OPENRULES®

Open Source Business Decision Management System

Release 6.2

Web Services

OpenRules, Inc.
www.openrules.com
August-2013

TABLE OF CONTENTS

INTRODUCTION

OpenRules® is as an open source Business Decision Management System (BDMS) with proven records of delivering and maintaining reliable decision support software. The detailed description of OpenRules® can be found in the User Manual.

If your system is based on a Service-oriented architecture, you will probably prefer to incorporate your business rules in a special type of decision services. Such services should be loosely coupled and easily pluggable into your decision making processes. With OpenRules you define and maintain your business rules using a combination of Excel, Java, and Eclipse. You may test your rules as a stand-alone application or execute them locally using a simple Java API. However, when you are ready to integrate your business rules into an enterprise level application, OpenRules provides a "push-button" mechanism for deployment of business rules as Web Services.

This manual is devoted to creation of OpenRules®-based web services. It describes a simple example that demonstrates how to create a decision and deploy it as a web service and how to create a client that can utilize this service. The standard OpenRules® installation comes with several samples of rules-based web services.

ENVIRONMENT

If you know how to deploy a Java program as a web service, you should have no problems to deploy an OpenRules®-based project too. Usually OpenRules projects can be invoke using Java API defined by the standard classes OpenRulesEngine or Decision – read about OpenRules API more in the User Manual. Such projects usually utilize a rule repository that can be accessed using standard URL protocols such as “file:”, “classpath:”, “ftp:”, “http:”, etc. When a Java program

that uses such repositories is deployed as a web service, you just have to make sure that this service still may access the same rule repository.

In the example below, we will use the following open source tools:

- Apache Tomcat as a deployment server
- Apache Axis 1.4 as a reliable and stable base on which to implement Java Web services.

The deployment properties are usually described in the file "build.properties" and should correspond to your Tomcat and Axis installations. Here is an example of the build properties:

```
# deployment properties
appserver.home=C:/apache-tomcat-6.0
deploy.path=${appserver.home}/webapps
tomcat.manager.url=http://localhost:8080/manager
tomcat.manager.username=tomcat
tomcat.manager.password=tomcat

axis.lib=C:/axis-1_4/lib
```

Building Web Service: Project "DecisionHelloWSCustomer"

The standard OpenRules® installation comes with the basic (not a web service) project "DecisionHelloJava" that shows how to invoke an Excel-based decision from a Java program. This project does the following:

- Creates a customer as an instance of the Java class Customer
- Creates a response as an instance of the Java class Response
- Creates and executes a decision that is based on the customer's attributes such as gender, marital status, and age is supposed to generate a response that contains a greeting like "Good Afternoon, Mrs. Robinson!"

The rules used by this decision are located in Excel files starting with "`file:rules/main/Decision.xls`".

Our objective is to deploy the same decision as a web service on Tomcat using Apache Axis. So, we will create a new project "DecisionHelloWSCustomer" using exactly the same Excel-based rules and Java classes Customer and Response.

After copying the project folder "DecisionHelloJava" to the new folder "DecisionHelloWSCustomer", we will create a new sub-folder "war" that will be a placeholder for all files that constitute a future war-file "DecisionHelloWSCustomer.war" – the main deployment objective. Its name also defined in the file "build.properties" as the property "war.name".

We will move the folder "rules" with all rules to "war/rules" without any changes. In the source folder "src" we will create a Java package "com.openrules.examples" and will move the same Java classes Customer and Response to this package.

Now we need to add to this package a very simple new class "DecisionHelloWSCustomer" that will serve as a launcher for our future web service. Below is its code:

```
package com.openrules.examples;

import com.openrules.ruleengine.Decision;

public class DecisionHelloWSCustomer {

   static Decision decision;

   static String mainExcelFile =
"file:./webapps/DecisionHelloWSCustomer/rules/main/DecisionHelloWSCustomer.xls";

   static String decisionName = "DetermineCustomerGreeting";

   public DecisionHelloWSCustomer() {
         init();
   }

   public Response hello(Customer customer, Response response) {
         decision.put("customer", customer);
         decision.put("response", response);
         decision.execute();
         return response;
   }

   static boolean initialized = false;

   static synchronized protected void init() {
         if (initialized)
               return;
```

```
            decision = new Decision(decisionName, mainExcelFile);

            initialized = true;
    }
}
```

During the first invocation of the web service, an instance of this class will be created with a static object Decision that is based on the rule repository defined by the main Excel file

```
"file:./webapps/DecisionHelloWSCustomer/rules/main/DecisionHelloWSCustomer.xls"
```

This path points to a relative location inside Tomcat server into which our web service will be deployed. A user may choose any other location using different URL protocols.

The method

```
Response hello(Customer customer, Response response)
```

specifies a signature for our web service that will take a customer and a response as parameters and after the processing will return a modified response.

The folder "war/WEB-INF" contains the folder "classes" in which all binaries will be moved. To complete our configuration we need to add a standard Axis 1.4 configuration file "**service-config.wsdd**" to the folder "war/WEB-INF". A template for this file comes with Axis 1.4 and you should only add several lines specific for you web service. In our case we added the following lines:

```
<!-- ********************** your service definition ********************
*************************** read axis docs before changing it ******* -->

 <service name="DecisionHelloWSCustomer" provider="java:RPC">
  <parameter name="className"
value="com.openrules.examples.DecisionHelloWSCustomer"/>
  <parameter name="allowedMethods" value="hello"/>
  <beanMapping qname="myNS:Customer" xmlns:myNS="urn:com.openrules.examples"
               languageSpecificType="java:com.openrules.examples.Customer"/>
  <beanMapping qname="myNS:Response" xmlns:myNS="urn:com.openrules.examples"
               languageSpecificType="java:com.openrules.examples.Response"/>
 </service>

<!-- ****************** end of your service definition ************ -->
```

The tag "<service" defines the name `"DecisionHelloWSCustomer"` of our web service. The parameter "className" points to our class `DecisionHelloWSCustomer` with one allowed method "`hello`". We also need to define "beanMapping" tags to associate the concepts Customer and Response with our Java classes Customer and Response located in the package "`com.openrules.examples`".

We may run "*build.bat*" to generate a war-file "DecisionHelloWSCustomer.war" and to check that there are no syntax errors. Then we may run "*deploy.bat*" to copy this war to the proper Tomcat's "webapps" directory.

Then we may start Tomcat that will expand this war file to a ready to go subfolder "webapps/ DecisionHelloWSCustomer". To validate if our web service was correctly generated we may open "*run.html*" with any Internet browser. It should show a generated WSDL file located at this address:

```
<a href="http://localhost:8080/DecisionHelloWSCustomer/services/DecisionHelloWSCustomer?wsdl">
```

Testing Web Service: Project "DecisionHelloWSCustomerClient"

We will create a special test project "DecisionHelloWSCustomerClient" to test the web service described above. All supporting Java classes for this test project can be automatically generated utilizing Axis 1.4. The file "build.properties" describes the configuration parameters:

```
axis.lib=C:/axis-1_4/lib
local.wsdl=http://localhost:8080/DecisionHelloWSCustomer/services/DecisionHelloWSCustomer
generated.dir=gen
client.classname=examples.openrules.com.DecisionHelloWSCustomerClient
```

Here is a brief description of these parameters:

- `axis.lib` points to the folder that contains standard Axis jar-files
- `local.wsdl` points to URL of our web service

- `generated.dir` is the name of source folder in which all generated Java classes will be saved
- `client.classname` points to the main Java class that will be used for testing. Note that it the package name "*examples.openrules.com*" is a reversed name of the initial package "*com.openrules.examples*".

To generate all supporting Java classes, make sure that your Tomcat server with deployed web service (described by `local.wsdl`) is up and running. Double-click on "*generate.axis.clien.bat*". It will read our WSDL file from the running Tomcat and will generate a Java package "*examples.openrules.com*" with the following classes:

- DecisionHelloWSCustomer
- DecisionHelloWSCustomerService
- DecisionHelloWSCustomerServiceLocator
- DecisionHelloWSCustomerSoapBindingStub
- Customer
- Response.

Now we need to manually create the main test class DecisionHelloWSCustomerClient that is based on these generated classes. To make sure that it is not being overwritten during re-generation, we will put this class in a different source folder "src" but in the package with the same name "*examples.openrules.com*". Here is its code:

```
public class DecisionHelloWSCustomerClient {

   public static void main(String[] args) throws Exception {
         new DecisionHelloWSCustomerClient().testHello();
   }

   public void testHello() throws Exception {
         DecisionHelloWSCustomerSoapBindingStub binding;
         try {
               binding = (DecisionHelloWSCustomerSoapBindingStub)
                           new DecisionHelloWSCustomerServiceLocator()
                           .getDecisionHelloWSCustomer();
         } catch (ServiceException jre) {
               if (jre.getLinkedCause() != null)
                     jre.getLinkedCause().printStackTrace();
               throw new
```

```
                    Exception("JAX-RPC ServiceException caught: " + jre);
            }

            // Time out after a minute
            binding.setTimeout(60000);

            // Test operation
            Customer customer = new Customer();
            customer.setName("Robinson");
            customer.setAge(5);
            customer.setGender("Male");
            customer.setMaritalStatus("Single");
            customer.setHour(16);
            Response response = new Response();
            response = binding.hello(customer, response);
            System.out.println("greeting: " + response.getGreeting()
                               + " salutation: " + response.getSalutation());
            System.out.println(response.getResult());
    }

}
```

In the method *testHello()* we create a "binding" as an instance of the generated class DecisionHelloWSCustomerSoapBindingStub. Then we create test instances of the classes "Customer" and "Response". Then we execute our web service for this customer and response using the code

```
response = binding.hello(customer, response);
```

After executing our rules-based decision it returns an object response that already contains a greeting and a salutation for this customer.

Note. If you receive an error that state that the file "`file:./webapps/DecisionHelloWSCustomer/rules/main/DecisionHelloWSCustomer.xls`" is not found, you may want to go back to your web service and add an additional "." to the definition of your main Excel file (e.g. "`file:../webapps/..`"). It depends on the way you start your Tomcat (as a service or not). Of course, then you should regenerate your web service, restart Tomcat, and try again.

More Rule Service Examples

The standard OpenRules® installation comes with several more samples of rules-based web services. The workspace "*openrules.web*" include in the complete version, contains two additional projects "**HelloWS**" and "**HelloWSCustomer**". They are based not on Decision but rather on basic rule projects that are invoke using the OpenRulesEngine. Additionally these projects demonstrate how to automatically generate the main web service files by defining additional generation properties in the file "*build.properties*". These projects come with readme-files that describe how they are organized and can be built.

Technical Support

Direct all your technical questions to support@openrules.com or to this Discussion Group.