and reliability and provides an additional choice to all WebSphere Message Broker flow developers that they can author business rules in ODM as an alternative to ESQL.

WebSphere Operational Decision Management (WODM), has 3 major components.

- WebSphere Decision Server (ILOG Rule Execution Server in v7.1) is the runtime rule engine which will be tightly coupled with WebSphere Message Broker to execute the required business rules for the message flow.
  - There is also a WebSphere Decision Server console which is often run from a separate system which manages 1 or more Decision Servers
- IT users have the ability to author, test and deploy business rules using an Eclipse based tool called WebSphere Rule Designer (ILOG Rule Studio in v7.1) which comes as part of WebSphere Decision Server (ILOG Rule Execution Server).
- There is an optional Rule Governance and Business User component called WebSphere Decision Center (ILOG Rule Team Server in v7.1) which allows and enterprise to provide a controlled environment to improve collaboration across Business and IT, and enable business users to view, or author and view business rules.

The recommended joint solution between WebSphere Message Broker and WebSphere Operational Decision Management is shown in figure 1.



Simply adding the 2 products together in this tightly coupled J2SE configuration of WODM had some management limitations and restrictions in the Rule Engine being informed when rules were being updated. These were eliminated by the new support pack by adding rule management capability to the J2SE engine using MBean events and management facility from J2EE engine. Both the Management/Authoring and runtime business rules engines work on a shared rule persistence layer (implemented as a database) to have a centralized single view of the latest business rules.

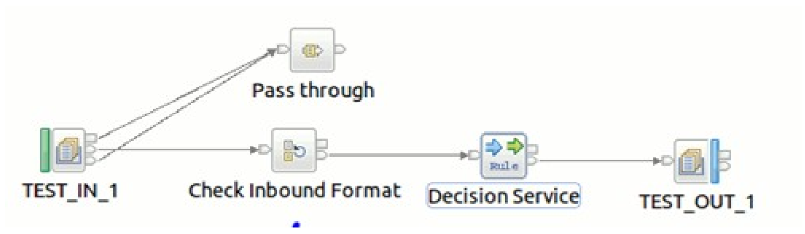
The new WebSphere Decision Service Node has two main processes which complement running the business rules

I. If a request enters a decision node a rule request is generated and the request is issued against the decision server (Call rule set process)
II. If a rule is changed by the Decision Server Management Console an event thrown by the MBean Server and this updates via a listener the cached rule in the J2SE Decision Server of the Message Broker (Update rule process).

A rule set is an ODM artifact which contains 1 or more individual rules and has pre-defined inputs and outputs which together provide the decisioning capability that the flow in WebSphere Message Broker requires at that point during the flow. Individual rules are not called directly.

2.How the Support Pack Capability works

The support pack provides an additional Message Broker flow action which can be used within a Message Broker flow ie (Decision Service)



The execution of a rule set starts when a message enters the WebSphere Decision Service Node. The message broker creates and caches a J2SE Decision Server in the execution group of the flow. The execution group is the flow container of message broker. If it's the first invocation of a specific rule set the J2SE Decision Server requests the rule set from the shared rule persistence layer one time and caches locally. Any subsequent rule set execution call only runs it within the JVM.
The rule set will run all the rules within it and return the result back to the Message Broker flow.

If the J2SE Decision Server is enabled for rule set listening the execution group starts a rule set listener. The listener connects the J2EE engine and registers for update events by the application servers MBean Interface[1].

If a rule is updated by a user via the Eclipse Rule Designer or WebSphere Decision Center and deployed as a RuleApp to the central WebSphere Decision Server persistence layer the execution group receives an event and forces the execution group to retrieve the updated rule set from the RuleApp.

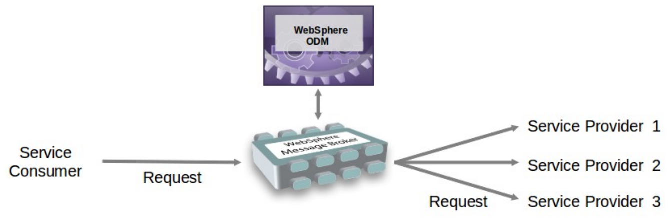
1 Jeff Cai from IBM Lab China implements first rule update was via WAS Admin Client. I take this as concept for my rule update solution.

3.Possible uses scenarios

This section describes applicable decision patterns which the WebSphere Decision Service Node enables. The Redbook Integrating WebSphere ILOG Jrules with IBM Software describes the following patterns in more detail.

Message routing

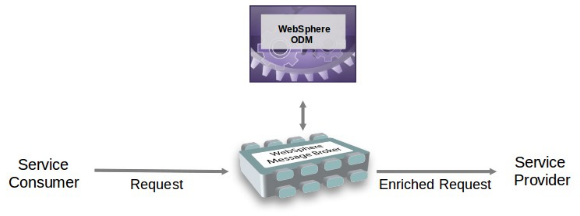
Using WebSphere Operational Decision Management to influence message flow routing decisions enables the business users to directly participate in the authoring and change management that is associated with routing decisions[2]. The Decision Node which invokes the WODM decision returns the result(s) which are the basis for the routing decision. The user can change individual rules within the decision on line by updating it using WebSphere Decision Center. No code update is needed. IT or business users can update on line and in real time.



Use Business Decisions to provide business level routing based on message content

Message enrichment

Another feature of an ESB is the ability to enrich messages with additional information to provide additional data, which cannot be supplied by the client application and which can only be obtained by the ESB interacting with other services, applications, data, or "business logic." This approach is known as message enrichment or message augmentation[3]. For example a customer sends an invoice to his health insurance. The company can check and extend the invoice with the decision system without having any rules in the bus system.

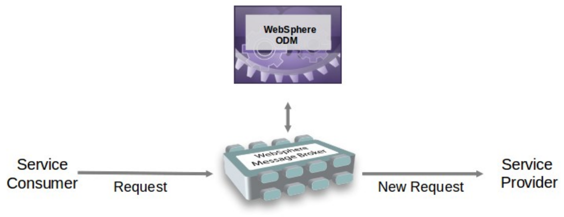


Use Business Decisions to provide additional information in messages based on business rules

2 Integrating WebSphere ODM with IBM Software
3 Integrating WebSphere ODM with IBM Software

Intelligent data mapping

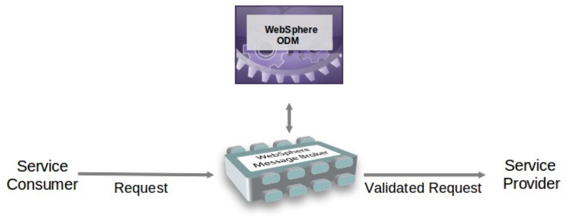
One of the roles of the connectivity layer is to provide features for transformation of the structure of messages. ESB message flows generally can transform messages at a structural and content level by using various mapping tools and features. It is more challenging where there is no direct mapping for the content between the source and target of the message transformation. In such cases, a level of business knowledge is sometimes needed to complete the required mapping[4]. The decision system is able to receive complex XML structures from the bus and send complex structures back. In general it is possible to relocate the mapping and endpoint look up to the decision system.



Use Business Decisions to provide mapping of messages data based on business rules

Business level message validation

Business rules can augment the standard ESB processing to provide a business context to the message validation. The decision system ensures that the content of data is correct and valid from a business perspective[5]. This means, you do not have to specify the validation rules in XML Schema and you might can go beyond of the basic XML schema validation capabilities.



Use Business Decisions to provide centralized business level validation of message content

4 Integrating WebSphere ODM with IBM Software
5 Integrating WebSphere ODM with IBM Software

4.Overall characteristics of the WebSphere Decision Service node

The WebSphere Decision Service Node is implemented as a Java User Defined Node. If you follow the installation instructions of this Support Pac, add the node from the WebSphere Message Broker Toolkit palette as a Decision Service element to a flow. This node can be included in message flows and sub flows.

You have to configure some properties to use the node. The section User interface for developers describes the properties and the section Input and Output Message Format shows the construction of a valid input tree.

The Java Virtual Machine (JVM) of the execution group caches the JMX connection to the application server and decision server. The key for a specific decision server in the cache is the XU Config property of a node.

Valid message domain's for the input tree are XML, XMLNS and XMLNSC. The domain for the output tree is always XMLNSC.

5.Other Characteristics

The WebSphere Decision Service Node does not delete any elements of the input tree. The output tree matches the input tree and is extended by queried result. The results are placed relatively to the call element under *DecisionServiceCall.*

6.Limitations

The current version of the node supports ILOG BRMS v7.1 or WebSphere Operational Decision Management version 7.5 and 8.0.

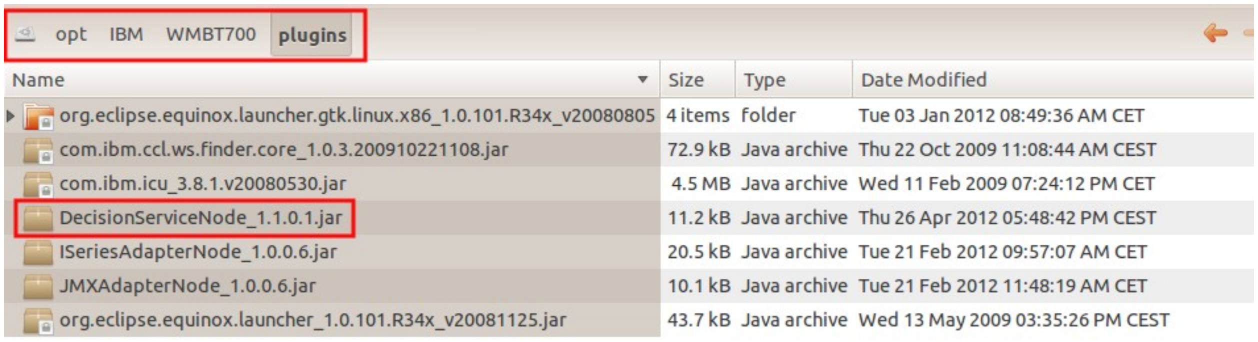
7.Prerequisites

This SupportPac requires WebSphere Message Broker Toolkit version 7.0 or later.

8.Installation

Installing the Support Pac in the Toolkit

The WebSphere Decision Service Node is available as a ZIP file, which can be downloaded from the IBM web site for “WebSphere MQ - SupportPacs by Product”. The SupportPac Number is IAM9.



Name	Size	Type	Date Modified
org.eclipse.equinox.launcher.gtk.linux.x86_1.0.101.R34x_v20080805	4 items	folder	Tue 03 Jan 2012 08:49:36 AM CET
com.ibm.ccl.ws.finder.core_1.0.3.200910221108.jar	72.9 kB	Java archive	Thu 22 Oct 2009 11:08:44 AM CEST
com.ibm.icu_3.8.1.v20080530.jar	4.5 MB	Java archive	Wed 11 Feb 2009 07:24:12 PM CET
DecisionServiceNode_1.1.0.1.jar	11.2 kB	Java archive	Thu 26 Apr 2012 05:48:42 PM CEST
ISeriesAdapterNode_1.0.0.6.jar	20.5 kB	Java archive	Tue 21 Feb 2012 09:57:07 AM CET
JMXAdapterNode_1.0.0.6.jar	10.1 kB	Java archive	Tue 21 Feb 2012 11:48:19 AM CET
org.eclipse.equinox.launcher_1.0.101.R34x_v20081125.jar	43.7 kB	Java archive	Wed 13 May 2009 03:35:26 PM CEST

- Unzip the file to a temporary directory
- Copy the DecisionServiceNode_*.jar file to the plugins directory of the WebSphere Message Broker Toolkit installation. By default the directory path is:
  <Toolkit_RootInstallDirectory>/plugins on the Windows or Linux workstation. This JAR file contains the WebSphere Decision Service User Defined Node.
- Start the Toolkit.
- The WebSphere Decision Service Node is now available on the Toolkit Node Palette.

Uninstalling the Support Pac in the Toolkit

- Stop the WebSphere Message Broker Toolkit.
- Locate the DecisionServiceNode_*.jar file in the WebSphere Message Broker Toolkit plugins directory.
- Delete the DecisionServiceNode_*.jar file.
- Restart the WebSphere Message Broker Toolkit.

Installing the node at the WebSphere Message Broker runtime

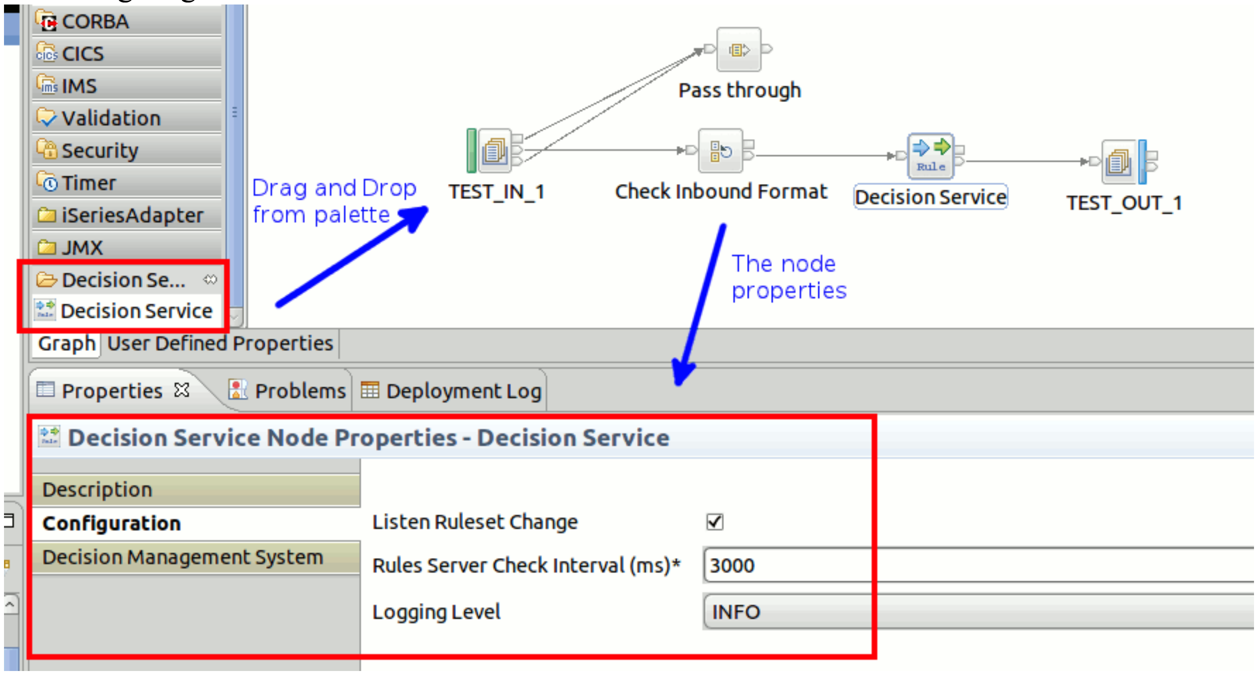
- Unzip the ZIP file to a temporary directory
- Copy the DecisionServiceImpl.jar file to the lil path of your WebSphere Message Broker runtime.
  - The default lil path for Linux environments is : /var/mqsi/lil
  - Hint: A user defined lil path can be set by using the mqsichangebroker command
- Copy the required jars to the shared-classes directory of your WebSphere Message Broker runtime,
  - WebSphere Operational Decision Management runtime and DB2 client.
  - Chapter *Developers Guide* describes this in detail.

- Default path for Linux environments is: /var/mqsi/shared-classes

9.User interface for developers

After completing the installation on the developer's workstation, the developer is able to drag and drop the WebSphere Decision Service Node from the Node Palette and use it like any the other WebSphere Message Broker Toolkit nodes.

Following diagram illustrates the user interface.

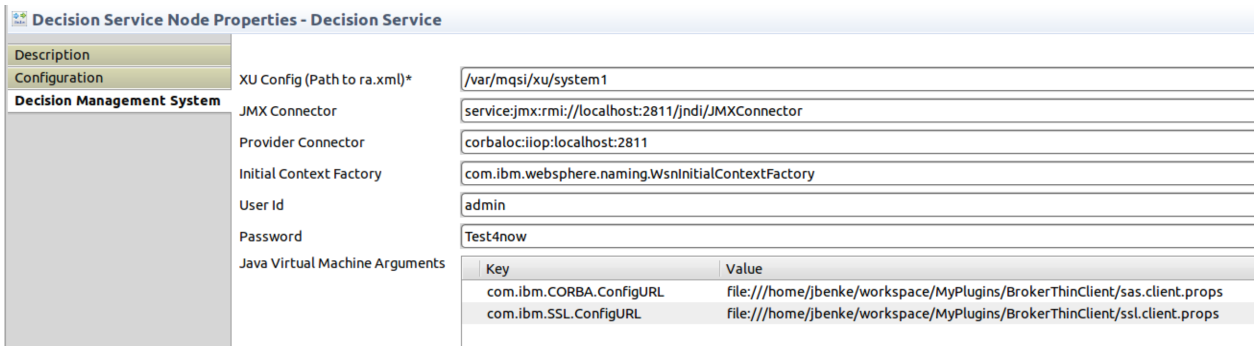


Listen Ruleset Change ☑
Rules Server Check Interval (ms)* 3000
Logging Level INFO

The section Configuration has three node properties:

- Listen Ruleset Change – Defines if the node receives rule change event or not.
  - The type is boolean.
  - This property is required.
  - The default is false.
  - True means, the broker listens at a J2EE Decision Management System.
- Rules Server Check Interval (ms) – Defines a interval in milliseconds which the broker uses to send a keep alive to the JMX interface of the application server. If the check fails it tries to reconnect the JMX interface.
  - The type is integer.
  - This property is required.
  - The default is 3000.
- Logging Level – Log Level for the node.
  - The type is enum.
  - The default is INFO.
  - The logs get written in the system out file of the execution group.

- The level INFO provides information for the rule update events and the raised exceptions.
- Use all level below INFO for debugging purposes.



Decision Service Node Properties - Decision Service

Description
Configuration
Decision Management System

XU Config (Path to ra.xml)*	/var/mqsi/xu/system1
JMX Connector	service:jmx:rmi://localhost:2811/jndi/JMXConnector
Provider Connector	corbaloc:iiop:localhost:2811
Initial Context Factory	com.ibm.websphere.naming.WsnInitialContextFactory
User Id	admin
Password	Test4now

Java Virtual Machine Arguments

Key	Value
com.ibm.CORBA.ConfigURL	file:///home/jbenke/workspace/MyPlugins/BrokerThinClient/sas.client.props
com.ibm.SSL.ConfigURL	file:///home/jbenke/workspace/MyPlugins/BrokerThinClient/ssl.client.props

Seven node properties can be specified in the section Decision Management System:

- XU Config (Path to ra.xml) – Defines the directory of the ra.xml file. The file name must not be specified. The file contains all needed information for the Rule Execution Server to connect the persistent rule source.
  - The type is string.
  - This property is required.
  - The default is “<default ra.xml>”. This case uses the internal ra.xml of jrules-res-execution.jar.
- JMX Connector – Defines the url of the JMX interface of the application server
  - The type is string.
  - Example string: service:jmx:rmi://<hostname>:<bootstrap port> /jndi/JMXConnector
- Provider Connector – Defines the URL of the IIOP interface of the application server
  - The type is string.
  - Example string: corbaloc:iiop:<hostname>:<bootstrap port>
- Initial Context Factory – Defines the initial JNDI context factory of the used decision server application server runtime.
  - The type is string.
  - Example for WebSphere Application Server: com.ibm.websphere.naming.WsnInitialContextFactory
- User Id – The application server user who access the MBean server.
  - The type is string.
- Password – The application server user password of the user who access the MBean server.
  - The type is string.
- Java Virtual Machine Arguments - Properties need by the client for a specific implementation of a application server to support IIOP connection and SSL security.
  - The type is table property with a string as key and value.
  - For WebSphere Application Server you need

com.ibm.SSL.ConfigURL	file:///<thin-client-dir>/ssl.client.props
com.ibm.CORBA.ConfigURL	file:///<thin-client-dir>/sas.client.props

10.Developer's Guide

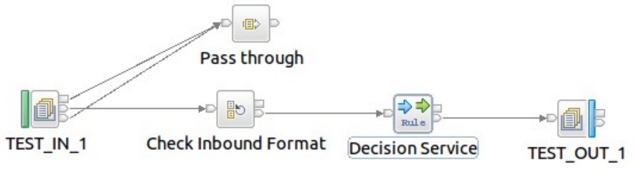
The following paragraphs illustrate how to use the WebSphere Decision Service together with WebSphere Application Server (WAS) version 7.0.0.19.

Example with Application Server Security enabled

The application server security is enabled in the example. If the application server security is not needed you can skip some steps and settings. These steps are marked in the following procedure.

Add the WebSphere Decision Service node to a Message Broker flow. The sample flow contains a MQ input node followed by a Reset Content Descriptor node and a Decision Service node. A MQ output node writes the result to a queue. There are some node properties to specify in the Decision Service Node.

The following diagram shows the message flow. The scenario uses the XML binding sample from the ILOG BRMS 7.1.1 info center as test rule. This is also applicable to WODM V7.5 You can deploy the rule set with Rule Designer to use the provided flow and test message.



Defining the node properties

1. Listen Ruleset Change – This property is checked to receive rule updates.
2. Rules Server Check Interval (ms) – The value is set to 4000. This means the data flow engine checks every 4000 Milliseconds if the connected JMX application server interface is still alive by using the JMX function getMBeanCount(). If the result is negative the data flow engine tries a reconnect to the server.
3. Logging Level – The enumeration is set to INFO. This setting allows to observe rule changes in the system out of the data flow engine.
4. XU Config (Path to ra.xml) – The path is set to /var/mqsi/xu/system1. The ra.xml resides here. The data flow engine uses this path for initializing the Decision Server (DS). If your broker environment uses more than one DS, specify a second path with an other ra.xml e.g. /var/mqsi/xu/system2. The property is determinative for a node. But you can mix several WebSphere Decision Service nodes with different XU Config property values in a flow. The node caches the constructed DC's and the JMX Connections for every used value. This means every directory describes an singular Decision Server. If two nodes use the same value the second one uses the cache.

5. JMX Connector – The URL is set to service:jmx:rmi://localhost:2811/jndi/JMXConnector. The ip address and the bootstrap port combination are normally the variable part of this property. The administrative console of your WAS installation shows the value you need for bootstrap. You need the bootstrap port of application server with deployed Decision Server.
6. Provider Connector – The URL is set to corbaloc:iiop:localhost:2811. The ip address and the bootstrap port combination are the variable part of this property. JMX Connector shows how to check the bootstrap value.
7. Initial Context Factory – The used application server type defines this value. Use for WebSphere Application Server: com.ibm.websphere.naming.WsnInitialContextFactory
8. User Id – The user is set to admin. The user has to be able to connect the JMX interface of the application server. The properties can have any value in a security turned off scenario.
9. Password – The password is set to Test4now. The properties can have any value in a security turned off scenario.
10. Java Virtual Machine Arguments – The table has two key value pairs with the keys com.ibm.SSL.ConfigURL and com.ibm.CORBA.ConfigURL. The node uses the values to build the IIOP connecting to Application server. The values are discussed in chapter Enable Application Server Security and IIOP Connection. The table is empty in a security turned off scenario.

Input message

```
<?xml version="1.0" encoding="UTF-8" ?>
<DecisionService  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="DecisionService.xsd">
        <Properties>
                <Property key="rulesetPath" value="/xmlbindingapp/xmlbindingrules" />
        </Properties>
        <Call>
                <Parameters>
                        <Parameter name="eventBatch" type="java.lang.String">
                        <![CDATA[<EventBatch xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                                        <CashEvents>
                                                <ID>ID_R1</ID>
                                                <parentID/>
                                                <type>CREDIT</type>
                                                <date>2004-01-16</date>
                                                <processed>true</processed>
                                                <accountID>124</accountID>
                                                <amount>100</amount>
                                                <currency>USD</currency>
                                                <originAccountID>123</originAccountID>
                                        </CashEvents>
                                        <Accounts>
                                                <ID>124</ID>
                                                <currency>USD</currency>
                                                <balance>300</balance>
                                                <denied>false</denied>
                                                </Accounts>
                                        </EventBatch>]]>
                        </Parameter>
                </Parameters>
        </Call>
</DecisionService>
```

The input message of a WebSphere Decision Service node looks like in the listing above. The chapter Input and Output Message Format explains the grammar of the structure in detail. The result of rule call is a new element in the input structure. This tag has the name Response and is a child of DecisionService. The support pac provides a DecisionService.xsd which could be used for tree validation.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<DecisionService  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="DecisionService.xsd">
        <Properties>
                <Property key="rulesetPath" value="/xmlbindingapp/xmlbindingrules" />
        </Properties>
        <Call>
                <Parameters>
                        <Parameter name="eventBatch" type="java.lang.String">
                        <![CDATA[<EventBatch xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                                        <CashEvents>
                                                <ID>ID_R1</ID>
                                                <parentID/>
                                                <type>CREDIT</type>
                                                <date>2004-01-16</date>
                                                <processed>true</processed>
                                                <accountID>124</accountID>
                                                <amount>100</amount>
                                                <currency>USD</currency>
                                                <originAccountID>123</originAccountID>
                                        </CashEvents>
                                        <Accounts>
                                                <ID>124</ID>
                                                <currency>USD</currency>
                                                <balance>300</balance>
                                                <denied>false</denied>
                                                </Accounts>
                                        </EventBatch>]]>
                        </Parameter>
                </Parameters>
        </Call>
        <Response>
                <Parameters>
                        <Parameter name="eventBatch">
                        <![CDATA[<EventBatch xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                                        <FinancialEvents>
                                                <ID>ID2</ID>
                                                <parentID>ID_R1</parentID>
                                                <type>ACKNOWLEDGEMENT</type>
                                                <date>2004-01-16</date>
                                                <processedDate>2012-02-16</processedDate>
                                                <processed>true</processed>
                                                <accountID>124</accountID>
                                        </FinancialEvents>
                                        <CashEvents>
                                                <ID>ID_R1</ID>
                                                <parentID></parentID>
                                                <type>CREDIT</type>
                                                <date>2004-01-16</date>
                                                <processedDate>2012-02-16</processedDate>
                                                <processed>true</processed>
                                                <accountID>124</accountID>
                                                <amount>100.0</amount>
                                                <currency>USD</currency>
                                                <originAccountID>123</originAccountID>
                                        </CashEvents>
                                        <Accounts>
                                                <ID>124</ID>
                                                <currency>USD</currency>
                                                <balance>400.0</balance>
                                                <denied>false</denied>
                                        </Accounts>
                                        </EventBatch>]]>
                        </Parameter>
                </Parameters>
        </Response>
</DesicionService>
```

Needed jars

1. Copy decision server jars from WebSphere Operation Decision Server to Message Broker.

ODM dir: <ODM install path>/executionserver/lib
WMB dir: /var/mqsi/shared-classes

```
asm-3.1.jar
asm-analysis-3.1.jar
asm-tree-3.1.jar
asm-commons-3.1.jar
asm-util-3.1.jar
j2ee_connector-1_5-fr.jar
jrules-engine.jar
jrules-res-session-java.jar
jrules-res-execution.jar
sam.jar
```

2. Copy DB2 client jars to message broker.

DB2 dir: /opt/ibm/db2/V9.1/java
WMB dir: /var/mqsi/shared-classes

```
db2jcc_license_cu.jar
db2jcc.jar
```

3. Copy WebSphere Application Server jar to message broker.

WAS dir: <was_install_root>/runtimes
WMB dir: /var/mqsi/shared-classes

```
com.ibm.ws.admin.client_7.0.0.jar
com.ibm.ws.security.crypto.jar
```

4. Copy XOM model jar to /var/mqsi/shared-classes. Export the jar with eclipse from xmlbinding-xom project. Do not use the compress option to export the code. The example contains a jar export of xmlbinding-xom project.

Enable Application Server Security and IIOP Connection

The following steps have to be completed for accessing WebSphere Application Server 7.0.0.19 with enabled Application Server Security. For using this example with other application servers the corresponding documentation needs to be read. Proceed the following steps and restart the broker:

- Make the Administration Thin Client JAR file available to the broker by copying com.ibm.ws.admin.client_7.0.0.jar from a WebSphere Application Server environment to the shared-classes folder of the broker.
- Make the security JAR file available to the broker by copying com.ibm.ws.security.crypto.jar from a WebSphere Application Server environment to the shared-classes folder of the broker.
- To use JSR160 RMI connectors, copy the sas.client.props file from the AppServer/profiles/*profileName*/properties directory and put it in a directory of your choice.
  ```
  com.ibm.CORBA.validateBasicAuth=false
  com.ibm.CORBA.securityServerHost=localhost
  com.ibm.CORBA.securityServerPort=<bootstrap port>
  ```
- Copy the ssl.client.props file from the AppServer/profiles/*profileName*/properties directory and put it in a directory of your choice. This file contains the user.root property. You must modify the value to your thin client directory. This file refers to a key store, copy it from your application server or create a new one and import the application server certificate.
- Copy the orb.properties file from the AppServer/properties directory and put it in a directory of your choice. On Unix systems the file must be placed in the home directory of the broker user or create a symlink from the file to the home directory.

For more information refer to the article “Using the Administration Thin Client" in the WebSphere Application Server information center.

Configure the ra.xml

The ra.xml configures the J2SE Decision Server. For a detailed configuration explanation refer to WODM information center of the used release.

The below property specify the persistence source.

```
<config-property>
        <config-property-name>persistenceProperties</config-property-name>
        <config-property-type>java.lang.String</config-property-type>
        <!--<config-property-value>DIRECTORY=/var/mqsi/res_data</config-property-value>-->
        <config-property-value>
DRIVER_CLASS_NAME=com.ibm.db2.jcc.DB2Driver,URL=jdbc:db2://localhost:50000/RESDB,USER=db2inst1,PASSWOR
D=test
        </config-property-value>
</config-property>
```

Set the persistence typ of the J2SE Decision Server Control to jdbc.

```
<config-property>
        <config-property-name>persistenceType</config-property-name>
        <config-property-type>java.lang.String</config-property-type>
        <config-property-value>jdbc</config-property-value>
</config-property>
```

Do not use any plugin running the J2SE Decsion Server in Message Broker.

```
<config-property>
        <config-property-name>plugins</config-property-name>
        <config-property-type>java.lang.String</config-property-type>
        <config-property-value></config-property-value>
</config-property>
```

Input and Output Message Format

The root element of the input message is DecisionService. It must contain an Properties and Call element.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<DecisionService>
        <Properties>
                <Property key="rulesetPath" value="$rulestring" />
        </Properties>
        <Call>
                <Parameters>
                        <Parameter name=”$ruleParameter1” type="java.lang.String">
                                <![CDATA[an xml document]]>
                        </Parameter>
                        <Parameter name=”$ruleParameter2” type="java.lang.String">
                                <![CDATA[aString]]>
                        </Parameter>
                        <Parameter name=”$ruleParameter3”  type="java.lang.Integer">
                                <![CDATA[aInteger]]>
                        </Parameter>
                </Parameters>
        </Call>
</DecisionService>
```

The Properties contains one Property element which has two attributes. These are key and value. The key has the value rulesetPath. The value attribute is the canonical ruleset string. If ruleAppVersion and rulesetVersion are absent, the most recent version of the rule is used.

*/ruleAppName/**ruleAppVersion**/rulesetName/**rulesetVersion***

The Call element contains a Parameters element. This element encloses an unbounded number of Parameter elements. The number relates to the rules parameters of the called rule. All defined ruleset parameters with direction IN and IN_OUT must appear as element. The name attribute of element is the name of the ruleset parameter. Every element has an additional attribute type to specify the ruleset parameter. This is normally a Java primitive. Valid java types for the node has to provide a constructor with one argument. This argument must be java.lang.String. If the parameter type is a complex XML structure the type is java.lang.String. A CDATA section must mask the value of the element.

```
<DecisionService>
        <Properties>
                <Property key="rulesetPath" value="$rulestring" />
        </Properties>
        <Call>
                <Parameters>
                        <Parameter name=”$ruleParameter1” type="java.lang.String">
                                <![CDATA[an xml document]]>
                        </Parameter>
                        <Parameter name=”$ruleParameter2” type="java.lang.String">
                                <![CDATA[aString]]>
                        </Parameter>
                        <Parameter name=”$ruleParameter3”  type="java.lang.Integer">
                                <![CDATA[aInteger]]>
                        </Parameter>
                </Parameters>
        </Call>
        <Result>
                <Parameters>
                        <Parameter name=”$ruleParameter3”>
                                <![CDATA[an xml document]]>
                        </Parameter>
                        <Parameter name=”$ruleParameter4”>
                                <![CDATA[aString]]>
                        </Parameter>
                        <Parameter name=”$ruleParameter5”>
                                <![CDATA[aInteger]]>
                        </Parameter>
                </Parameters>
        </Result>
</DecisionService>
```

The output message enriches the input message with a result element. It contains all ruleset parameter with direction IN_OUT and OUT. The Parameter element has a attribute name. The value is the name of the ruleset parameter. A CDATA section masks the output value and contains the toString() method output of the ruleset parameter type. XML structures are handled as java.lang.String.