Taylor Osmun (w91pq[AT]unb[DOT]ca)

Derek Smith (smith[DOT]de[AT]unb[DOT]ca)

Harold Boley (harold[DOT]boley[AT]nrc[DOT]gc[DOT]ca)

Adrian Paschke (paschke[AT]inf[DOT]fu-berlin[DOT]de)

Zhili Zhao (zhaozhil03[AT]gmail[DOT]com)

RuleML Report, 20 December 2011
http://ruleml.org/RuleResponder/RuleResponderGuide

Previous version

Acknowledgements

We thank our many users for giving valuable feedback on all aspects of Rule Responder. In particular, we thank Chaudhry Usman Ali for helping us to improve our presentation of how to set up Rule Responder.

[RULE RESPONDER GUIDE]

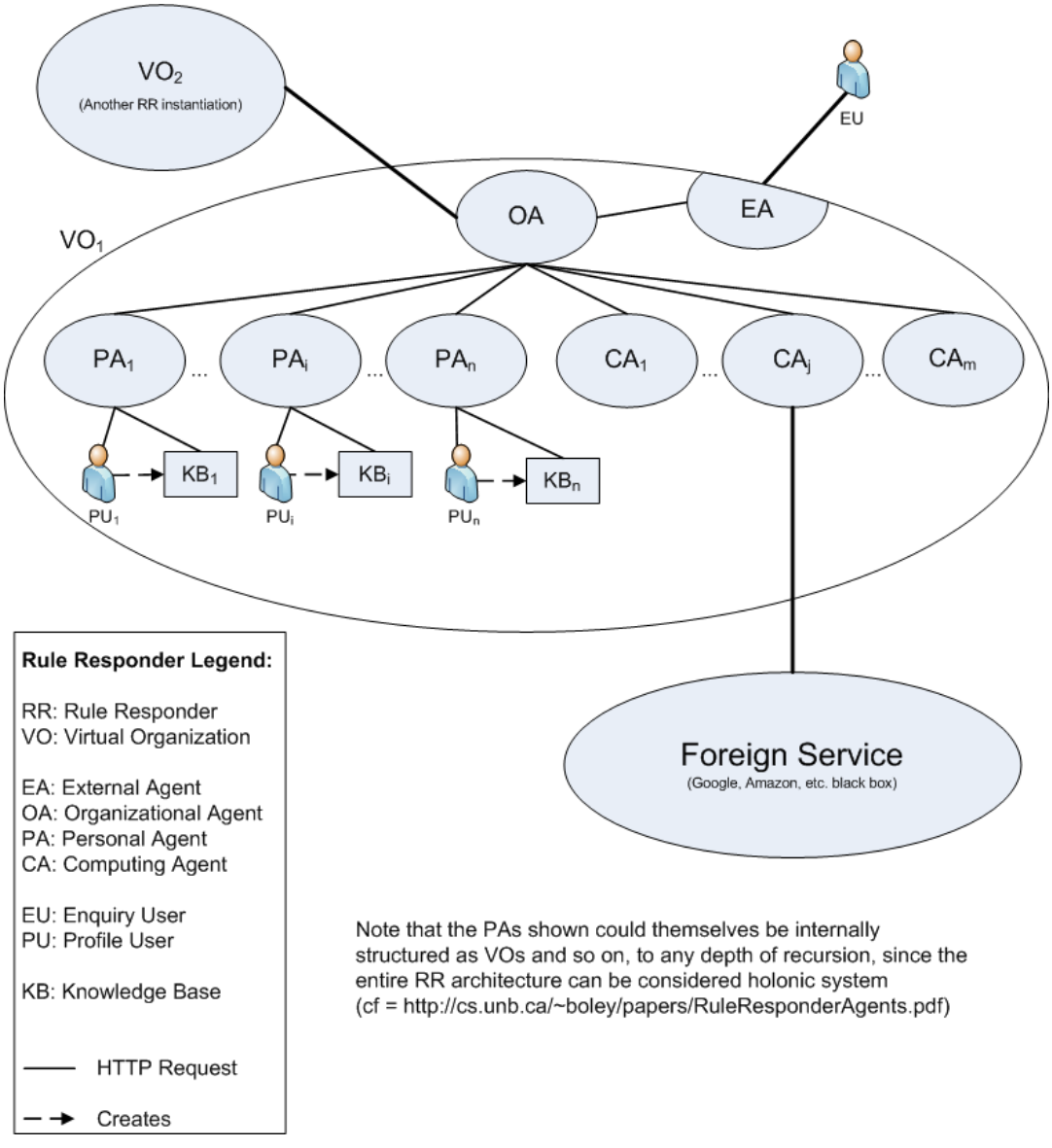
Rule Responder is an open source framework for creating virtual organizations as multi-agent systems. This guide shows how to set up Rule Responder on top of the Mule Enterprise Service Bus (ESB) as well as the four kinds of agents used by Rule Responder, i.e. Organizational Agents, Personal Agents, External Agents, and Computing Agents. It describes the instantiations SymposiumPlanner (2008-2011), WellnessRules(2), and PatientSupporter. The supported reasoning engines (with their languages) are Prova, OO jDREW with POSL, and Euler with N3. The guide discusses RuleML Query Generation, the PatientSupporter Profile Generator, and the Rule Responder Benchmarking Tool. It also explains troubleshooting of Rule Responder instantiations.

Table of Contents

(click to navigate)

1 Introduction

Rule Responder is an open source framework for creating virtual organizations as multi-agent systems that support collaborative teams on the Semantic Web. It comes with a number of official instantiations implementing virtual organizations such as SymposiumPlanner for supporting the chairs of the RuleML Symposia. Rule Responder provides the infrastructure for rule-based collaboration between the distributed members of such a virtual organization. Human members are assisted by semi-autonomous rule-based agents, which use Semantic Web rules that describe aspects of their owners' derivation and reaction logic. The diagram below shows the architecture of a Rule Responder system.



This guide to Rule Responder shows how to set up Rule Responder on top of the Mule Enterprise Service Bus (ESB) as well as the four kinds of agents used by Rule Responder, i.e. Organizational Agents, Personal Agents,

External Agents, and Computing Agents. The guide then describes the official instantiations SymposiumPlanner (2008-2011), WellnessRules(2), and PatientSupporter. Next are the supported reasoning engines (with their languages), namely Prova, OO jDREW with POSL, Euler with N3, and DR-Device with DR-POSL. Subsequently, the guide discusses RuleML Query Generation, the PatientSupporter Profile Generator, and the Rule Responder Benchmarking Tool. The final guide section explains troubleshooting of Rule Responder instantiations.

2 Setting up Rule Responder

Rule Responder employed Mule 1.4.1 and Prova 2.0 in its first release. During the last few years, both Mule and Prova communities have made significant improvements on their products. Since Mule 1.4, Mule has been made a lot of configuration and code changes (More information can be found at: http://www.mulesoft.org/documentation/display/MULE2INTRO/Migrating+Mule). Prova 3.1.3 also resolved many issues compared to its previous version (Please find its specification at: http://www.prova.ws/). In 2011, we upgraded Rule Responder and made it support Mule 3.1.0 and Prova 3.1.3. Since most of the libraries of Mule and its configuration have been replaced or changed, we publish our latest Rule Responder (Rule Responder 3) at a different branch of repository. However, we still reserve the previous release for our old users. Following the same specification in this section, users can choose either release to install.

2.1 Windows Environment

2.1.1 Prerequisites

Note for Windows Vista Users: Be advised that several communication errors between the OA and PA may occur due to Vista security issues. These will hopefully be remedied in future versions of Rule Responder. It is advisable to not use Windows Vista at this time. See more information in Section 14.5 Issues and Workarounds.

Take note of the IP of your PC. In the following referred to it as **%local-ip%**.

Java JDK:

http://java.sun.com/javase/downloads/widget/jdk6.jsp

1. Select windows and click download.
2. Skip the registration step
3. Download and install to the directory of your choice (I now will call **%jdk-dir%**)

Apache HTTP Server:

http://httpd.apache.org/download.cgi

1. Select the new stable (non-alpha) version
2. Select the “Win32 Binary without crypto”
3. *Note: There will be a conflict between this service and Skype if you are using it. Please turn Skype off before installation.*
4. Install to the directory of your choice (I now will call **%apache-http-dir%**)

Apache Tomcat:

http://tomcat.apache.org/download-55.cgi

1. Select the “zip” option under “Core:”
2. Download and extract to a location (I now will call **%apache-tomcat-dir%**)

Eclipse:

http://www.eclipse.org/downloads/

1. Select the appropriate package for you (typically “Eclipse IDE for Java Developers”) and Windows (32-bit or 64-bit).

2. Select the appropriate mirror, download, and then extract to the directory of your choice (I now will call **%eclipse-dir%**)

Ports:
*Note: these steps can be done automatically by running **WinXPVista7FirewallConfig.bat** once the Rule Responder project has been downloaded from the SVN (located in the Rule Responder folder).

To Open a Window's Firewall Port (Windows XP):
Note: these steps will be very similar for Vista or 7
1. Control Panel, Security Center, Windows Firewall
2. Exceptions tab. This contains all forwarded ports.
3. Click add port, give it a name, and the port number, as well as TCP or UDP (do one of each just in case).

Ports Required by Rule Responder:
- Apache-HTTP-Server: 80
- Apache-Tomcat: 8080
- Mule: 8888, 60504
- RR OA's: 9990, 9992, 9998 (SP2011), 9993(SP2010),9994(PS),9995(WR),9996(SP2009),9997(SP2008),9999(WR2)

2.1.2 Apache HTTP Server

Installing the EA:

The Apache HTTP server starts its file directory under "%apache-http-dir%/htdocs". This is where you need to place any websites you wish to be installed.
Remember that an HTML file labeled "index.html" located under root will be defaulted to when your IP is navigated to.

1. Choose the EA scheme you wish to use**:**
   RuleML-2009: http://ruleml.org/RuleML-2010/RuleResponder/
   WellnessRules2: http://ruleml.org/WellnessRules2/RuleResponder.html
   PatientSupporter: http://ruleml.org/PatientSupporter/RuleResponder.html
   Additional examples can be found through: http://ruleml.org/RuleResponder

2. Download the webpage source + files (varies depending on your browser), to "%apache-http-dir%/htdocs". If you wish to have it as default, save as index.html, otherwise you may navigate to it. Firefox: File, save page as...

3. The page must now be told to direct its message to your PC and not the previous server. Edit the page and find the old IP address (approximately halfway down), replace it with "%local-ip%:8888" with the same port.

SymposiumPlanner-2011 Client:

SymposiumPlanner-2011 client employs servlets and needs to be deployed as a Web application. For installation, copy its archive from %RR-dir%/rules/symposiumPlanner2011/ SymposiumPlanner2011Client.war and paste it into the "%apache-tomcat-dir%/webapps" directory.

Before issuing queries by the client, please make sure the Rule Responder 3 and the Personal Agents of SymposiumPlanner-2011 have been well installed. In addition the address in rruleml.jsp should be directed to Mule server. The default address is: http://localhost:8888.

2.1.3 Eclipse & Mule

Prerequisites:

- Latest Version of Subclipse:
  Instructions: http://subclipse.tigris.org/servlets/ProjectProcess?pageID=p4wYuA

- Latest Version of Maven:
  1. Inside eclipse go to Help->Software Updates->Available Software
  2. Instructions: http://m2eclipse.sonatype.org/installing-m2eclipse.html
  3. Install all Maven components you see to be safe.
  4. Enable VM option for eclipse: Create a shortcut for eclipse or use an existing one and properties it
     Under target add at the end:
     -vm "%jdk-dir%\bin"
  5. Create a "settings.xml" file under the .m2 folder located in your user home directory (e.g. "C:\Documents and Settings\Administrator\.m2") with the following contents:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
   <localRepository>${user.home}/.m2/repository</localRepository>
  <interactiveMode>true</interactiveMode>
  <usePluginRegistry>false</usePluginRegistry>
  <offline>false</offline>

  <pluginGroups/>
  <servers/>
  <mirrors/>
  <proxies/>
  <profiles/>
  <activeProfiles/>
</settings>
```

Obtaining the Source Code:

1. Windows->Show View->Other->SVN->SVN Repositories
2. Right Click->New->Repository Location...
3. Add the Rule Responder SVN repository:
   The repository of Rule Responder 3 available at:

   https://mandarax.svn.sourceforge.net/svnroot/mandarax/RuleResponder3

   or the version before Rule Responder 3 at:

   https://mandarax.svn.sourceforge.net/svnroot/mandarax/PragmaticAgentWeb/

4. Navigate to the root -> Right click on RuleResponder3 (or PragmaticAgentWeb for the version before Rule Responder 3) -> Checkout

(leave everything as default)
*The project is now located under "%eclipse-workspace%/"
I will now refer to this as **%RR-dir%***

Configuring Mule:

1. Open the mule-all-config.xml file:
   "%RR-dir%/conf/mule-all-config.xml"
2. The most important piece to configure is the TOMCAT variable. Please make sure this reflects your Tomcat IP and port.
3. **Note:** The remaining settings need not be changed unless you make specific changes to the instantiations (e.g. the addition of an endpoint, modification of Prova location, OA port change, etc.).
4. Save, and compile the JAR.
   If Maven is installed correctly:
   a. Right click on project -> Run as -> Maven install
   b. Copy the produced "%RR- artifactId %-SNAPSHOT.jar"
   (located under %RR-dir%/target) file to "%RR-dir%/lib", where % RR-artifactId % corresponds to the artifactId of the project and it can be found in "pom.xml".

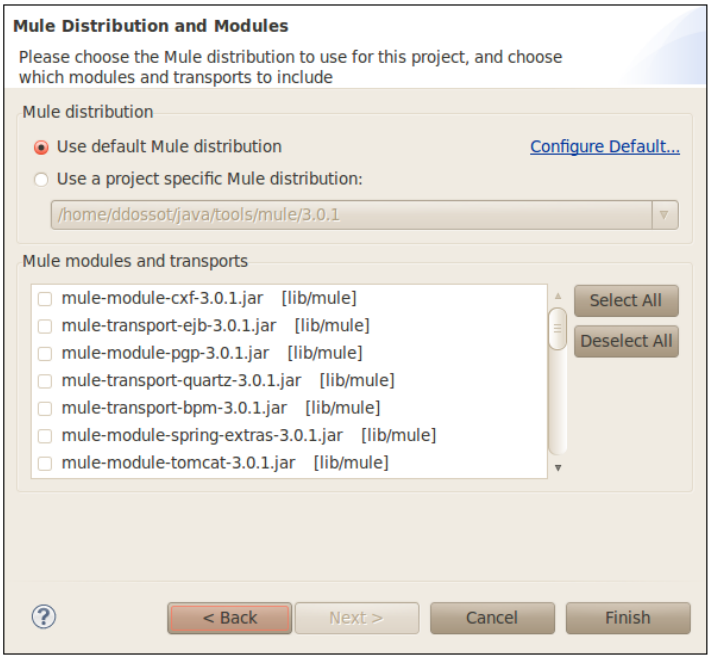
   If not:
   a. Select the packages "rules, src, repository, src/test/java, src/main/java, /src/main/resources, conf" -> Right click -> Export -> Java -> JAR File
   b. Name the jar file "% RR-artifactId %-SNAPSHOT.jar" and place in "%RR-dir%/lib"

2.1.4 Mule IDE Installation and Configuration (Optional)

Mule IDE is a development and testing environment based on Eclipse. You can install it if you want to develop a new instantiation of Rule Responder or test existing instantiations.

Instructions: http://www.mulesoft.org/mule-ide (You can find the latest installation guide of Mule IDE on this page)

Activating Mule IDE



To benefit from Mule IDE's features, you currently need to manually add Mule's library to your Eclipse project build path. For this, right click on project, select "Build Path > Add Library > Mule Classpath->Select All" and click "Finish" in the dialog shown above. Be sure to use a Mule distribution version that is the same as the one configured in your pom.xml.

Run Rule Responder with Mule IDE

To startup the Rule Responder in Eclipse IDE environment, right click on the Mule configuration file "%RR-dir%/conf/mule-all-config.xml", select "Run as > Mule Server".

2.1.5 Tomcat

All servlets are located under the webapps folder ("%apache-tomcat-dir%/webapps"). This is where we will be working.
All source code for the PAs is located under "%RR-dir%/personalAgents/%package%", where %package% corresponds to the instantiation package.
Class files for the PAs are located under "%RR-dir%/target/classes/%package%", where %package% corresponds to the instantiation package.

Add the JAVA_HOME environment variable to point to java install directory:

Properties on My Computer->Advanced->Environment Variables System Variables->New Variable Name: JAVA_HOME Variable Value: (JDK install directory, not bin just root of it, no quotes needed)

Configuring the PA Source:

1. Navigate to the PA configuration file
   (e.g. %RR-dir%/personalAgents/pa_Configuration/PAConfiguration.java")
2. Change the "tomcat" global variable to %local-ip%
3. If you made any other changes along the way or wish to change the location of online repository files, this is the place to reflect such changes.
   Additionally, the Prova file will need to be changed if the OWL ontology location has been modified.
4. Save, and now the class file is located in the target directory mentioned above.

Creating the Tomcat Servlets:

There are two ways to create the tomcat servlets, manually or automatically. It is not required to do both.

Automatically:

1. To install all the servlets from all the current Rule Responder instantiations (WellnessRules, PatientSupporter, SymposiumPlanner, etc) navigate to the %RR-dir% and run the batch file **InstallAllServlets.bat**.
2. You will be immediately prompted to enter %local-ip%
3. Once the batch file has completed its operation (takes about 2-3 minutes) copy all the folders that were just generated from %RR-dir%/PAServlets and paste them into the "%apache-tomcat-dir%/webapps" directory.

Manually (this series of steps must be done for each PA):

Most of these steps are general steps used from creating Tomcat servlets.

1. Create a new directory under "%apache-tomcat-dir%/webapps" which has the same name as the PA which you are implementing (e.g. "GeneralChairRuleML2009").
2. Create two more directories under the new one: "META-INF" & "WEB-INF"
3. Navigate to META-INF and create a new file "MANIFEST.MF" with the following contents:

```
Manifest-Version: 1.0
    Ant-Version: Apache Ant 1.6.5
    Created-By: 1.5.0_06-b05 (Sun Microsystems Inc.)
    Built-By: %your-name%
    Main-class: %main-class-name%
```

%main class name% = the name of the PA class (e.g. GeneralChairRuleML2009.class)

4. Navigate to the WEB-INF folder and create the file "config.xml" with the following contents:

```
<sign>
    <url>%local-ip%</url>
    <context>%class-name%</context>
    <delay>5000</delay>
    <log>yes</log>
    </sign>
```

%class-name% = The main class name (e.g. GeneralChairRuleML2009)

5. Navigate to the WEB-INF folder and create the file "web.xml" with the following contents:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <!DOCTYPE web-app
      PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
      "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
    <web-app>
      <display-name>%class-name%</display-name>
      <description>Takes incoming messages and excutes queries</description>
      <servlet>
        <servlet-name>%class-name%</servlet-name>
        <servlet-class>%package%.%class-name%</servlet-class>
      </servlet>
```

```
  <servlet-mapping>
    <servlet-name>%class-name%</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

%package% = The package which the main class is contained within (if any)
%class-name% = The main class name (e.g. GeneralChairRuleML2009)

Note: If you wish to run the servlet at startup (e.g. the WeatherUpdater) you need to add the following parameters:

```
  <servlet>
      ...
      <load-on-startup>1</load-on-startup>
  </servlet>
```

6. Navigate to the WEB-INF folder and create the folders “classes” and “lib”

7. Under the lib folder, copy all contents from “%RR-dir%/personalAgents/lib” to it.

8. Under the classes folder, copy the PA package you want to use
(e.g. “%RR-dir%/target/classes/pa_SymposiumPlanner/”)

9. Under the classes folder, copy the PA configuration package
(e.g. “%RR-dir%/target/classes/pa_Configuration/”)

2.1.6 Startup

It is recommended that you create your own batch file which takes care of this in single steps

Starting Up:

1. Start Apache-HTTP-Server:
“%apache-http-dir%\bin\httpd.exe"

2. Start Tomcat Server:
“%apache-tomcat-dir%\bin\startUp.bat”

3. Start Mule Server:
“%RR-dir%\startUp.bat”

2.2 Linux Server Environment

This section explains the various processes required to use the Rule Responder system. It details installing, starting up, shutting down, and debugging the Rule Responder system.

2.2.1 General Prerequisites

Before attempting any procedure below, insure the following:

1. Your Linux account has the proper privileges on the server. It should have permission to start and stop Tomcat from running as well as have access to the location of the Rule Responder folder (or soon to be location of it if you are installing for the first time).

2. You are logged into the server using an SSH client (For Windows, Putty is recommended, most Linux distributions already have an SSH client integrated in).

2.2.2 Installing Rule Responder

Prerequisites

1. You have gone through the section 1.1 documentation intended for a local installation.
2. FTP client must be installed. Filezilla is recommended.
3. The system administrator for the server has allowed external access to port 8888.

1. In the Mule configuration file (mule-all-config.xml) change the TOMCAT and MULE_PORT variables to match the server's local IP.
2. In PAConfiguration, replace the current IP with the server's local IP. The operating system variable should be changed to "Linux" and the WeatherRespositoryVariables should have their IP changed to the local IP of the server as well.
3. Once everything has been updated with server information, perform a maven install and move the result file into the lib folder like you normally would.
4. FTP into the server and go to the folder where you are going to put Rule Responder. Locally, open up the Rule Responder folder. Copy over all the files (not folders) in the Rule Responder folder over to the server. Additionally, copy the lib folder as well.
5. Go to the local versions of your tomcat PA servlets in your tomcat webapps folder. Backup the current versions for future use.
6. Go through and change all the IP's in the servlets to the local IP for the server.
7. On the server, navigate to the tomcat webapps directory. The default installation folder for this is "/usr/local/tomcat/webapps". Place all of the updated PA servlets on the server.
8. Proceed to Section 1.2.3 **Starting Rule Responder** section.

2.2.3 Starting Up Rule Responder

1. Build the your project with Maven via:
   Right Click on Project->Maven->Install
   Copy "target/%RR-artifactId %-SNAPSHOT.jar" to "lib"
2. To start up Mule, navigate to the Rule Responder folder and enter the command "*./startup 0 &*". Note: If Mule is running and needs to be restarted, consult the next section 1.2.4 Shutting Down Rule Responder as to how to shut down Mule before starting it again.
3. To startup tomcat enter the command "*/etc/init.d/tomcat start*". Reminder: This will not work if you do not have the correct permissions.

2.2.4 Shutting Down Rule Responder

1. As of right now, the best way to stop Mule is to find its process ID in the list of running processes (this is being worked on). Enter the command "ps aux".
2. Look for the process ID of the command called "java -Xms512m -Xmx512m -XX:NewSize=256m -XX:MaxNewSize=256m ws.prova.mule.impl.ESBManager start mule-all-config.xml".
3. Enter command "*kill -9* <ProcessIDHere>". Example: "*kill -9 14759".*
4. To stop tomcat enter the command "*/etc/init.d/tomcat stop*". Reminder: This will not work if you do not have the correct permissions.

2.2.5 Debugging Rule Responder

1. Open 2 instances of your SSH client and log into the server on both of them. For the rest of these instructions we will call these instances **terminal 1** and **terminal 2**.
2. In terminal 1, navigate to your Rule Responder installation folder. Enter the command "tail –f mule.log". Terminal 1 now presents all log activity as it happens. This shows us what the Organizational Agent is doing in the system.
3. In terminal 2, enter "*tail -f <tomcat directory>/logs/catalina.out*". In most cases, this is will look like "*tail –f /usr/local/tomcat/logs/catalina.out*". This presents all log activity on the Personal Agent side of the system.
4. With both of these windows open, you can now view all error messages generated by the Rule Responder system as it is running.

3 Enterprise Service Bus (ESB) - Mule

The Enterprise Service Bus (ESB): Mule

Rule Responder uses an Enterprise Service Bus (ESB) called Mule to transfer data. Mule transfers this data via "data endpoints". In the case of Rule Responder, each agent (EAs/OA/PAs) has their own endpoint for which data will travel upon. Additionally, Rule Responder only uses HTTP as a transfer protocol but others can certainly be implemented. For a more thorough description and implementation of Mule ESB please see this presentation: http://ruleml.org/RuleResponder/UsingMuleESB.

Rule Responder requires Mule running on your host server. It is provided in the source package. Otherwise, the full ESB download of Mule should contain all of the information needed to install the Mule ESB on your server: http://www.mulesoft.org/display/MULE/Download+Mule+CE.

As we mentioned in Section 2, the Rule Responder 3 supports Mule 3.1.0, which was upgraded with significant changes compared to Mule 1.4.1. Here, we just describe key concepts in Mule 1.4.1. For information of Mule 3.1.0, please find at the Section of this guide: Migrating Mule 1.x to 3.0 for RuleResponder.

The Mule 1.4.1 Config File

All endpoints and descriptors used with Mule are defined in the mule-all-config.xml file. Therefore, any changes must go here. Changes are then applied when the library of project is rebuilt and implemented. The config file is located under:"%RR-dir%->conf->mule-all-config.xml

- **Endpoint Identifiers:**
  Endpoint identifiers describe the endpoints that are used in the ESB. Each identifier has a name and an address.
  - **Endpoint Identifiers - name:**
    The name of the endpoint is **identically** linked to the agent name that is derived by both the OWL ontology and Prova. Therefore, they must be the same.
    (Note that the names must be unique as well)
  - **Endpoint Identifiers - value:**
    The value of the endpoint is the address. In the case of Rule Responder, we always use:

"http://**%local-ip%**/..." where "..." represents the name of the tomcat servlet.

- **Mule Descriptors:**

  Each OA is defined in the mule config file as a mule-descriptor. For example, WellnessRules is described as:

```
<mule-descriptor name="WellnessRules"
        implementation="ws.prova.mule.impl.ProvaUMOImpl" singleton="true">
        <inbound-router>
                <endpoint address="jms://topic:wellnessRules"/>
                <endpoint name="httpEndpointWR" address="http://localhost:9995">
        </endpoint>
</inbound-router>
        <properties>
                <property name="rulebase" value="/use_caseWellnessRules/Wellness-
                Rules/WellnessRules-Responder.prova" />
        </properties>
</mule-descriptor>
```

  - **Mule Descriptors - name:**

    The name of the descriptor corresponds to the "XID" that is passed in with Rule Responder queries. It is the name of your Organizational Agent.

    <mule-descriptor name="WellnessRules"

    (Note that the names must be unique)

  - **Mule Descriptors - Implementation:**

    The implementation attribute of the descriptor is the java file that is executed when a message is received. In the case of RuleResponder, *ws.prova.mule.impl.ProvaUMOImpl.java* is used. It will use the OA's Prova knowledge base (defined under property) to handle the message.

    implementation="ws.prova.mule.impl.ProvaUMOImpl" singleton="true

    Additionally, some instantiations need a longer period of time for their longest query. This time should be tied to the time it takes for the OA to time-out. Ideally the time-out time should be as short as possible so the user knows the query was not successful. Timeout times can be changed for an individual use-case in *ws.prova.mule.impl.ProvaUMOImpl.java* as well. For example:

```
if(temp.contains("RuleML-2010")) {
                        System.out.println("Timeout
increase to 5 minutes for RuleML-2010. Will be reset
upon new query.");
                        timeout=300000;
```

  - **Mule Descriptors - endpoint name:**

    This endpoint name will be specific for every OA. For example, RuleML-2009 uses *http://localhost:9996* while WellnessRules uses *http://localhost:9995*. Consult with the server admin to see which port is best for you to use.

<endpoint name="httpEndpointWR" address="http://localhost:9995">

(Note that the endpoint name must match the endpoint definitions described previously)

- **Mule Descriptors - properties:**

  Properties are used to further define the descriptor. In the case of Rule Responder, they are used to point to the Prova knowledge base to be used.

  <property name="rulebase" value="/use_caseWellnessRules/Wellness-Rules/WellnessRules-Responder.prova"/>

4 Organizational Agent

4.1 Prova Knowledge Base

Organizational Agent – Prova Knowledge Base:

The Prova knowledge base describes rules that help the Organizational Agent delegate queries to the appropriate Personal Agents in the community. This document describes the framework that Rule Responder uses and some of the general performatives.

- **Location:**
  Each Prova knowledge base file is pointed to by the Mule config file. Generally the KB location has the following format:

  %RR-dir%\rules\use_case*Instance**NameOfKB.prova*

- **Committing Changes:**
  To commit a change to the KB, you need to rebuild the .jar via Maven, copy it to "%RR-dir%/lib" and restart Rule Responder.

- **Assigning the OWL Ontology:**
  A fact is used to point Prova to the OWL ontology used with the particular Rule Responder instance.
  e.g.:

  ```
  %import external ontology of responsibility assignment matrix
  import("http://ruleml.org/WellnessRules/files/WellnessRules.owl").
  ```

- **Defining Interfaces:**
  Interfaces describe what queries are expected to be issued to the Rule Responder. They have the following format:

  ```
  interface(performative(Performative),performative("?"),"description").
  ```

  For example:

  ```
  interface(getContact(Topic,Request,Contact),
  getContact("+","+","-"),"request personal contact information for a certain Topic and Request regarding RuleML-2008").
  ```

  The first argument is the performative. The second is the relation, with +'s and -'s to represent constants and variables, respectively. The last argument is the description of the performative.

  These interfaces must be written as facts for every expected query, otherwise a "no public interface" message will be returned.

- **General processMessage Rule:**
  A query is handled by the processMessage rule and its conclusion looks like the following:

  ```
  processMessage(XID,From,UserName,performative(Performative)) :-
  ```

  XID = The name of the OA (e.g. WellnessRules).
  From = The name of the endpoint

Primitive = Username (e.g. User)
performative = The relation name surrounded by <Rel>
Performative = The contents of the relation (<Ind>'s, <Vals>'s, <Expr>'s, <Plex>'s, etc.)

Optionally, you then can check the free and bound variables to restrict the rule to certain <Ind>'s (constants) and <Var>'s (variables):

free(ProfileID),
bound(InOut,

free = A variable (<Var>)
bound = A constant (<Ind>)

A rule is then used to "look up" the responsible Agent:

assigned(XID,Agent,Responsibility,Role),

XID = The name of the OA (e.g. WellnessRules)
Agent = Will be the name of the found Agent
Responsibility = A responsibility name for the query (provided in the query somehow).
Role = What kind of role the responsibility will be matched to.

The query is then sent to the Agent that was found:

sendMsg(XID,esb,Agent,Type,performative(Performative)),

XID = The name of the OA (e.g. WellnessRules)
Agent = Will be the name of the found Agent
Type = "query"
performative = The relation name of the query
Performative = The contents of the relation

The answer is then received from the Agent:

rcvMult(XID,esb,Agent,Type,performative(Performative)),

XID = The name of the OA (e.g. WellnessRules)
Agent = Will be the name of the found Agent
Type = "answer"
performative = The relation name of the query
Performative = The contents of the relation

The answer is then sent back to the EA:

sendMsg(XID,esb,From,"answer",performative(Performative),

XID = The name of the OA (e.g. WellnessRules)
Agent = Will be the name of the EA endpoint
Type = "answer"

performative = The relation name of the query
Performative = The contents of the relation

4.2 OWL Ontology

Organizational Agent - OWL Ontology:

The OWL ontology is used to delegate queries to the responsible Personal Agents. It is called upon by the Prova knowledge base.

- **How to Implement Changes:**
  Simply replace the file.

- **Responsible:**
  This is the most commonly used aspect of the OWL ontology. Each topic queried in the ontology has a PA which is responsible for it. It is defined by:

  …

  <owl:ObjectProperty rdf:ID="responsible">
  <rdfs:comment>
  This role conducts the actual work/owns the problem. There should be only one R. If multiple R s are listed, then the work needs to be further subdivided to a lower level.
  </rdfs:comment>
  <rdf:type rdf:resource="owl#FunctionalProperty"/>
  <rdfs:domain rdf:resource="#*WhatDomainResponsibilityIsIn-eg-OAName*"/>
  <rdfs:range rdf:resource="#Responsibility"/>
  </owl:ObjectProperty>

  <owl:Class rdf:ID="Responsibility">
  <rdfs:subClassOf rdf:resource="#*TheOrganizationalAgentName*"/>
  </owl:Class>

  <Responsibility rdf:ID="*A-PA-Name*"/>

  <*SomeUniqueName* rdf:ID="*TheOrganizationalAgentName* ">
  <responsible rdf:resource="*#A-Responsible-Topic*"/>
  </*SomeUniqueName*>

  …

Note that the hash (#) sign denotes the remainder of the ontology, e.g.:
"*TheOrganizationalAgentName_A-Responsible-Topic*"

Here is an **example** for **WellnessRules** following the one seen above:

…

<owl:ObjectProperty rdf:ID="responsible">
<rdfs:comment>
This role conducts the actual work/owns the problem. There should be only one R. If multiple Rs are listed, then the work needs to be further subdivided to a lower level.
</rdfs:comment>

```
        <rdf:type rdf:resource="owl#FunctionalProperty"/>
        <rdfs:domain rdf:resource="#Activity"/>
        <rdfs:range rdf:resource="#Responsibility"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="Activity_Walking">
        <rdfs:subClassOf rdf:resource="#Activity"/>
</owl:Class>

<Activity_Walking rdf:ID="Walking">
        <responsible rdf:resource="#Walking"/>
</Activity_Walking>

...
```

5 Personal Agents

Personal Agent Requirements:

Tomcat installed on host server (see Section 1 for installation information).
You can find the tomcat documentation here: http://tomcat.apache.org/index.html

Knowledge base hosting is also required (e.g. for WellnessRules profiles). These can be on any host server as they are transferred via HTTP. For detailed implementation information and other tips, please see the remaining documentation.

5.1 How to Implement

Personal Agents – How to Implement:

Personal Agent servlets are used by Tomcat, which makes them accessible through HTTP.

- **Location on Server:**
  Every servlet has their own unique folder on your Tomcat server:
  %apache-tomcat-dir%/webapps/

  e.g.
  %apache-tomcat-dir%/webapps/GeneralChairRuleML2009

  Generally, the folder names match the main Class of the servlet
  (typically a PA has only one "unique" Class).

- **Contents of the Servlet Folder:**
  - **META-INF:**
    The META-INF folder contains the MANIFEST.MF.
    - **META-INF/MANIFEST.MF:**
      This file must contain the main class, e.g.: Main-class: GeneralChairRuleML2009.class
      If this is not correctly written, then the servlet will fail.

  - **WEB-INF:**
    This folder houses the config files of the servlet, as well as the main Classes and libraries.
    - **WEB-INF/config.xml:**

      This file must have the following format. If this is not correctly written, then the servlet will fail.

      ```
      <sign>
        <url>Address</url>
        <context>MainClass</context>
        <delay>integer</delay>
        <log>yes/no</log>
      </sign>
      ```

      e.g.

      ```
      <sign>
        <url>http://localhost:8080</url>
        <context>GeneralChairRuleML2009</context>
        <delay>5000</delay>
        <log>yes</log>
      </sign>
      ```

- **WEB-INF/web.xml:**
  This file must have the following format. If it is not correctly written, then the servlet will fail.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
<display-name>NameToAppearInURL </display-name>
<description>ADescription</description>
        <servlet>
                <servlet-name> MainClassName </servlet-name>
                <servlet-class> MainClassName </servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>NameToAppearInURL </servlet-name>
                <url-pattern>/</url-pattern>
        </servlet-mapping>
</web-app>
```

  e.g.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
        <display-name>GeneralChairRuleML2009</display-name>
        <description>Takes incoming messages and excutes  queries</description>
        <servlet>
                <servlet-name>GeneralChairRuleML2009</servlet-name>
                <servlet-class>GeneralChairRuleML2009</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>GeneralChairRuleML2009</servlet-name>
                <url-pattern>/</url-pattern>
        </servlet-mapping>
</web-app>
```

- **WEB-INF/lib:**
  This folder contains all of the .jars that the Class will use. These library .jars can be found under "%RR-dir%/personalAgents/lib"

  If ANY .jar's are missed, the servlet will fail.

- **WEB-INF/classes:**
  This is where the main Class file is to be placed as well as any supporting Classes. If packages exist, then folders must be used accordingly. For example, SymposiumPlanner-20XY and WellnessRules use the resources_PA package and so the class folder must be copied over which is located in

"%RR-dir%/target/classes/resources_PA". Additionally, the PAs themselves can be found under "%RR-dir%/target/classes".

5.2 Servlets

Personal Agents – Servlets:

Each personal agent in the Rule Responder system is implemented via a Java servlet. However, there is a major difference between them. For an example we examine the SymposiumPlanner and WellnessRules (PatientSupporter and WellnessRules2 generally have the same servlet layout as WellnessRules) instantiations.

- **SymposiumPlanner Servlets:**
  SymposiumPlanner servlets are much less complex, yet more unorganized than their WellnessRules counterparts. They have the following format:

MainClass

MessageGenerator *MessageParser* *QueryBuilder*

MainClass: This Class handles the incoming query by converting it to POSL, executing POSL on it, and converting the answer(s) to RuleML before sending them back to the OA. It does so via the three supporting classes:

MessageGenerator: Generates messages for HTTP transfer from the provided RuleML. There is no particular format in the main class itself. It calls upon OO jDREW after first retrieving its knowledge base.

MessageParser: Parses the RuleML query.

QueryBuilder: Helps convert RuleML to an object oriented format.

** Please see the documentation within the code for detailed information**

- **WellnessRules Servlets:**
  Like SymposiumPlanner, WellnessRules servlets consist of a main class and many supporting classes. However, the main class and supporting classes are now capable of both OO jDREW and Euler executions.

MainClass

MessageGenerator *MessageParser* *QueryBuilder*

Now introduces additional supporting classes:

GeneralHandler *N3Handler* *POSLHandler*

***GeneralHandler**:* This class handles general method calls for PA's such as retrieving the time and date, and determining a PA's responsible profiles and user's requested profile.

***N3Handler**:* This class handles N3 method calls for PA's such as generating N3 profiles parsing a RuleML query, and answering an N3 query with Euler EYE.

***POSLHandler**:* This class handles POSL method calls for PA's such as generating POSL profiles parsing a RuleML query, and answering a POSL query with OO jDREW.

** Please see the documentation within the code for detailed information**

5.3 Profile Responsibility

Personal Agents – Profile Responsibility Matrix:

The Profile Responsibility Matrix (PRM) is a new concept introduced to Rule Responder via the newest insanitation, WellnessRules. Very similar to the OWL ontology used with the OA, this is an XML file similar to an ontology, which designates engine execution to specific profiles.

- **Format:**
  For WellnessRules, the PRM has the following format:

```
<WellnessRules>
        <Activity>
                <Walking>
                        <ResponsibleProfile name="p0001" format="posl"/>
                        <ResponsibleProfile name="p0002" format="n3"/>
                        <ResponsibleProfile name="p0003" format="posl"/>
                </Walking>
        </Activity>
</WellnessRules>
```

  This means that the "Walking" PA is responsible for p0001, p0002, and p0003. Additionally, it will use the format information to execute the appropriate engine (OO jDREW for POSL and Euler for N3).

- **Implementation:**
  The retrieval of this XML file and its handling is done via the calling of getResponsibleProfiles() in GeneralHandler.java.

```
public static String[] getResponsibleProfiles
    (String, configXML,String activity)
```

5.4 Javadoc - Servlets

http://ruleml.org/WellnessRules/PADocumentation/index.html

6 Computing Agents

Computing agents (or CAs) can be seen as an (often low level) agent that performs an automated (computing) task. Often, CAs are used to gather and transform data from (Google, Amazon, ...) services. The WellnessRules2 use case of Rule Responder (http://ruleml.org/WellnessRules2/RuleResponder.html) employs one agent of this category: the 'dynamic weather' CA. Invoked by the Organizational Agent (OA), the CA queries and wraps the Google Weather service, then parses and returns the 7 day forecast as a knowledge base in N3 and POSL. (Any data that was retrieved previously that is no longer correct is overwritten.) The CA acts as an extension to a global knowledge base (for example: http://ruleml.org/WellnessRules2/resources/OA/WR-Global.posl). Computing Agents generally run by themselves periodically. The time between intervals is specified in pa_Configuration.java. Implementation source code can be found with comments at

(https://mandarax.svn.sourceforge.net/svnroot/mandarax/RuleResponder3)

Or

(http://mandarax.svn.sourceforge.net/viewvc/mandarax/PragmaticAgentWeb/).

CAs are comparable to Personal Agents. Their output is meant to assist the OA (in the example, expanding the global knowledge base with weather information) in answering the query from the EA. They are designed to perform very specific tasks (like the weather example above) that may involve invoking services independently from the rest of the virtual organization.

Tomcat implementation of CAs is identical to PAs. Please see the section 5 on Personal Agents for more information.

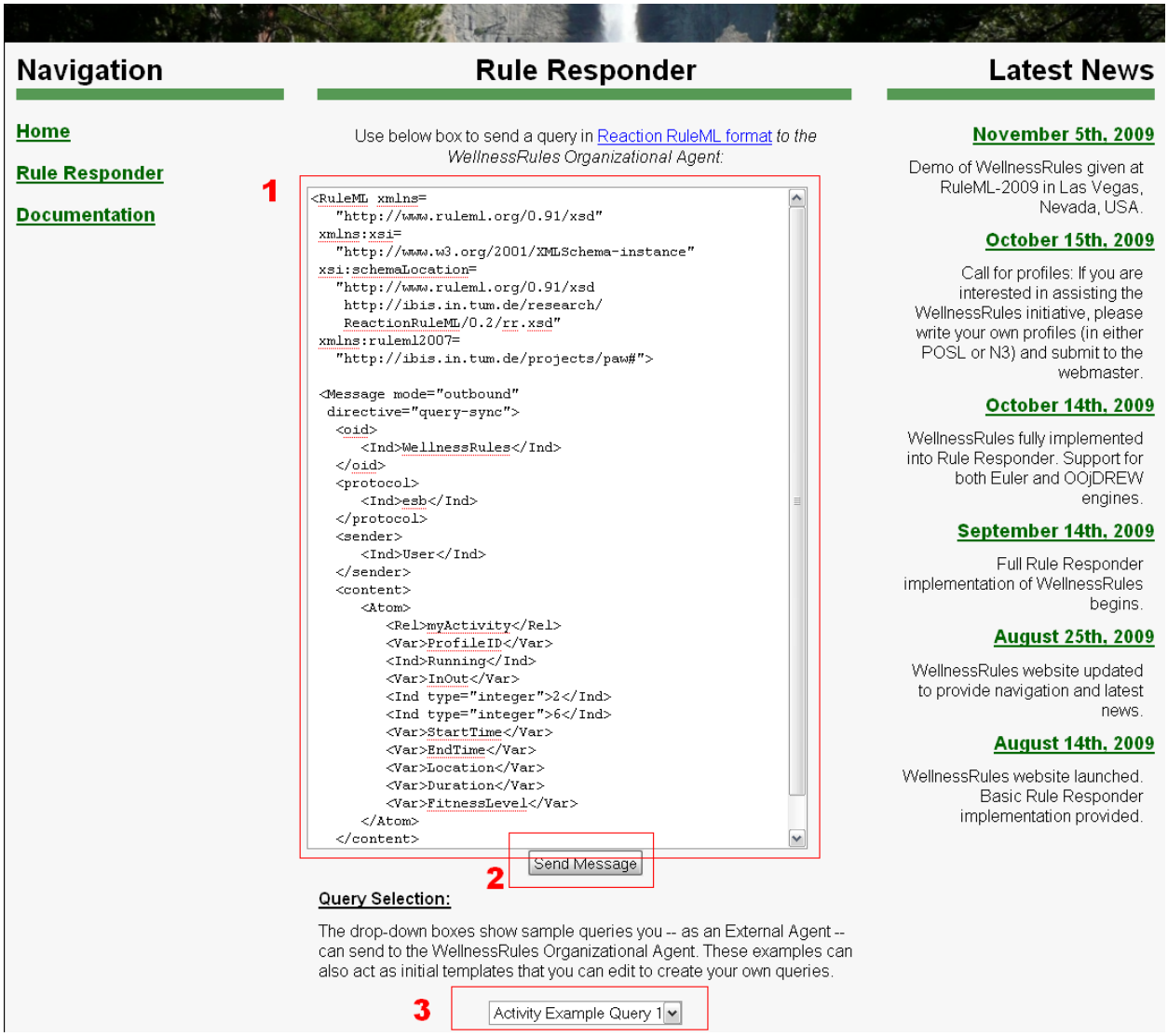
7 External Agents

You may set up your own version of an EA for Rule Responder; be it a website (typically), stand-alone application, etc. However, the documentation for the EA in this package pertains to the website implementation.

7.1 Basic Websites for External Agents

Rule Responder - Websites:

Each Rule Responder uses a website as its EA which will accept a query in RuleML format:



(1) **Query Window:**

The query window is where the RuleML query is written. It is implemented via the following HTML code:

<form name="form2" method="get" **action**="http:***IP***:8888">
<input name="Horde" value="3a466331bca877eeb014be19eee68790" type="hidden">
<textarea name="box1" **cols**="57" **rows**="36" **WRAP**=OFF></textarea>

action = The address of the recipient of the message. (This is the Mule instance on the external development server).

IP = Replace this with the IP of the Rule Responder server

cols = The width of the query box

rows = The height of the query box

WRAP = Word wrapping

(2) **Submit Button:**

Submits the query to the OA via the following HTML code:

```
<input value="Send Message" type="submit">
```

(3) **Query Selection:**

Every Rule Responder website provides examples for users to try out. They are full-fledged RuleML queries which are guaranteed to work with the system. It is implemented via the following HTML code (example):

```
<form name="form1" method="post">
        <div align="center">
                <select name="select1" onchange="elementSelected()">
                        <option value="A">Activity Example Query 1</option>
                        <option value="B">Activity Example Query 2</option>
                        <option value="C">Activity Example Query 3</option>
                </select>
        </div>
</form>
```

elementSelected() is implemented via javascript:

```
function elementSelected(){

   var choice = document.form1.select1.selectedIndex;

   var choice0 = "<Atom>" + "\n        " +
                 "<Rel>myActivity</Rel>" + "\n        " +
                 "<Var>ProfileID</Var>" + "\n        " +
                 "<Ind>Running</Ind>" + "\n        " +
                 "<Var>InOut</Var>" + "\n        " +
                 "<Ind type=\"integer\">2</Ind>" + "\n        " +
                 "<Ind type=\"integer\">6</Ind>" + "\n        " +
                 "<Var>StartTime</Var>" + "\n        " +
                 "<Var>EndTime</Var>" + "\n        " +
                 "<Var>Location</Var>" + "\n        " +
                 "<Var>Duration</Var>" + "\n        " +
                 "<Var>FitnessLevel</Var>" + "\n      " +
                 "</Atom>" + "\n";

        var messageHeader =
 "<RuleML xmlns=\n \"http://www.ruleml.org/0.91/xsd\""+"\n"+
        "xmlns:xsi=\n\"http://www.w3.org/2001/XMLSchema-instance\""+ "\n"+
        " xsi:schemaLocation=\n\"http://www.ruleml.org/0.91/xsd" + "\n   " +
        "http://ibis.in.tum.de/research/"+"\n" +       "ReactionRuleML/0.2/rr.xsd\""+"\n"+
        "\n " + "<Message mode=\"outbound\"" + "\n  " +
        "directive=\"query-     sync\">" +
"\n   " + "<oid>" + "\n      " +
"<Ind>WellnessRules</Ind>" + "\n   " +
```

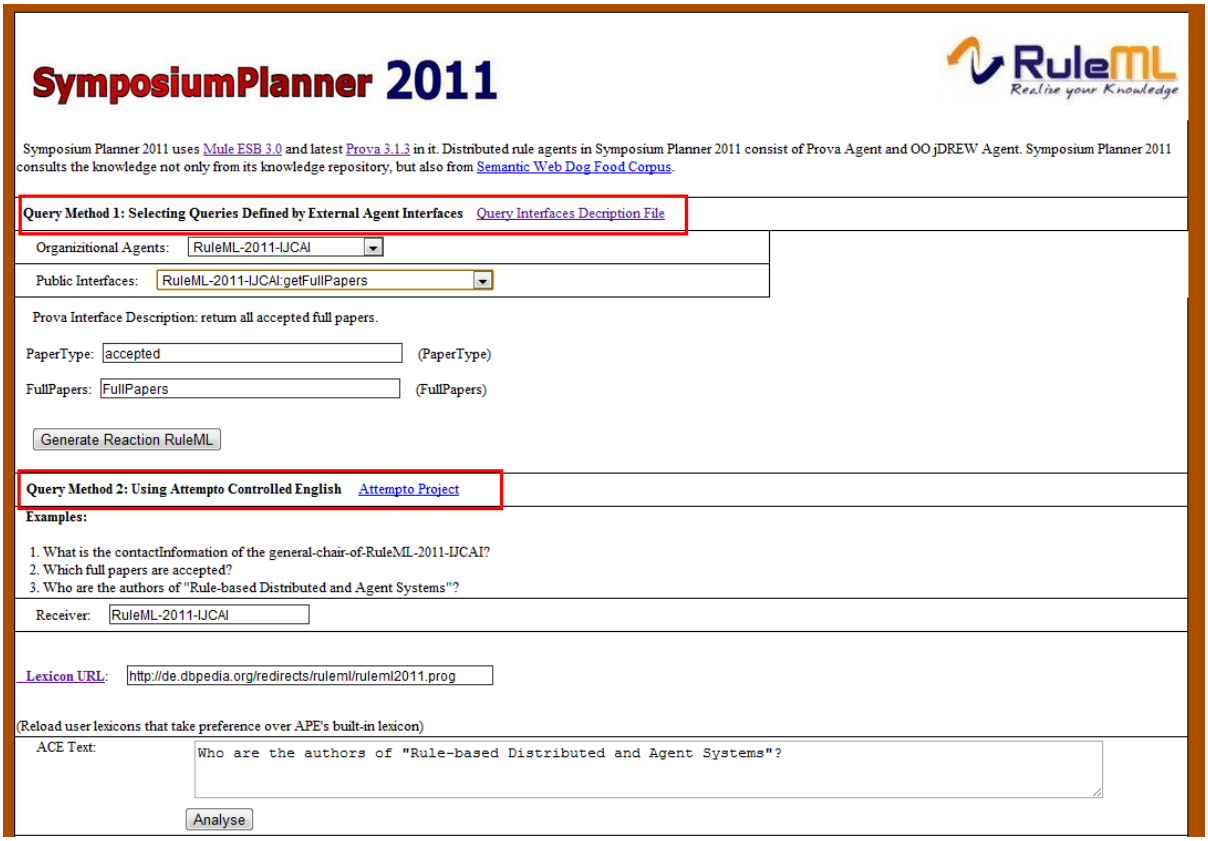
```
"</oid>" + "\n   " +
"<protocol>" + "\n     " +
"<Ind>esb</Ind>" + "\n   " +
"</protocol>" + "\n   " +
"<sender>" + "\n     " +
"<Ind>User</Ind>" + "\n   " +
"</sender>" + "\n   " +
"<content>" + "\n     ";

var messageFooter = "   " + "</content>" + "\n " +
        "</Message>" + "\n" +
        "</RuleML>";

        if(choice == 0){
        document.form2.box1.value = messageHeader + choice0 + messageFooter;
        document.getElementById('description1').innerHTML = dChoice0;
    }
```

7.2 Advanced Websites for External Agent

In SymposiumPlanner-2011, we developed a new client, which employs servlets and needs to be deployed as a Web application. It facilitates users to issue queries in two ways: using Attempto Controlled English (ACE) or selecting query templates based on the RuleML Interface Description Language (RuleML IDL), as shown in following figure.



For installation, copy its archive from %RR-dir%/rules/symposiumPlanner2011/ SymposiumPlanner2011Client.war and paste it into the "%apache-tomcat-dir%/webapps" directory. Then restart Tomcat server if it has been already started .

Before issuing queries by the client, please make sure the Rule Responder 3 and the Personal Agents of SymposiumPlanner-2011 have been well installed. In addition the address in rruleml.jsp should be directed to Mule server. The default address is: http://localhost:8888.

7.3 RuleML Queries

Rules Responder - RuleML Queries:

Rule Markup Language (RuleML) is the language used for rule interchange. In the case of Rule Responder, RuleML is used as a generic query language, which will be transformed to Prova, POSL, and N3. The following are the basics for Rule Responder's use of RuleML:

- **Message Header:**
  Contains the namespaces used in the query (they are the same across all Rule Responder).

  <RuleML xmlns=http://www.ruleml.org/0.91/xsd
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:schemaLocation=http://www.ruleml.org/0.91/xsd
  http://ibis.in.tum.de/research/ReactionRuleML/0.2/rr.xsd
  xmlns:ruleml2007="http://ibis.in.tum.de/projects/paw#">

- **Message Footer:**
  Contains the remainder of the message following the payload.
  </Message>
  </RuleML>

  - **Message Payload:**
    This is where the query is held.

  - **Message Payload - Header:**
    The head of the message payload is the same throughout.

    <Message mode="outbound" directive="query-sync">

  - **Message Payload - OID:**
    Contains the name of the Organizational Agent as a constant.

    <oid>
    <Ind>WellnessRules</Ind>
    </oid>

  - **Message Payload - Protocol:**
    The protocol used for message transfer (almost always esb).

```
<protocol>
        <Ind>esb</Ind>
</protocol>
```

- **Message Payload - Sender:**
  The message must have a unique sender (username). Functionality has yet to be implemented.

```
<sender>
        <Ind>User</Ind>
 </sender>
```

- **Message Payload - Content:**
  This is the query itself. It can contain X number of Atoms.

```
<content>
...
</content>
```

- **Message Payload - Atom:**
  An atom of the query (all instances only use one).

```
<Atom>
...
</Atom>
```

- **Message Payload - Query:**
  A query has a single relation name, followed by constants, variables and complex expressions.

<Rel> = The relation name
<Ind> = Individual constant
<Var> = Variable
<Expr> = Complex expression
e.g.:

```
        <Rel>myActivity</Rel>
        <Var>ProfileID</Var>
        <Ind>Running</Ind>
        <Var>InOut</Var>
```

Looks like the following in Prolog:
myActivity(ProfileID,running,InOut).

or in POSL:
myActivity(?ProfileID,Running,?InOut).

or in N3:

```
_:myActivity
        rdf:type :MyActivity;
```

```
:profileID          ?ProfileID;
:activity  :Running;
:inOut     ?InOut.
```

8 Instantiations

8.1 SymposiumPlanner

As the semantic web becomes more prominent, it is important to apply these new ideas to existing organization models to demonstrate efficiency and effectiveness. SymposiumPlanner is a series of use cases based on the RuleML Symposium series created with Rule Responder (RR).

Using Friend of a Friend (FOAF) profiles, each chair position (general chair, panel chair, etc) has a Personal Agent (PA). Each PA has a knowledge base containing the responsibilities of the position in order to answer queries relevant to the chair's role.

8.1.1 Upgrading SymposiumPlanner

Upgrading SymposiumPlanner:
Mule:

Changing ConfigFile:
1. You must add a new OA (e.g. RuleML-2010) as a Mule Descriptor. It will be placed on a new port (see ESB guide for an example).
2. You must add a new PA endpoint for each PA you are adding (see ESB guide for an example).

PAs (example: PublicityChair): Initial SetUp
1. Create the following directories on your tomcat server:
%apache-tomcat-dir%/webapps/**PublicityChairRuleML20XY**
2. Copy and paste all files and folders from a previous version's corresponding root folder (as named above).
3. Modify the file: %apache-tomcat-dir%/webapps/PublicityChairRuleML2009/WEB-INF/**config.xml** By changing the context to contain the correct version year. I.E: <context>PublicityChairRuleML2009</context> => <context>PublicityChairRuleML2010</context>
4. Modify the file: /usr/local/apache-tomcat-6.0.10/webapps/PublicityChairRuleML2009/WEB-INF/**web.xml** By finding and replacing all occurrences of the old version year and replace with the new version year.
5. Modify the file: /usr/local/apache-tomcat-6.0.10/webapps/PublicityChairRuleML2009/META-INF/**MANIFEST.MF** By changing the "Main-class" declaration to correspond to the new version year. I.E: Main-class: PublicityChairRuleML2009.class => Main-class: PublicityChairRuleML2010.class

New Servlet
1. Create a new java class named "**PublicityChairRuleML20XY**" to correspond to the new version year.
2. Copy and paste all information from a previous PublicityChair servlet to the new class.
3. Find and replace all occurrences of the old version year with the new version year.
4. Modify the global "port" value to correspond to the port chosen to use in the mule config file.

POSL & RDF Files
1. Create a new directory for the new version's POSL and RDF files which are referenced by the servlet:
/var/www/jdrew.org/oojdrew/**ruleSetsRuleML-20XY**
2. Create a new file in this directory named "**publicityChairOntologyRuleML-20XY.rdf**".
3. Copy and paste a previous version's contents to this new file. (no changes need be made)
4. Create a new file in this directory named "**publicityChairRuleML-20XY.posl**".
5. Copy and paste a previous version's contents to this new file.

6. Find and replace all occurrences of the old version year with the new version year.

OA Changes:

Adding a new use case:
1. Create a new folder under "%RR-dir%/rules/" which will contain the prova file your mule descriptor points to.
2. Replicate a previous SymposiumPlanner file system (e.g. copy paste).
3. Modify the "RuleML-20XY-Responder.prova" (which the mule descriptor points to) to the appropriate year.
4. Now, you may modify the queries and interfaces, owl files, etc. to reflect changes you are making to 2010. Foy symmetry, all references for 20XY should reflect the current year.

ALL CHANGES ARE COMPLETE FOR PUBLICITYCHAIR. Now you can create your own knowledge bases and rules in the OA/PA.

8.2 WellnessRules

With an ever-increasing emphasis on exercise and diet for individuals around the world, it becomes necessary to find new and innovative means to provide support for these wellness-related practices. WellnessRules (WR) is a new use case created with Rule Responder (RR) with the intention of forming an online- interactive wellness community.

As in Friend of a Friend (FOAF), people can choose a (community-unique) nickname and create semantic profiles about themselves, specifically their wellness practices, for their own planning and to network with other people supported by a system that `understands' those profiles.

These profiles can currently be written in either **POSL** or **N3**, which are handled by the engines OO jDREW and Euler respectively. Therefore, the interoperation between Prolog and N3 is realized via this unique use case.

8.2.1 Knowledge Base Signatures

Knowledge base signatures are available through the WellnessRules website.
(http://ruleml.org/WellnessRules/Signatures)

8.2.2 WellnessRules2

WellnessRules2 continues the work of WellnessRules using Rule Responder (RR) to form an online-interactive wellness community. It presents new features such as the ability to use dynamic weather data parsed into a rule format, complex expressions which allow facts to be much more useful in regards to being used in rules, and typed variables which allows users to use the Wellness Taxonomy to query for subclasses of activities (not yet fully implemented).

All further development of WellnessRules will be completed in WellnessRules2.

8.2.3 Knowledge Base Signatures

Knowledge base signatures are available through the WellnessRules2 website.
(http://ruleml.org/WellnessRules2/Signatures)

8.3 PatientSupporter

Patients are increasingly seeking interaction in support groups, which provide shared information and experience about diagnoses, treatment, etc. We present a Social Semantic Web prototype, PatientSupporter, that will enable such networking between patients within a virtual organization. PatientSupporter is an instantiation of Rule Responder that permits each patient to query other patients' profiles for finding or initiating a matching group. Rule Responder's External Agent (EA) is a Web-based patient-organization interface that passes queries to the Organizational Agent (OA). The OA represents the common knowledge of the virtual patient organization, delegating queries to relevant Personal Agents (PAs), and hands validated PA answers back to the EA. Each PA represents the medical subarea of primary interest to a corresponding patient group. The PA assists its patients by advertising their interest profiles employing rules about diagnoses and treatments as well as interaction constraints such as time, location, age range, gender, and number of participants. PAs can be distributed across different rule engines using different rule languages (e.g., Prolog and N3), where rules, queries, and answers are interchanged via translation to and from RuleML/XML. We discuss the implementation of PatientSupporter in a use case where the PA's medical subareas are defined through sports injuries structured by a partonomy of affected body parts.

8.3.1 Knowledge Base Signatures

Knowledge base signatures are available through the PatientSupporter website.
(http://ruleml.org/PatientSupporter/Signatures)

9 Supported Reasoning Engines and Languages

9.1 OO jDREW

OO jDREW, a deductive reasoning engine for the RuleML web rule language, is an Object Oriented extension to **jDREW**. OO jDREW implements Object Oriented extensions to RuleML which include:

- Order Sorted Types
- Slots
- Object Identifiers

OO jDREW is written in the Java programming language (Tested with Version 1.6.0).

9.1.1 POSL (Positional-Slotted Language)

A presentation, shorthand, and exchange syntax is described that integrates Prolog's positional and F-logic's slotted syntaxes for representing knowledge (facts and rules) in the Semantic Web. This positional-slotted (POSL) language accommodates various assertional-logical and object-centered modeling styles, yet strives for maximum conciseness and orthogonality. After an introduction, the document covers POSL selectors, unification, webizing, typing, and implementation. Webizing takes up and extends the use of URIs in N3. For humans, the POSL language is faster to write and easier to read than any XML syntax. However, since a parser and a generator map the POSL syntax to Object-Oriented RuleML and back, the machine advantages of XML can be preserved.

9.1.2 Running OO jDREW Within Eclipse

Running OO jDREW Within Eclipse

1) To obtain OOjDREW, go to the OOjDREW downloads page (http://www.jdrew.org/oojdrew/download.html) and download the source.
2) Import the source package into whatever IDE you prefer (I use eclipse)
3) To run OO jDREW, run:
   OO jDREW - Rule Responder->src/jdrew->jdrew.oo.gui->TopDownGUI.java

4) Place your taxonomy in the Type Definition window and load the type information. Make sure to select RDFS if the taxonomy is in RDF/RDFS.

5) Place the POSL knowledge base(s) in the Knowledge Base window and Parse Knowledge Base. Again, make sure to select POSL if your knowledge base is in POSL.

6) Type your query in the Query window and click Issue Query.

9.2 Euler

Euler is an inference engine supporting logic based proofs. It is a backward-chaining reasoner enhanced with Euler path detection. It has implementations in Java, C#, Python, Javascript and Prolog. Via N3 it is interoperable with W3C Cwm.

9.2.1 Notation 3 (N3)

Notation3, or N3 as it is more commonly known, is a shorthand non-XML serialization of Resource Description Framework models, designed with human-readability in mind: N3 is much more compact and readable than XML RDF notation. The format is being developed by Tim Berners-Lee and others from the Semantic Web community.

Notation 3 (N3) rule language:

- Standard N3 triple format:
  SUBJECT PROPERTY OBJECT

- Using prefix's:
  Prefixes can be used to abbreviate url's used in a given N3 file. For example:
  Instead of using:

  <example#Bob>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <example#Person>.

  The following prefixes can be implemented to shorten the statement:

  @prefix : <example#>.
  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
  :Bob rdf:type :Person.

- Representing variables:
  When using rules, variables can be used to test subjects against them. In N3, a variable is represented with a ? followed by characters used to represent it. For example:

  @prefix : <example#>.
  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
  {?S rdf:type :Swan} => {?S :color :White}.

  This means that all subjects who have a rdf:type of :Swan, then they also will have the :color property of object:White.

Triples written for Euler generally follow the format of axioms and queries, where the axioms contain relations and facts regarding the knowledge base, and the query is to retrieve the triples from it.

Axioms:

A standard axiom contains a "facts" and a "relations" file.

- Relations:
  The relations file lists all the relationships between the triples in the knowledge base (using inference). It can also be used to define any initial properties associated to subjects. All relations and initial values are written in the form of rules and have this form:

  {SUBJECT PROPERTY OBJECT} => {SUBJECT PROPERTY OBJECT}.

  For example, this relation will imply that if a certain individual C, has a father F, then F must have a child C:

  @prefix : <example#>.
  @prefix log: <http://www.w3.org/2000/10/swap/log#>.

{?C :father ?F} log:implies {?F :child ?C}

The ? symbol indicates a variable used in the formula. Therefore, ?C means every subject in the facts file is checked to see if they have a property of :father and if it has a value.

log:implies is equivalent to => in the N3 language. Therefore, => can be used in rules to represent an implication.

Facts:

A facts file could be thought of as creating an instance of a class such as in object-oriented programming. New subjects are created, which are generally given an rdf:type and other property values. These will be used with the query later to draw conclusions about these facts.

9.2.2 Running Euler (EYE) Within Eclipse:

1) Download YAP and install:
http://www.dcc.fc.up.pt/~vsc/Yap/current/yap-5.1.3-installer.exe (Windows)

2) Add the 'Yap\bin' folder to the Paths environment variable

3) Download WGet and install:
http://sourceforge.net/project/downloading.php?groupname=gnuwin32&filename=wget-1.11.4-1-setup.exe&use_mirror=voxel (Windows)

4) Download EulerSharp:
http://sourceforge.net/project/showfiles.php?group_id=77046

5) Copy the '\bin**Euler.jar**' file to a safe location on your hard drive.

6) Create a new project in eclipse.

7) Copy the Euler.jar file to BOTH the Java JRE and JDK external libraries. For Windows, they are typically under:
"C:\Program Files\Java\jre6\lib\ext"
and
"C:\Program Files\jdk1.6.x_yz\jre\lib\ext"

Alternatively:
You may also add the jar file as an "external JAR" under:
Project->Properties->Java Build Path->Libraries->Add External JARs

8) The command to execute the Euler EYE engine is as follows:

```
euler.Eye.runEye(args);
```

Where 'args' are the arguments (explained next). Typically you will want to use a print statement to view the results of the execution.

9.3 Prova

@@@To be completed@@@

9.3.1 Prova (Rule Language)

9.3.2 Running Prova in Eclipse

9.4 DR-Device

@@@To be completed@@@

9.4.1 DR-POSL

9.4.2 Running DR-Device in Eclipse

10 RuleML Query Generation

10.1 Preliminaries

10.1.1 Introduction

This documentation explains the RuleML Query Generation (RQG) GUI used in systems such as PatientSupporter and WellnessRules2 to create a wide variety of different RuleML queries. It is built upon HTML using JavaScript. Sections 2 details the several different sections of the JavaScript file *gui.js*. Section 3 details the HTML side of the RQG GUI. The document concludes with recommendations and some troubleshooting guidelines.

10.1.2 Files Involved

/files/gui.js (JavaScript file) – This contains all the JavaScript functions required for the RQG GUI. Comments in the file are used to split the file into 5 separate sections which are detailed below.

RuleResponder.html (HTML file) – This is the file that contains the HTML form elements referenced in the gui.js file. The RQG GUI itself is accessible by opening this file in a web browser. Information regarding this file can be found in section 3.

/files/datetimepicker_css.js (JavaScript file) – A CSS and JavaScript based calendar tool found at http://www.rainforestnet.com. How to use the calendar tool is not mentioned in this document; however the referenced website contains many examples and pieces of documentation in some of the code itself.

/files/layout.css – A Cascade Style Sheet (CSS) used with RuleResponder.html. The only thing of interest to the RQG GUI here is that during error checking, textboxes change their style template to show there is an error or change back when it is corrected. The styles used are noted with comments in this file.

10.2 JavaScript

The file *gui.js* is broken down into 5 different parts. Each sub-section below explains its corresponding part.

10.2.1 Preliminary Declarations and Functions

At the very start of *gui.js*, the global variables are declared. For each variable in the desired RuleML query (be it a free variable or bound constant), there are two JavaScript variables. One variable is for the RuleML query creation itself, and one is the English representation for the English description creation.

For instance, in WellnessRules2, for the ProfileID variable used in the RuleML query, we create the JavaScript variables: *profileID* and *englishProfileID*. *profileID* will eventually contain the section of the RuleML query that contains ProfileID information. *englishProfileID* will contain a string that represents the contents of ProfileID that could be used in a sentence. If the user is querying for any ProfileID (ProfileID is a free variable in the query), the corresponding values in this situation could be the following:

profileID = “<Var>ProfileID</Var>”

englishProfileID = “any profile”

The variable *messageHeader* contains a large string of all the static information which comes before the variables in the RuleML query. *messageFooter* contains all the static information that comes after the variables. Some of this static information is different in each use-case such as the id name which is just the use-case name.

Finally, any lower level functions required for the operation of the functions in the next section are placed in the preliminary section. In PatientSupporter and WellnessRules2, there are several functions used, each one described in commenting of *gui.js*. A few examples of these are:

checkTime() – Checks to make sure string matches time format.

pressSubmit() – Runs all functions required when submit button is pressed.

selectQuery() – Sets the dropdown menus to the corresponding example query values.

10.2.2 Variable Functions

In this section, each RuleML query variable has a function associated with it. The naming scheme in this section is: *elementSelected***NameOfVariable***()*. These functions are run when the *RuleResponder.html* page is opened or whenever the function’s form element is edited. Based on the user’s input in the variable’s corresponding HTML form element, the two variables (English description and RuleML variable) are given their proper values.

For example for the variable ProfileID, the function name is *elementSelectedProfileID()*. It uses a local variable (usually called *choice*) to store the value of the index number of the element selected in the html form element (which in this case is a drop down box). It then uses *padZeros()* from Section 2.1, to pad the index value to have the correct number of 0’s to be a profileID value. If the index value is equal to zero that implies “any” was selected in the html form. In RuleML, this would be a free variable so the global variable profileID stores the value

"<Var>ProfileID</Var>". Otherwise, profileID becomes a individual constant which would take the form "<Ind>p(**padded index number here)**</Ind>".

Form elements that require user text input have an additional error function that is invoked in the corresponding variable's function. This is described in Section 2.3. At the end of all variable functions, the *createRuleML()* function is called, which regenerates the RuleML query with the new values, and also calls the function *createEnglish()* to regenerate the English description as well. See section 2.4 for more details.

10.2.3 Error Checking

When dealing with HTML form elements that require text input from the user, the chance of user error goes up significantly. Currently the RQG GUI implements simple error checking that informs the user of their mistakes by simply changing the color of the text field that contains the error. When the error is removed, the text field returns to its original color.

Each form element that requires error checking has another function which is called right before *createRuleML()* in its corresponding variable function (see Section 2.2). This function's purpose is to check for possible errors. Some variables require the same error check function as other variables because they are related. Because of this, the naming convention of these error checking functions are *elementError<TopicName>Check().* An example of variables requiring the same error check function is the *startDate, startTime, endDate, endTime* variables. While these variables may have no standalone errors, together they may have an error. For example, if the user put *endDate* and *endTime* occurring before *startDate* and *startTime*.

10.2.4 Query and English Generation

In this section of the code, RuleML query and English generation is done. In *createRuleML()*, all the RuleML global variables (*messageHeader*, *profileID*…..*messageFooter*) are concatenated together as one large string and placed in the RuleML query box. *createEnglish()* is then called which combines all the English variables into a paragraph describing the query. The sentence structure is mostly up to the person creating the GUI.

10.2.5 JavaScript Starter

The *startCheck()* function runs when the web page is loaded (see the <body> tag in RuleResponder.html). It runs every form element function to generate a RuleML query. When sample query is chosen, *recompile()* is used (*startCheck()* would cause a loop in this situation).

10.3 HTML

As stated in Section 2, each form element function is tied to an HTML form element. That is how the values are generated and how updates to the RuleML query start. Each form element's type depends on the type of information that is required. Generally, the standard currently used is:

Selection (drop down boxes): This is used when there are a specific number of values for the variable. The corresponding form element function is triggered when a new selection is made.

Text box: This is used for inputting numerical values. The corresponding form element function linked to this type of form element is triggered whenever a key is pressed.

More information regarding various html form object can be found on the W3C site: http://www.w3.org/TR/html401/interact/forms.html

10.4 Recommendations

If you decide to modify the RQG GUI, it is suggested while you are making your modifications, you keep a very simple barebones html file containing only the required elements for the RQG GUI. This is to insure that while you're modifying the code, no bugs appear that are not related to the RQG GUI. After you are satisfied with the modfications, proceed to integrating it into the actual webpage that it will be accessible from.

The RQG GUI currently uses regular expressions for pattern matching in several places to avoid faulty values and does alert the user when an incorrect value is inputted (the box with the incorrect value turns red). However, further error checking could be used to prevent more kinds of errors in the queries. The large issue with error checking multiple values such as StartTime, StartDate, EndTime and EndDate, is that some of these may be free variables while others are not.

In the future, the ability for the RQG GUI to work "both ways" would prove very useful. This would mean the user could edit the RuleML manually and the user's changes would appear in the HTML form elements. This could be accomplished by having the Atom part of the query in a separate textbox from messageHeader and messageFooter, and then doing performing some form of parsing.

10.5 Troubleshooting

Blank or no result from Rule Responder

Examine the query in the address bar and ensure nothing extra is being added on or removed from the query. Some web browsers have built in address bar character limits (which does not meet the standard for HTML submission forms!). In Internet Explorer this dealt with by removing all unnecessary extra white space right before the query is sent.

Blank RuleML Query

This can be caused by any kind of coding mistake; however it was mostly found this occurred when a single string had a syntax error.

Undefined Values in RuleML Query

Most often it is found that the HTML form elements have been given the incorrect or duplicate name.

Debugging Tips

Using an IDE that supports JavaScript is very helpful for debugging.

Having a web browser equipped with a Java console is very useful for understanding script failures. Google Chrome is especially useful for this.

Using the JavaScript *alert("a message here")* is useful for debugging conditional statements.

11 PatientSupporter Profile Generator

11.1 Overview

A tool have has been developed for PatientSupporter for use with the Rule Responder Benchmarking Tool to test its performance regarding scalability. The Profile Generator tool generates *n* amount of random profiles for a use-case along with the corresponding responsibility matrix for the Personal Agents. Assuming you have a Rule Responder PatientSupporter system already installed and running, these files can be quickly inserted into your system for testing purposes. This tool is available on the Rule Responder SVN documented in section 2.1.3.

11.2 Example

A sample of output from this tool can be found here (http://ruleml.org/PatientSupporter/test/). 10 sample profiles and a responsibility matrix were generated in this instance.

11.3 Implementation

The RunMain() class uses the functions from ProfileCreation to make a number of POSL profiles specified by the variable NUMBER_OF_PROFILES. The file allProfiles.posl contains all the profiles, concatenated one after the other (for testing use in OO JDrew). legProfiles.posl is another special file containing only the profiles with leg injuries; it is for testing how fast Leg queries would be in OO JDrew without the rest of profiles in the knowledge base. More information regarding the implementation of this tool can be found within the comments of the source code.

12 Rule Responder Benchmarking Tool

12.1 Overview

The Rule Responder Benchmark and Testing tool can be used to test the response time of the various use cases of Rule Responder. Currently only benchmarking tools are automated, and provide the user with the ability to gather execution time data for specific instantiation queries of their choice. This tool is available on the Rule Responder SVN documented in section 2.1.3.

12.2 Example

Using the following Java statement, we can execute the “challenge chair contact” query from SymposiumPlanner2008 on 3 trials, and store the average running time across all trials:

```
Long runTime =
      Utilities.getTimeForSpecificQuery(
            Configuration.symposiumPlanner2008,
            "Challenge_Chair_Contact.txt",3);
```

Output:

```
=============================================
SymposiumPlanner2008 BENCHMARKING Challenge_Chair_Contact.txt ON 3 TRIALS
-----------------------------
Issuing query Challenge_Chair_Contact.txt, 3 times...
Execution Time for trial 1: 4807 milliseconds.
Execution Time for trial 2: 2851 milliseconds.
Execution Time for trial 3: 2613 milliseconds.
```

```
Average Time for 3 trials: 3423 milliseconds.
----------------------------
============================================
```

12.3 Implementation

There are two main packages in the benchmarking tool project "benchmarkUtilities" and "configuration".

Configuration is of course to configure the benchmarking tool and stores the names of the of the instantiations being used, as well as the server URL of the Rule Responder server being benchmarked/tested.

BenchmarkUtilities contains the actual implementation. Currently only the Utilities.java file is used and has methods available to get timing information from sets of queries stored in the project (i.e. **getTimeForSpecificQuery, getTimesForAllQueriesOfInst**). Both of these methods will use the **getQuery** method to build the query from those stored in the "queries" folder, as well as the **executeQuery** to send the built query off to the Rule Responder server defined in the configuration.

13 Troubleshooting Rule Responder Instantations

13.1 Installation

Problem:

Cannot open firewall ports.

Solution:

Firewall ports can only be opened on administrative accounts. You must switch to a user account with administrative privileges.

Problem:

Locally created EA does not function properly.

Solution:

- Ensure that the IP address has been change in your local version of the EA. Also insure that all files used in the EA are pointing to a local and existing file.

Problem:

After Maven installation, Eclipse gives the message: "`Eclipse is running in a JRE, but a JDK is required`".

Solution:

- Eclipse currently is not pointing to the JDK that was supposed to be installed. Consult Section 1.3 Prerequisites on how to correct this.

Problem:

Lost permission on Tomcat Webapps Folders

Solution:

- This is a known issue for some system setups. While the specifications of the problem are not clear at this time, the best solution to have the system admin restore you privileges to the corresponding files.

13.2 Sending Queries (High Level)

Problem:

I sent a query but my browser returned an error (e.g. "cannot connect to <IP>:8888")

Solution:

- Ensure that you started the Rule Responder service (see Section 1.5: Setup).
- Ensure that the EA is properly configured and pointed to the IP of where the service is running (see Section 1.1.2: Apache HTTP Server)

Problem:

I sent a query and received a RuleML answer with the relation "noPublicInterface"

Solution:

- There is no interface set up for the query you sent to the Prova Knowledge Base. Create a fact following the other interface templates which will satisfy your query.

Problem:

I sent a query and received only a blank, white screen.

Solution:

- This happened because the server timed out, meaning that there were no answers sent back within the allotted time limit (configured by ProvaUMOImpl.java). There are many background reasons for this, so please check your log files in mule.log and catalina.out, and troubleshoot them individually.

Problem:

I sent a query, Mule receives an error regarding Prova and the browser attempts to download a file called "download".

Solution:

- This happened because the EA interface being used was not properly downloaded. Redo Section 1.1.2.

13.3 Mule

Note: These problems are based on the output generated by the mule.log file

Problem:

Attempted to start Mule but received the error:

```
Exception in thread "main" java.lang.NoSuchMethodError:
org.mule.config.i19n.Message<init><Ljava/lang/
String;ILjava/lang/Object;>V
        …
```

Solution:

- An instance of Mule is already running. Shut it down and try starting again.

Problem:

Attempted to start Mule but received the error:

```
Exception in thread "main" java.lang.NoClassDefFoundError:
ws/prova/mule/impl/ESBManager
```

Solution:

- There is no "%RR- artifactId %-SNAPSHOT.jar" file inside the project "lib" folder. Compile one and copy it from "target" to "lib" (see 1 and 2 of Section 1.2.3: Starting Up Rule Responder)

Problem:

Mule successfully started and I sent a query, but nothing happened (i.e. no acknowledgement messages).

Solution:

- Make sure that you have configured your mule configuration file such that the descriptor (OA config) is enabled and working (see Section 2.0: Enterprise Service Bus (ESB) – Mule)
- If the descriptor is properly configured, check the Prova knowledge base which it points to. The knowledge base must have a rule which matches the query you sent, otherwise it will not be satisfied and no actions will be taken (see Section 3.1: Prova Knowledge Base)

Problem:

"Sent message to <PA> PA." shown, but followed by the error:

```
ERROR http.HttpConnector handleWorkException.1534 – Work caused
exception on 'workCompleted'.
…
```

Solution:

- Ensure that the Tomcat server is setup (see Section 1.1.4: Tomcat) and running (see Section 1.1.5: Startup for Windows and Section 1.2.3: Starting Up Rule Responder for Linux)
- The Mule configuration file may not be pointing to the active servlet, make sure that the endpoint that is returned by OWL is set up in the mule config file (see configuring mule of Section 1.1.3: Eclipse & Mule)
- Ensure that OWL returned the endpoint name that you intended (see Section 3.2: OWL Ontology)

Problem:

Stuck on "Looking up Responsible Personal Agent"

Solution:

- Make sure that you are pointing to the OWL file correctly in the Prova knowledge base you are using (see Section 3.1: Prova Knowledge Base)
- Make sure that the OWL ontology is properly configured (see Section 3.2: OWL Ontology)

Problem:

Stuck on "Sent message to <PA> PA."

Solution:

- The PA most likely received the message. Look into issues with either reasoning inside the PA, or how sending the message back is configured (see Section 4.0: Personal Agents)

13.4 Personal Agents

Problem:

Personal Agents (PAs) send a response but the Organizational Agent never receives it.

Solution:

- The pa_Configuration source used for that particular OA likely has the wrong IP address. Be sure pa_Configuration in the Rule Responder project is compiled with the proper IP and then copy the files back into the PA's servlet. Additionally, insure that config.xml for that specific web servlet contains the correct IP as well (see Section 1.4 Tomcat for more details).

13.5 Issues and Workarounds

Fixed (June, 2010):

- Prova substitutions are currently hard-coded in the knowledge base, and would in theory require hundreds or thousands of rules.
  - Reason:
    In Prova, there is currently no way to handle any combination of variables (<Var>) and constants (<Ind>), so rules must be created for "expected" inputs as a workaround. Currently working with Adrian in an attempt to resolve.

  - Location:
    %RR-dir%->rules->use_caseWellnessRules->Wellness-Rules->WellnessRules-Responder.prova
    Rule:
    processMessage(XID,From,Primitive,myActivity(ProfileID,Activity,InOut,MinRSVP,MaxRSVP,StartTime,EndTime,Location,Duration,FitnessLevel)):-

  - **Solution:** Modified MessageGenerator to include a "messages()" method which sends the verbatim RuleML message containing only the Ind's and Var's of the results. These are stored in new variables, which are constructuted back into the original relation before being sent back to the EA.
    (see WellnessRules2/PatientSupporter PAs for Message sending change)
    (see
    rules->usecaseWellnessRules2->Wellness-Rules2->WellnessRules2-Responder.prova
    &
    rules->usecasePatientSupporter->PatientSupporter->PatientSupporter-Responder.prova

    for changes in the Prova knowledge base.

Fixed (February, 2010):

- Rule Responder now creates the required query predicates to simulate the input of Min and Max RSVP.
  - Reason:
    Because N3 is bottom-up, nothing can be passed in like OO jDREW. So RR must build the query to inject predicates containing the provided numbers.

  - Location:
    %RR-dir%->personalAgents->pa_WellnessRules2->N3Message.java
    Method:
    private String parseRuleMLQuery() {}

  - **Solution:** Modified "personalAgents->pa_WellnessRules2->N3Handler.java->parseRuleMLQuery()" so that it now constructs a query such that ?minRSVP and ?maxRSVP use math:notLessThan and math:notGreaterThan to calculate whether or not the proposed min and max RSVP's are within acceptable limits. It does so by first querying for myActivity with ANY min/maxRSVP and then performing this check; if it succeeds if performs a "select", removing these constraints and just displaying the query result.

- POSL requires input of MinRSVP and MaxRSVP, while N3 must have variable input. Therefore, the RuleML query uses constants, and N3 treats them as variables.
  - Reason:
    WellnessRules must fill the MinRSVP and MaxRSVP values in order to pass the premise in the myActivity rule. POSL does so by accepting the values as input from the query because it is top-down.

  - **Location (of N3 workaround):**
    oojdrew->src->default->N3Handler.java

Method:
public static Vector<String> parseRuleMLQuery(DefiniteClause ruleMLQuery) {}

- **Solution:** Modified "personalAgents->pa_WellnessRules2->N3Handler.java->parseRuleMLQuery()" so that it now constructs a query such that ?minRSVP and ?maxRSVP use math:notLessThan and math:notGreaterThan to calculate whether or not the proposed min and max RSVP's are within acceptable limits. It does so by first querying for myActivity with ANY min/maxRSVP and then performing this check; if it succeeds if performs a "select", removing these constraints and just displaying the query result.

Fixed (December 16th, 2009):

- ProfileID must be the name of the variable profiles (to determine profileSpecified).
  - Reason:
    The servlet must determine, from the message, which profile is specified. If it finds "<Var>ProfileID</Var>" then the profile specified is "All". The alternative is to assume that the ProfileID will be the first given variable following "<Rel>myActivity</Rel>"

  - Location:
    oojdrew->src->default->GeneralHandler.java
    Method:
    public static String getProfileSpecified(String message) {}

- ProfileID no longer needs to start with a 'p'

OO jDREW - Workarounds and Not Yet Working:

Workarounds:

- Anonymous typed variables caused a warning statement for every possible combination. This caused a slowdown by a factor of greater than 15x. The warning print statement for the debugger has been removed.
  - Reason:
    The usage of anonymous typed variables caused a significant slowdown in execution time.
  - Location:
    oojdrew->jdrew.oo.td->Unifier.java
    Method:
    private boolean unify(Term term1, Term term2, int varBindType) {}

General Rule Responder - Workarounds and Not Yet Working:

Workarounds:

- The RuleML to Prova XSLT translation used a '.' Instead of ':' for types. This resulted in Prova throwing an error during typed variable translation. Therefore, the XSLT was changed to output ':' instead of '.' In between a type and its variable.
  - Reason:
    The XSLT translation was causing Prova to throw errors.
  - Location:
    %RR-dir%->src->main->resources->rml2prova.xsl

- Typed variables are not recognized during ProvaToRuleML serialization. A special case was added which checks for '<' and ':' in the String, which should indicate it as a typed variable.
  - Reason:
    Typed variables are just assumed to be constants and so are not parsed correctly.

Location:
%RR-dir%->src/main/java->ws.prova.mule.impl->Prova2RuleMLTranslator.java
Method:
private String serializeConstantTerm(Object ct, String spacer) {}
private Object doTransform(Object src, String encoding) {}

- RuleML inputs with more than one colon are assumed to not be types, and instead treated as literals.
  - Reason:
    Using a literal such as '2009-06-15T10:15:00' results in the following translation:
    <Ind type=15:00>2009-06-15T10</Ind>
    When the following is expected:
    <Ind>2009-06-15T10:15:00</Ind>

  - Location:
    %RR-dir%->src/main/java->ws.prova.mule.impl->Prova2RuleMLTranslator.java
    Method:
    private String serializeConstantTerm(Object ct, String spacer) {}

- Windows Vista installations of Rule Responder can sometimes have communication issues between the OA and the PAs due to Vista security permission issues. Generally, the OA's message is never received by the PA. Not many details are known on the exact specifications of this issue; however current work on the Rule Responder may lead to a fix for this.

SymposiumPlanner - Workarounds and Not Yet Working:

Workarounds:

- Variables are excluded from complex terms.
  - Reason:
    Complex terms had variable names included, where non-complex terms do not. To be consistent this was removed.
  - Location:
    %RR-dir%->src/main/java->
    ws.prova.mule.impl->Prova2RuleMLTranslator.java
    Method:
    private String serializeComplexTerm(ComplexTerm ct, String spacer) {}

WellnessRules2 - Workarounds and Not Yet Working:

Workarounds:

- If a variable substitution is returned by OO jDREW or Euler it is ignored. This is to implement typed variables.
  - Reason:
    In Prova, the substitutions performative expects Ind's only, so Var's will not work. Therefore, they are simply not included in the answer.
  - Location:
    %RR-dir%->personalAgents->pa_WellnessRules2->MessageGenerator.java

Method:
public String[] Messages2() {}

WellnessRules & WellnessRules2 - Workarounds and Not Yet Working:

Workarounds:

- All query-able facts must be hard-coded into the PA servlet, in order to properly translate RuleML to N3 query.
  - Reason:
    No slot information is given with the RuleML query (only positional). Therefore, the slot names are coded directly in the servlet, corresponding to the provided relation name.

  - Location:
    oojdrew->src->default->N3Handler.java
    Method:
    public static Vector<String> parseRuleMLQuery(DefiniteClause ruleMLQuery) {}

- Even for N3 queries, the position of variables matters, and must correspond to the Signatures.pdf.
  - Reason:
    Similarly due to the previously mentioned hard-coding, and because slot names are required for N3 but are not provided as input, the input is assumed to be in correct order.

  - Location:
    oojdrew->src->default->N3Handler.java
    Method:
    public static Vector<String> parseRuleMLQuery(DefiniteClause ruleMLQuery) {}

- ProfileID must correspond to the exact filename of the profile knowledge base.
  - Reason:
    The servlet must determine, from the message, which profile is specified. If it finds “<Var>ProfileID</Var>” then the profile specified is “All”. The alternative is to assume that the ProfileID will be the first given variable following “<Rel>myActivity</Rel>”

  - Location:
    oojdrew->src->default->GeneralHandler.java
    Method:
    public static String getProfileSpecified(String message) {}

WellnessRules 1 - Workarounds and Not Yet Working:

Workarounds:

- For N3 query parsing, type values are currently ignored. Types may be supported later with rdf:type.
  - Reason:
    N3 does not require type declaration for integers or literals.

  - Location:
    oojdrew->src->default->N3Handler.java
    Method:
    public static Vector<String> parseRuleMLQuery(DefiniteClause ruleMLQuery) {}

- String literals for date-time are assumed, meaning that N3 query parsing automatically places quotes around constants that have greater than 1 colon. Also, for a still unknown reason, POSL automatically places quotes around these date-time constants, so no workaround needed for that.
  - Reason:
    RuleML does not allow its input to contain quotes, but the WR knowledge base requires quotes around date-time values.

  - Location:
    oojdrew->src->default->N3Handler.java
    Method:
    public static Vector<String> parseRuleMLQuery(DefiniteClause ruleMLQuery) {}

14 Appendix: Migrating Mule 1.x to 3.0

The appendix of how to migrate Mule 1.x to 3.0 is divided into two parts. We firstly describe the changes during migrating Mule 1.x to 3.0 which are involved in our RuleResponder System. In Section 15.2 we describe how to migrate the RuleResponder System to Mule 3.0 in detail.

14.1 Migrating Mule 1.x to 3.0 Principles

This section describes the major configuration and code changes in Mule 3.0 to help Mule 1.x users understand what have been changed between two releases. (The following description is based on the migration specifications from Mule ESB Community: http://www.mulesoft.org/)

14.1.1 XML Configuration Changes

Following are some high-level changes in the XML configuration. For details, see Configuration Changes below.

1.x	3.0
Interceptors	supported starting from Mule 2.1.0
Mule implementations are attributes	Mule implementations are elements
Mule descriptor implementation is explicitly required	Service component configuration is optional. PassThroughComponent is default.
Endpoint: address or name attribute	Endpoint: address, name, or new path attribute

14.1.2 Code Changes

1.x	3.0
org.mule.umo.*	org.mule.api.* in most cases
org.mule.providers.*	org.mule.transport.*.
org.mule.extras.*	org.mule.module.NAME.*
package names are singular/plural	package names are singular
MuleClient.receive(...)	MuleClient.request(...)
MuleManager utility class	MuleContext instance available through life cycle methods or MuleAwareContext implementation.
QuickConfigurationBuilder	No longer supported
MuleXMLConfigurationBuilder	DefaultMuleContextFactory
UMOMessage	MuleMessage
MuleMessage	DefaultMuleMessage
UMOEndpoint	InboundEndpoint or OutboundEndpoint
UMOEndpointURI	EndpointURI
MuleEndpoint.getOrCreateEndpointForUri(...,	new

UMOEndpoint.ENDPOINT_TYPE_SENDER)	DefaultEndpointFactory().getOutboundEndpoint(...)
MuleEndpoint.getOrCreateEndpointForUri(..., UMOEndpoint.ENDPOINT_TYPE_RECEIVER)	new DefaultEndpointFactory().getInboundEndpoint(...)
doPostFunctionalSetUp()	suitePreSetUp() and doSetUp()

14.1.3 Configuration Changes

File Header

Mule configuration files from are now based on an XML schema instead of DTDs. Therefore, you must configure the Mule core namespace plus any modules or transports as shown below in this empty configuration file.

Mule 1.4

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mule-configuration PUBLIC "-//MuleSource //DTD mule-configuration XML V1.0//EN"
                    "http://mule.mulesource.org/dtds/mule-configuration.dtd">
<mule-configuration id="Samples" version="1.0">
…
</mule-configuration>
```

Mule 3.0

```
<?xml version="1.0" encoding="UTF-8"?>
<mule
xmlns=http://www.mulesoft.org/schema/mule/core
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:spring="http://www.springframework.org/schema/beans"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:jms=http://www.mulesoft.org/schema/mule/jms  xmlns:http="http://www.mulesoft.org/schema/mule/http"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-2.5.xsd
    http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/mule/http/3.0/mule-http.xsd
    http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
    http://www.mulesoft.org/schema/mule/jms http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd
    http://www.mulesoft.org/schema/mule/xml http://www.mulesoft.org/schema/mule/xml/3.0/mule-xml.xsd
>
…
</mule>
```

Note: This page describes the recommended approach of using mule as the default namespace, but you can also use other default namespaces.

Environment Properties

Mule 3.0 now uses the <configuration/> element for configuring global settings, such as thread profiles and the default endpoint setting for synchronous/asynchronous messaging. For information on using this element, see Global Settings Configuration Reference.

You now set the queue profile on the model and set the pooling profile directly on the pooled component. For more information on threading profiles, queue profiles, and pooling, see Tuning Performance.

Connectors

Just as with Mule 1.x, connectors are configured as top-level elements in the Mule configuration, but they now use transport-specific schemas. The connector elements have changed accordingly as shown below:

Mule 1.4

```
<connector className="org.mule.providers.vm.VMConnector"/>
<connector className="org.mule.providers.jms.JmsConnector"/>
```

Mule 3.0

```
<vm:connector/>
<jms: activemq-connector/>
```

The new approach provides a huge advantage in that each transport now defines its own syntax for configuring connectors, avoiding class names and untyped properties. Additionally, you can use your favorite IDE's auto-completion feature, making Mule3.0 fast and easy to configure.

Following is another example illustrating a jms connector:

Mule 1.4

```
<connector name="jmsConnector"
                    className="org.mule.providers.jms.JmsConnector">
                    <properties>
                              <property name="connectionFactoryJndiName"
                                        value="ConnectionFactory" />
                              <property name="jndiInitialFactory"
                                        value="org.activemq.jndi.ActiveMQInitialContextFactory" />
                              <property name="specification" value="1.1" />
                              <map name="connectionFactoryProperties">
                                        <property name="useEmbeddedBroker" value="true" />
                              </map>
                    </properties>
          </connector>
```

Mule 3.0

```
<jms:activemq-connector name="jmsConnector" specification="1.1" brokerURL=" vm://localhost"/>
```

Note: To use a specific transport, you must import its namespace as explained above.

Endpoints

The @remoteSync attributes on endpoints has been removed. This also removes the confusion around how endpoint synchronicity is configured and behaves. From Mule 2.1 the synchronicity of an endpoint had to be specifically defined on every endpoint (otherwise the default for the Mule instance is used). Now the following rules apply when configuring endpoints:

* If the endpoint is asynchronous a result is never returned

* If the endpoint is transacted a response cannot be read from a back channel (i.e. JMSReplyTo)

* If the endpoint is synchronous, it will always return a result even if the result is null, a NullPayload will be returned. This behaviour tries to open a back channel to receive a result, this is the equivalent to @remoteSync=true. The new @responseTimeout configuration option on the endpoint allows Mule to timeout if nothing has been received via a back channel.

Endpoints are now easier to configure and more context-specific in Mule 3.0, as described below.

i. Global Endpoints

Global endpoints act as endpoint templates, allowing you to configure and reuse an endpoint configuration for multiple logical endpoints. To configure a global endpoint, you use the "endpoint" element as a top-level element instead of on routers or exception strategies. You can configure all the same elements on a global endpoint as on a logical endpoint, including making them transport-specific.

An <endpoint-identifier> in 1.4 is equivalent to a global endpoint in 3.0 that just has a URI configured.

Notes:

* Be careful defining transformers on global endpoints that are referenced by inbound and outbound endpoints, as transformers are not the same for inbound and outbound endpoints.

* Although logical endpoints inherit and can extend a global endpoint configuration, you cannot extend properties. Properties configured on a logical endpoint will overwrite properties configured on the referenced global endpoint.

ii. Explicit Endpoint Types

You now specify endpoint types explicitly (inbound, outbound, or response). This approach allows endpoints to have more specific configuration. For example, by specifying an endpoint as inbound, you could ensure that pollingFrequency is configured on the inbound file connector but not on its outbound counterpart.

iii. Transport-specific Endpoints

Endpoint configuration has been improved to avoid untyped properties. Some of these properties are cross-transport attributes or elements, whereas others are defined by transport-specific endpoint definitions.

Mule 1.4

```
<endpoint address="pop3://bob:secret@localhost:62002" transformers="BytesToMime"/>
```

Mule 3.0

```
<pop3:inbound-endpoint user="bob" password="secret" host="localhost" port="62002">
   <email:bytes-to-mime-transformer/>
</pop3:inbound-endpoint>
```

iv. Services and Components (Formerly MuleDescriptors)

Services and components are now configured differently. Following are the Mule Hello example in different ways of configuration:

Mule 1.4

```
<mule-descriptor name="ChitChatUMO" implementation="org.mule.samples.hello.ChitChatter">
   <inbound-router>
      <endpoint address="vm://chitchatter" transformers="NameStringToChatString"/>
   </inbound-router>
   <outbound-router>
      <router className="org.mule.routing.outbound.OutboundPassThroughRouter">
         <endpoint address="stream://System.out" transformers="ChatStringToString" />
```

```
      </router>
   </outbound-router>
</mule-descriptor>
```

Mule 3.0

```
<service name="ChitChatUMO">
   <inbound>
      <vm:inbound-endpoint path="chitchatter" transformer-refs="NameStringToChatString"/>
   </inbound>
   <component class="org.mule.samples.hello.ChitChatter"/>
   <outbound>
      < pass-through-router>
         <stdio:outbound-endpoint system="OUT" transformer-refs="ChatStringToString"/>
      </ pass-through-router>
   </outbound>
</service>
```

Following is a description of the changes:

(1) New Nomenclature

MuleDescriptor is now Service

Implementation is now Component

(2) Extensibility

A component is abstract, allowing multiple implementation types. By default, Mule includes two implementations: <component/> and <pooled-component/>. Other modules can add other component types.

The <component> and <pooled-component> elements are configured in exactly the same way, except that the <pooled-component> has an additional optional <pooling-profile> child element that can be use to customize the pool behavior. These elements can either use the "class" attribute or define an object-factory as a child element. If you use the "class" attribute, the prototype object factory will be used by default. You can configure components with singleton, prototype, and Spring object factories.

(3) Minimal Configuration

```
<component class="org.your.PrototypeComponent"/>
<pooled-component class="org.your.PooledPrototypeComponent"/>
```

(4) Complete Configuration Example:

```
<spring:bean name="myPooledSpringBeanComponent" class="org.your.PooledSpringBeanComponent" />
..
<pooled-component>
   <reflection-entry-point-resolver/>
   <spring-object bean="myPooledSpringBeanComponent"/>
   <custom-lifecycle-adapter-factory class="org.mule.config.spring.parsers.specific.TestLifecycleAdapterFactory"/>
   <binding interface="java.lang.String" method="setMethod">
      <outbound-endpoint address="vm://myEndpoint" />
   </binding>
   <pooling-profile exhaustedAction="WHEN_EXHAUSTED_FAIL"
      initialisationPolicy="INITIALISE_ALL" maxActive="1"
      maxIdle="2" maxWait="3" />
</pooled-component>
```

Routers and Filters

Routers and filters also take advantage of the new schema-based configuration. Most are defined in Mule, but specific modules and transports can also contribute routers or filters by defining them in their own namespace.

Mule 1.4

```
<router className="org.mule.routing.inbound.SelectiveConsumer"/>
<router className="org.mule.routing.outbound.FilteringOutboundRouter"/>
<router className="org.mule.routing.outbound.MulticastingRouter"/>
<router className="org.my.CustomRouter"/>
```

Mule 3.0

```
<selective-consumer-router/>
<filtering-router/>
<multicasting-router/>
<custom-inbound-router class="org.my.CustomRouter"/>
```

Mule 1.4

```
<global-endpoints>
  <endpoint name="CustomerResponses" address="vm://customer.responses"/>
</global-endpoints>
..
<router className="org.mule.routing.outbound.FilteringOutboundRouter">
   <global-endpoint name="CustomerResponses"/>
<filter expectedType="org.mule.examples.loanbroker.messages.LoanQuote"
 className="org.mule.routing.filters.PayloadTypeFilter"/>
</router>
```

Mule 3.0

```
<vm:endpoint name="CustomerResponses" path="customer.responses"/>
..
<filtering-router>
   <outbound-endpoint ref="CustomerResponses"/>
   <payload-type-filter expectedType="org.mule.examples.loanbroker.messages.LoanQuote"/>
</filtering-router>
```

Transformers

Transformers also take advantage of the new schema-based configuration. Modules and transports contribute transformers by defining them in their own namespace.

Transformers are now defined directly rather than by specifying class names.

Mule 1.4

```
<transformer className="org.mule.transformers.simple.MessagePropertiesTransformer"/>
<transformer className="org.mule.transformers.simple.ByteArrayToObject"/>
<transformer className="org.mule.transformers.xml.XsltTransformer"/>
<transformer className="org.my.CustomTransformer"/>
```

Mule 3.0

```
<message-properties-transformer/>
<byte-array-to-object-transformer/>
<xml:xslt-transformer/>
<custom-transformer class="org.my.CustomTransformer"/>
```

Custom transformers can still be used with the "custom-transformer" element.

Transformers can be referenced from endpoints using the "transformer-refs" attribute or can be declared inline.

Mule 1.4

```
<global-endpoint name="CustomerRequestsREST" transformers="RestRequestToCustomerRequestTransformer"/>
```

Mule 3.0

```
<inbound-endpoint ref="CustomerRequestsREST" transformer-refs="Transformer1 Transformer2"/>
<inbound-endpoint ref="CustomerRequestsREST">
   <transformer ref="RestRequestToCustomerRequestTransformer" />
</inbound-endpoint>
<inbound-endpoint ref="IncomingData">
   <byte-array-to-object-transformer/>
</inbound-endpoint>
```

Bridging

Bridging configuration has been simplified in Mule 3.0. To implement a bridge service, you simply configure inbound and outbound routers. Bridging occurs implicitly.

Mule 1.4

```
<mule-descriptor name="bridge" implementation="org.mule.components.simple.BridgeComponent">
   <inbound-router>
      <endpoint address="vm://bridge.inbound"/>
   </inbound-router>
   <outbound-router>
      <router className="org.mule.routing.outbound.OutboundPassThroughRouter">
         <endpoint address="vm://bridge.outbound"/>
      </router>
   </outbound-router>
</mule-descriptor>
```

Mule 3.0

```
<service name="bridge">
  <inbound>
     <vm:inbound-endpoint path="bridge.inbound"/>
  </inbound>
  <outbound>
     <pass-through-router>
        <vm:outbound-endpoint path="bridge.outbound"/>
     </ pass-through-router>
  </outbound>
</service>
```

The "bridge-component" and "pass-through-component" can still be used for backward-compatibility but are no longer needed.

Exception Strategies

Developers now have much finer control over transactions through configuration. Pattern matching filters can be used to match different types of exceptions. For example:

```
<model>
   <default-service-exception-strategy>
      <commit-transaction exception-pattern="*">
      <jms:outbound-endpoint queue="DLQ"/>
   </default-service-exception-strategy>
   ...
</model>
```

This configuration tells Mule to keep any current transaction open until after we dispatch to the JMS DLQ (Dead Letter Queue), and then commit the current transaction.

For certain transactions, you may want to roll back the transaction immediately. For example:

```
<model>
   <default-service-exception-strategy>
      <commit-transaction exception-pattern="*">
      <rollback-transaction exception-pattern"com.acme.a.*,com.acme.b.*"/>
      <jms:outbound-endpoint queue="DLQ"/>
   </default-service-exception-strategy>
   ...
</model>
```

This configuration tells Mule to roll back transactions when there are exceptions with packages com.acme.a and com.acme.b. Otherwise, commit the current transaction. Note that the <rollback-transaction> has priority over the <commit-transaction> element.

14.2 Migrating Mule 1.x to 3.0 for RuleResponder

This section describes the details of how to migrate the RuleResponder System to the Mule 3.0. The latest instantiation SymposiumPlanner-2011 employs Mule 3.1.2 and Prova 3.1.3, which is available at: http://ruleml.org/SymposiumPlanner/

14.2.1 ESBManager (ws.prova.mule.impl. ESBManager)

The Class “org.mule.MuleManager” has been replaced by “org.mule.api.MuleContext” which is available through life cycle methods or MuleAwareContext implementation. Therefore, to the methods of setup and tearing down the RuleResponder have been changed accordingly as shown below:

RuleResponder with Mule 1.X

```
        public static void setUp() {
    MuleXmlConfigurationBuilder builder;
                try {
                        builder = new MuleXmlConfigurationBuilder();
                        builder.configure(config);
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        public static void tearDown() {
                MuleManager.getInstance().dispose();
        }
```

RuleResponder with Mule 3.0

```
public static MuleContext context = null;
```

```
public static void setUp() {
        try {
                context = new DefaultMuleContextFactory().createMuleContext
                (new SpringXmlConfigurationBuilder(config));
                context.start();
        } catch (InitialisationException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        } catch (ConfigurationException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        } catch (MuleException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }

}

public static void tearDown() {
        try {
                context.stop();
        } catch (MuleException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}
```

14.2.2 ProvaUMOImpl (ws.prova.mule.impl. ProvaUMOImpl)

The MuleDescriptor has been changed to Service from Mule 2.0. Therefore, the Class ws.prova.mule.impl. ProvaUMOImpl has to implement the Interface "org.mule.api.service.ServiceAware" to initialize the Service instead of "org.mule.impl.UMODescriptorAware" in Mule 1.x. The changes are shown as follows.

RuleResponder with Mule 1.X

```
public class ProvaUMOImpl extends LogComponent implements ProvaUMO, Initialisable, Callable, UMODescriptorAware {
private UMODescriptor descriptor;
…
}
```

RuleResponder with Mule 3.0

```
public class ProvaUMOImpl extends LogComponent implements ProvaUMO,
        Initialisable, Callable, ServiceAware {
private Service service;
…
}
```

Accordingly, the initialization of Prova instance is changed. Firstly, we have to get the property of "rulebase" and its agent name. The agent name is used to find the destination of Prova agent's endpoints. In the RuleResponder with Mule 1.X, MuleDescriptor and its endpoint have the same name. However, in Mule 3.0, the name of the Service and its endpoint should be different. Therefore, we use the endipoint name instead of the service name to express the agent name. Meanwhile, the properties of Prova agent only can be configured as a sub-element of "inbound-endpoint". The changes are shown as follows:

RuleResponder with Mule 1.X

```
String rulebase = null;
        try {
                rulebase = (String) descriptor.getProperties().get("rulebase");
        } catch (Exception e) {
                logger.error("rulebase property not set");
                logger.error(e);
        }
        agentName = descriptor.getName();
```

RuleResponder with Mule 3.0

```
String rulebase = "";
        ServiceCompositeMessageSource source = (ServiceCompositeMessageSource) service
                        .getMessageSource();
        Iterator iter = source.getEndpoints().iterator();
        while (iter.hasNext()) {
                ImmutableEndpoint endpoint = (ImmutableEndpoint) iter.next();
                if (endpoint.getProperties() != null) {
                        if (endpoint.getProperties().containsKey("rulebase")) {
                                rulebase = (String) endpoint.getProperties()
                                                .get("rulebase");
                                agentName = endpoint.getName();
                                break;
                        }
                }
        }
```

Note: From Mule 2.0, the users have to specify endpoint types explicitly (inbound, outbound, or response). Therefore, in order to get the properties of Prova agent, we need to get its Messages Source first. Then we iterate all the inbound-endpoint to find the "rulebase" Property. Usually, there is only one property named "rulebase".

Meanwhile, from the Mule 2.0, the UMOMessage has been changed to MuleMessage, and the UMOEventContext has been changed to MuleEventContext. Therefore, the process of an inbound Prova RMessage needs to be changed as follows:

RuleResponder with Mule 1.X

```
public Object onCall(UMOEventContext context) throws Exception {
    UMOMessage inbound = context.getMessage();
…
}
```

RuleResponder with Mule 3.0

```
public Object onCall(MuleEventContext context) throws Exception {
        MuleMessage inbound = context.getMessage();
…
}
```

The registration of temporary synchronous UMO agent is also changed after the "org.mule.MuleManager" is replaced by "org.mule.api.MuleContext" and the <endpoint-identifier> in 1.4 is replaced by the global endpoint. The changes are shown as follows:

RuleResponder with Mule 1.X

```
UMOManager mgr = MuleManager.getInstance();
                                        try {
                                                mgr.registerEndpointIdentifier(tmpAgent, "jms://topic:"+tmpAgent);

                                        } catch (Exception e) {
                                                logger.error("Can not register synchronous UMO for "+tmpAgent+". ");
                                                logger.error(e);
                                                context.setStopFurtherProcessing(true);
                                                return e.toString()+" This user is already registered. Please use another user
name as sender address."; // return exception
                                        }
}
```

RuleResponder with Mule 3.0

```
try {
                                MuleRegistry helper = service.getMuleContext().getRegistry();
                                        EndpointBuilder builder = new DefaultTransportServiceDescriptor(
                                                        tmpAgent, new Properties(), null)
                                                        .createEndpointBuilder("jms://topic:" + tmpAgent);
                                        helper.registerEndpointBuilder(tmpAgent, builder);
                                } catch (Exception e) {
                                        e.printStackTrace();
                                        context.setStopFurtherProcessing(true);
                                        return e.toString()+ " This user is already registered. Please use another user
name as sender address."; // return
                                }
}
```

By giving the current MuleContext as described in the previous section, an org.mule.api.client.MuleClient instance got from the Service is created. The change isshown as follows:

RuleResponder with Mule 1.X

```
MuleClient client = new MuleClient();
```

RuleResponder with Mule 3.0

```
MuleClient client = new DefaultLocalMuleClient(service.getMuleContext());
```

14.2.3 mule-all-config.xml

Based on the description mentioned in the Section 15.1.3, we need to accordingly change the File Header, Environment Properpties Connectors, Global Endpoints and Services and etc. in “mule-all-config.xml”. The details are shown as follows:

File Header

RuleResponder with Mule 1.X

```
<!DOCTYPE mule-configuration PUBLIC "-//MuleSource //DTD mule-configuration XML V1.0//EN"
                        "http://mule.mulesource.org/dtds/mule-configuration.dtd">
```

RuleResponder with Mule 3.0

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:spring="http://www.springframework.org/schema/beans"
        xmlns:context="http://www.springframework.org/schema/context"
```

```
        xmlns:jms="http://www.mulesoft.org/schema/mule/jms" xmlns:http="http://www.mulesoft.org/schema/mule/http"
        xmlns:xm="http://www.mulesoft.org/schema/mule/xml" xmlns:stdio="http://www.mulesoft.org/schema/mule/stdio"
        xsi:schemaLocation="
    http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-2.5.xsd
    http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/mule/http/3.0/mule-http.xsd
    http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
    http://www.mulesoft.org/schema/mule/jms http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd
    http://www.mulesoft.org/schema/mule/xml http://www.mulesoft.org/schema/mule/xml/3.0/mule-xml.xsd
    http://www.mulesoft.org/schema/mule/stdio http://www.mulesoft.org/schema/mule/stdio/3.0/mule-stdio.xsd">
```

Environment Properpties

RuleResponder with Mule 1.X

```
<environment-properties>
        <property name="MULE_IMPLEMENTATION" value="ws.prova.mule.impl.ProvaUMOImpl"/>
        <property name="TOMCAT" value="http://10.1.3.35:8080/"/>
        <property name="MULE_PORT" value="8888"/>
…
</ environment-properties >
```

RuleResponder with Mule 3.0

```
        <global-property name="MULE_PORT" value="8888"/>
        <global-property name="MULE_IMPLEMENTATION" value="ws.prova.mule.impl.ProvaUMOImpl"/>
        <global-property name="TOMCAT" value="http://127.0.0.1:8080"/>
```

Note: The element “environment-properties” has been removed in Mule 3.0.

Connectors

RuleResponder with Mule 1.X

```
<connector name="jmsConnector"
                    className="org.mule.providers.jms.JmsConnector">
                    <properties>
                            <property name="connectionFactoryJndiName"
                                    value="ConnectionFactory" />
                            <property name="jndiInitialFactory"
                                    value="org.activemq.jndi.ActiveMQInitialContextFactory" />
                            <property name="specification" value="1.1" />
                            <map name="connectionFactoryProperties">
                                    <property name="useEmbeddedBroker" value="true" />
                            </map>
                    </properties>
        </connector>
```

RuleResponder with Mule 3.0

```
<global-property name="http.host" value="localhost"/>
<global-property name="jms.url" value="vm://localhost"/>
<jms:activemq-connector name="jmsConnector"
                    specification="1.1" brokerURL="${jms.url}"/>
```

Note: The element “mule-environment-properties” has been removed in Mule 3.0.

Global Properties

RuleResponder with Mule 1.X

```
        <property name="PatientSupporterOAName" value="PatientSupporter"/>
        <property name="PatientSupporterTopicName" value="patientSupporter"/>
        <property name="PatientSupporter_PORT" value="9994"/>
        <property name="PatientSupporter_Prova" value="/use_casePatientSupporter/Patient-Supporter/PatientSupporter-
Responder.prova"/>
        <property name="PatientSupporterTopic1" value="Leg"/>
        <property name="PatientSupporterTopic2" value="Torso"/>
        <property name="PatientSupporterTopic3" value="Neck"/>
        <property name="PatientSupporterTopic4" value="Shoulder"/>
        <property name="PatientSupporterTopic5" value="Head"/>
        <property name="PatientSupporterTopic6" value="Back"/>
        <property name="PatientSupporterTopic7" value="Arm"/>
…
```

RuleResponder with Mule 3.0

```
<global-property name="PatientSupporterOAName" value="PatientSupporter"/>
        <global-property name="PatientSupporterTopicName"
                value="patientSupporter"/>
        <global-property name="PatientSupporter_PORT" value="9994"/>
        <global-property name="PatientSupporter_Prova"
                value="/use_casePatientSupporter/Patient-Supporter/PatientSupporter-Responder.prova"/>
        <global-property name="PatientSupporterTopic1" value="Leg"/>
        <global-property name="PatientSupporterTopic2" value="Torso"/>
        <global-property name="PatientSupporterTopic3" value="Neck"/>
        <global-property name="PatientSupporterTopic4" value="Shoulder"/>
        <global-property name="PatientSupporterTopic5" value="Head"/>
        <global-property name="PatientSupporterTopic6" value="Back"/>
        <global-property name="PatientSupporterTopic7" value="Arm"/>
…
```

Global Properties

RuleResponder with Mule 1.X

```
<endpoint-identifier name="${PatientSupporterOAName}" value="jms://topic:${PatientSupporterTopicName}" />
    <endpoint-identifier name="${PatientSupporterTopicName}_${PatientSupporterTopic1}"
value="${TOMCAT}/${PatientSupporterOAName}${PatientSupporterTopic1}/" />
    <endpoint-identifier name="${PatientSupporterTopicName}_${PatientSupporterTopic2}"
value="${TOMCAT}/${PatientSupporterOAName}${PatientSupporterTopic2}/" />
    <endpoint-identifier name="${PatientSupporterTopicName}_${PatientSupporterTopic3}"
value="${TOMCAT}/${PatientSupporterOAName}${PatientSupporterTopic3}/" />
    <endpoint-identifier name="${PatientSupporterTopicName}_${PatientSupporterTopic4}"
value="${TOMCAT}/${PatientSupporterOAName}${PatientSupporterTopic4}/" />
    <endpoint-identifier name="${PatientSupporterTopicName}_${PatientSupporterTopic5}"
value="${TOMCAT}/${PatientSupporterOAName}${PatientSupporterTopic5}/" />
    <endpoint-identifier name="${PatientSupporterTopicName}_${PatientSupporterTopic6}"
value="${TOMCAT}/${PatientSupporterOAName}${PatientSupporterTopic6}/" />
    <endpoint-identifier name="${PatientSupporterTopicName}_${PatientSupporterTopic7}"
value="${TOMCAT}/${PatientSupporterOAName}${PatientSupporterTopic7}/" />
…
```

RuleResponder with Mule 3.0

```
<jms:endpoint name="PatientSupporter" topic="${PatientSupporterTopicName}" />
```

```
        <http:endpoint name="patientSupporter_Leg"
                address="${TOMCAT}/${PatientSupporterOAName}${PatientSupporterTopic1}/" />
        <http:endpoint name="patientSupporter_Torso"
                address="${TOMCAT}/${PatientSupporterOAName}${PatientSupporterTopic2}/" />
        <http:endpoint name="patientSupporter_Neck"
                address="${TOMCAT}/${PatientSupporterOAName}${PatientSupporterTopic3}/" />
        <http:endpoint name="patientSupporter_Shoulder"
                address="${TOMCAT}/${PatientSupporterOAName}${PatientSupporterTopic4}/" />
        <http:endpoint name="patientSupporter_Head"
                address="${TOMCAT}/${PatientSupporterOAName}${PatientSupporterTopic5}/" />
        <http:endpoint name="patientSupporter_Back"
                address="${TOMCAT}/${PatientSupporterOAName}${PatientSupporterTopic6}/" />
        <http:endpoint name="patientSupporter_Arm"
                address="${TOMCAT}/${PatientSupporterOAName}${PatientSupporterTopic7}/" />
…
```

Note: The link of global properties with "_" is not supported in Mule 3.0. For example, "${PatientSupporterTopicName}_${PatientSupporterTopic1}" is not a correct address of endpoint.

Services and Component

RuleResponder with Mule 1.X

```
…
<endpoint-identifier name="${PatientSupporterOAName}" value="jms://topic:${PatientSupporterTopicName}" />….

<mule-descriptor name="${PatientSupporterOAName}"
                        implementation="${MULE_IMPLEMENTATION}" singleton="true">
                        <inbound-router>
                                <endpoint address="jms://topic:${PatientSupporterTopicName}"/>
                                <endpoint name="httpEndpointPS"
address="http://localhost:${PatientSupporter_PORT}">
        </endpoint>
                        </inbound-router>
                        <properties>
                                <property name="rulebase" value="${PatientSupporter_Prova}" />
                        </properties>
</mule-descriptor>
…
```

RuleResponder with Mule 3.0

```
…
<jms:endpoint name="PatientSupporter" topic="${PatientSupporterTopicName}" />
…

<model name="PatientSupporterModel">
                <service name="PatientSupporterService" initialState="started">
                        <inbound>
                                <jms:inbound-endpoint ref="PatientSupporter">
                                        <properties>
                                                <!-- Each receiver polls with a 5 second interval -->
                                                <spring:entry key="rulebase" value="${PatientSupporter_Prova}" />
                                        </properties>
                                </jms:inbound-endpoint>
                                <http:inbound-endpoint name="httpEndpointPS"
```

```
                                        address="http://localhost:${PatientSupporter_PORT}" />
                    </inbound>
                    <component class="${MULE_IMPLEMENTATION}" />
                </service>
        </model>
…
```

Note that names on inbound and outbound endpoints cannot be used to send or receive messages; using a named global endpoint instead (Use an endpoint-identifier instead in Mule 1.X).

15 Appendix: Prova 2.x-to-Prova 3.0 Upgrade

Rule Responder uses Mule ESB to handle the message-based interaction between agents, which are partially implemented by Prova. Thus the upgrade from Prova 2.x to Prova 3.0 is based on the Mule 3.0.The users need to upgrade the Mule first before upgrading Prova in Rule Responder as described in Section 15.2.

Prova 3 has benn changed a lot compared to the previous version. In this part, we only focus on the sections which are very important to communication between Prova agent and Rule Responder. Generally speaking, there are four main aspects that we need to consider in Prova 3.0 in order to upgrade it in Rule Responder:

(1) **Prova Communicator**, which acts as a Prova instance in RuleML Responder System.

(2) **Prova Agent** defines the methods of sending messages to other agents and receiving messages from other agents. We override its methods to translate the Prova message from and to the RuleML (http://ruleml.org/), which is a unifying family of XML-serialized rule languages spanning across all industrially relevant kinds of Web rules. It accommodates and extends other recent rule languages, building interoperation bridges between them.

(3) **Prova Message** describes the messages which are processed by Prova Agents. Prova 3.0 now uses the ProvaList to describe the Prova Message instead of the RMessage in previous Prova 2.x.

(4) **Translation between RuleML and ProvaList**. In Rule Responder all the messages among distributed multi agents are transported as RuleML. Mule translates the query request in RuleML to ProvaList before Prova Agent receives it. On the contrary, Mule translates the ProvaList to RuleML when Prova Agent forwards the query request to other agents.

15.1 Prova Communicator

Prova communicator acts as a Prova instance in RuleML Responder System. In Prova 2.x, the Prova communicator was implemented by the ws.prova.Communicator. But in Prova 3.0, Prova communicator has been changed to ws.prova.api2.ProvaCommunicatorImpl which implements the interface of ws.prova.api2.ProvaCommunicator. Correspondingly, the constructors of ws.prova.api2.ProvaCommunicator are redesigned. Following are Prova Communicator constructors in two versions:

Prova 2.x

```
…
new Communicator(agentName, null, new BufferedReader(
                                        new InputStreamReader(getClass().getResourceAsStream(
                                                rulebase))), -1, Communicator.ASYNC, this);
```

```
…
```

Prova 3.0

```
…
comm = new ProvaCommunicatorImpl(agentName, null, new BufferedReader(
                                                    new InputStreamReader(getClass().getResourceAsStream(
                                                                rulebase))), ProvaCommunicatorImpl.ASYNC,this, null);
…
```

Note: We can see that the two constructors are not isomorphic and the last parameter of ProvaCommunicatorImpl constructor which is initialized null here is the global Map of arguments which are passed to the initialized rulebase. “this” in the parameters which is the Prova Agent and describes in the following section.

15.2 Prova Agent

Prova Agent implements the methods of message sending and receiving. In Prova 2.x, the Java class of ws.prova.mule.impl.ProvaUMOImpl implemented the interface ws.prova.esb.ProvaUMO as the Prova Agent. In Prova 3.0, ProvaUMOImpl uses the interface ws.prova.esb2.ProvaAgent instead of ws.prova.esb.ProvaUMO. Following are changes in both versions of Prova:

Prova 2.x

```
…
public class ProvaUMOImpl extends LogComponent implements ProvaUMO,
                        Initialisable, Callable, ServiceAware {
…
public void receive(ProvaList rMsg) throws Exception {
….
}
public void send(String receiver, ProvaList provaList) throws Exception {
…
}

public String getAgentName() {
…
}
```

Prova 3.0

```
…
public class ProvaUMOImpl extends LogComponent implements Initialisable,
                        Callable, ServiceAware, ProvaAgent {
…
public void receive(RMessage rMsg) throws Exception {
….
}
public void send(String agent, String receiver, RMessage message)
                                throws Exception {
…
}
```

Note that, both interfaces have the method of message sending and receiving. However, the message has been changed from RMessage to ProvaList. We will give the detail in the next section.

15.3 Prova Message

Prova Message describes the messages which are processed by Prova Agents. Prova 3.0 uses the interface ws.prova.kernel2.ProvaList instead of the class ws.prova.RMessage to describe it. However, we know that Rule Responder uses the RuleML as a standard to exchange messages among distributed rule agents. It is necessary for Prova Agent to translate the RuleML to Prova Message after receiving the message and to translate the Prova Message to RuleML before sending the message to other rule agents. Following changes marked in blue which are involved in both versions of Prova:

Prova 2.x

```
...
public Object onCall(MuleEventContext context) throws Exception {
                        // Extract Prova RMessage
                        MuleMessage inbound = context.getMessage();
                        String temp = inbound.getPayloadAsString();
                        RMessage rMsg = null;

                        try {
                                if (inbound.getPayload() instanceof ObjectMessage) {
                                        Object message = ((ObjectMessage) inbound.getPayload())
                                                        .getObject();
                                        if (message instanceof RMessage)
                                                rMsg = (RMessage) message;
                                } else if (inbound.getPayload() instanceof ActiveMQTextMessage) {
                                        RuleML2ProvaTranslator ruleml2prova = new RuleML2ProvaTranslator();
                                        Object message = ruleml2prova
                                                        .transform(((ActiveMQTextMessage) context.getMessage()
                                                                        .getPayload()).getText());
                                        if (message instanceof RMessage)
                                                rMsg = (RMessage) message;
                                } else if (inbound.getPayload() instanceof String) {
                                        RuleML2ProvaTranslator ruleml2prova = new RuleML2ProvaTranslator();
                                        String http_message = URLDecoder.decode(
                                                        inbound.getPayloadAsString(), inbound.getEncoding());
                                        if (http_message.indexOf("<") != -1) { // message comes from HTTP request
                                                http_message = http_message.substring(http_message
                                                                .indexOf("<")); // remove extra HTTP "parameter" signs
                                        }
                                        Object message = ruleml2prova.transform(http_message);
                                        if (message instanceof RMessage)
                                                rMsg = (RMessage) message;
                                } else {
                                        Object message = context.getMessage().getPayload();
                                        if (message instanceof RMessage)
                                                rMsg = (RMessage) message;
                                }
                        } catch (Exception ex) {
                                ex.printStackTrace();
                                logger.error("Translation of message into RMessage failed: "
                                                + inbound);
                                logger.error(ex);
                        }
                        if (rMsg == null) {
```

```
                        logger.error("Message has format: " + inbound);
                        // TODO: Failure handling for wrong message formats
                        return null; // do nothing
                }
                // Synchronous processing of queries in order to synchronously return
                // the results to synchronous requesters (e.g. a Web browser)
                if (inbound.getPayload() instanceof String) {
                        if (!rMsg.performative().equals("query-sync")) {
                                comm.addMsg(rMsg);
                                context.setStopFurtherProcessing(true);
                                return null;
                        } else {
                                int timeout = 10000; // default timeout
                                // register temp. synchronous UMO agent
                                if (temp.contains("BRF-2011")) {
                                        timeout = 300000;
                                }

                                tmpAgent = System.currentTimeMillis()+"";
                                rMsg.getFixed()[2] =ProvaConstantImpl.create(tmpAgent);
                                // register the temporary object which acts as user
                                try {

                                        MuleRegistry helper = service.getMuleContext()
                                                        .getRegistry();
                                        EndpointBuilder builder = new DefaultTransportServiceDescriptor(
                                                        tmpAgent, new Properties(), null)
                                                        .createEndpointBuilder("jms://topic:" + tmpAgent);
                                        helper.registerEndpointBuilder(tmpAgent, builder);

                                } catch (Exception e) {
                                        e.printStackTrace();
                                        context.setStopFurtherProcessing(true);
                                        return e.toString()
                                                        + " This user is already registered. Please use another user
name as sender address."; // return
                                }
                        // collect synchronously all answers
                                ....
        }
...
```

Prova 3.0

```
...
public Object onCall(MuleEventContext context) throws Exception {
                MuleMessage inbound = context.getMessage();
                String temp = inbound.getPayloadAsString();
                // translate message into execution syntax
                ProvaList rMsg = null;
                try {
                        if (inbound.getPayload() instanceof ObjectMessage) {
                                Object message = ((ObjectMessage) inbound.getPayload())
```

```
                                        .getObject();
                    if (message instanceof ProvaList)
                            rMsg = (ProvaList) message;
            } else if (inbound.getPayload() instanceof ActiveMQTextMessage) {
                    // translate RuleML message to Prova message
                    // TODO: might be move to config file (as an inbound endpoint
                    // translator)
                    RuleML2ProvaTranslator ruleml2prova = new RuleML2ProvaTranslator();
                    Object message = ruleml2prova
                                    .transform(((ActiveMQTextMessage) context.getMessage()
                                                    .getPayload()).getText());
                    if (message instanceof ProvaList)
                            rMsg = (ProvaList) message;
            } else if (inbound.getPayload() instanceof String) {
                    RuleML2ProvaTranslator ruleml2prova = new RuleML2ProvaTranslator();
                    String http_message = URLDecoder.decode(
                                    inbound.getPayloadAsString(), inbound.getEncoding());
                    if (http_message.indexOf("<") != -1) { // message comes from HTTP request
                            http_message = http_message.substring(http_message
                                            .indexOf("<")); // remove extra HTTP "parameter" signs
                    }
                    Object message = ruleml2prova.transform(http_message);
                    if (message instanceof ProvaList)
                            rMsg = (ProvaList) message;
            } else {
                    Object message = context.getMessage().getPayload();
                    if (message instanceof ProvaList)
                            rMsg = (ProvaList) message;
            }
    } catch (Exception ex) {
            ex.printStackTrace();
            logger.error("Translation of message into RMessage failed: "
                            + inbound);
            logger.error(ex);
    }

    if (rMsg == null) {
            // The message is not a RMessage or can not be translated into a
            logger.error("Message has format: " + inbound);
            // TODO: Failure handling for wrong message formats
            return null; // do nothing
    }
    // Synchronous processing of queries in order to synchronously return
    // the results to synchronous requesters (e.g. a Web browser)
    if (inbound.getPayload() instanceof String) {
            // if not a synchronous query then add message to Prova Agent
            if (!rMsg.performative().equals("query-sync")) {
                    comm.addMsg(rMsg);
                    // context.setStopFurtherProcessing(true);
                    return null;
            } else {
                    int timeout = 10000; // default timeout
                    // register temp. synchronous UMO agent
```

```
                                        if (temp.contains("RuleML-2010")) {
                                                timeout = 300000;
                                        }
                                        tmpAgent = ((ProvaConstant) rMsg.getFixed()[2]).toString();
                                        // register the temporary object which acts as user
                                        try {
                                                MuleRegistry helper = service.getMuleContext()
                                                                .getRegistry();
                                                EndpointBuilder builder = new DefaultTransportServiceDescriptor(
                                                                tmpAgent, new Properties(), null)
                                                                .createEndpointBuilder("jms://topic:" + tmpAgent);
                                                helper.registerEndpointBuilder(tmpAgent, builder);

                                        } catch (Exception e) {
                                                e.printStackTrace();
                                                context.setStopFurtherProcessing(true);
                                                return e.toString()
                                                                + " This user is already registered. Please use another user
name as sender address."; // return
                                        }
                                        // collect synchronously all answers
                                        ....
                        }
                }
...
```

15.4 Translation between RuleML and ProvaList Object

15.4.1 RuleML2Prova

In previous version of Rule Responder, the RuleML2Prova translator firstly uses the XSLT to translate the message in RuleML to a String which has the structures of RMessage. Then the translator keeps on refining the architecture of RMessage to complete the translation. In this upgrade, we reuse the XSLT to translate the message in RuleML to a String which has the main information the message. Then we use a Java class ws.prova.mule.impl.String2ProvaList to construct ProvaList object from the String. Following shows the changes marked in blue in class ws.prova.mule.impl.RuleML2Prova.

Prova 3.0

```
...
public Object doTransform(Object src, String encoding)
                                throws TransformerException {
....
                                        xmlSource = new StreamSource(is); // XML Source

                                        // transform XML message into Prova RMessage
                                        transformer = tFactory.newTransformer(xslSource);

                                        // Perform the transformation.
                                        StringWriter provaMessage = new StringWriter();
                                        transformer.transform(xmlSource, new StreamResult(provaMessage));
                                        String output = provaMessage.toString();
```

```
            // invoke another class to finish the tranlation
                                    ProvaList list = new String2ProvaList().createProvaList(output);
                                    return list;
        }
....
```

ws.prova.mule.impl.String2ProvaList

```
package ws.prova.mule.impl;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import ws.prova.kernel2.ProvaKnowledgeBase;
import ws.prova.kernel2.ProvaList;
import ws.prova.kernel2.ProvaObject;
import ws.prova.reference2.ProvaConstantImpl;
import ws.prova.reference2.ProvaKnowledgeBaseImpl;
import ws.prova.reference2.ProvaListImpl;

public class String2ProvaList {

        public ProvaKnowledgeBase kb = new ProvaKnowledgeBaseImpl();
        private String regex = "\\.?@\\d+=";
        private LinkedList ll = new LinkedList();

        public ProvaList createProvaList(String output) {
                String[] items = output.split(",");
                int i = 0;
                if (items[0].indexOf("id") != -1)
                        i = 1;
                String id = items[i].substring(0, items[i].length() - i);

                ProvaList list = ProvaListImpl.create(new ProvaObject[] {
                                        ProvaConstantImpl.create(id),
                                        ProvaConstantImpl.create(removeQutationMark(items[i + 1])),
                                        ProvaConstantImpl.create(removeQutationMark(items[i + 2])),
                                        ProvaConstantImpl.create(removeQutationMark(items[i + 3])),
                                        parseContent(output.substring(output.indexOf(items[i + 3])
                                                        + items[i + 3].length() + 1, output.length() - 1)) });
                return list;

        }

        private String removeQutationMark(String string) {
                // TODO Auto-generated method stub
                if (string.startsWith("\""))
                        string = string.substring(1);
                if (string.endsWith("\""))
                        string = string.substring(0, string.length() - 1);
```

```
		return string;
	}

	private String removeComma(String content) {
		// TODO Auto-generated method stub
		if (content.startsWith(","))
			content = content.substring(1);
		if (content.endsWith(","))
			content = content.substring(0, content.length() - 1);
		return content;
	}

	private ProvaList parseContent(String content) {

		List list = new ArrayList();
		// TODO Auto-generated method stub
		if (content.startsWith("["))
			content = content.substring(1, content.length() - 1);
		if (content.indexOf("=") != -1) {
			if (content.contains(System.getProperty("line.separator")))
				content = content.replaceAll(
						System.getProperty("line.separator"), "");

			content = content.substring(0, content.length() - 1);
			Pattern pattern = Pattern.compile(regex);
			Matcher m = pattern.matcher(content);
			while (m.find()) {
				content = m.replaceAll(",");
				content = "substitutions" + content;
			}
		}
		int tag = 0;
		if (content.indexOf("[") == -1 || content.indexOf("]") == -1) {
			parseSimpleContent(list, content);
		} else
			for (int i = 0; i < content.length(); i++) {
				if (content.charAt(i) == '['
						|| (i == content.length() - 1 && content.charAt(i) != ']')) {
					if ((i - tag) != 0 && ll.isEmpty()) {
						String temp = "";
						if (i == content.length() - 1)
							temp = content.substring(tag, i + 1);
						else
							temp = content.substring(tag, i);
						if (!temp.equalsIgnoreCase(",")) {
							parseSimpleContent(list, temp);
						}
					}
					ll.addFirst(i);
					tag = i;
				} else if (!ll.isEmpty() && content.charAt(i) == ']') {
					int pos = (Integer) ll.removeFirst();
					if (ll.isEmpty())
```

```
                                        list.add(parseContent(content.substring(pos, i + 1)));
                                tag = i + 1;
                        }

                }
        return ProvaListImpl.create(list);
    }

    private void parseSimpleContent(List list, String content) {
        // TODO Auto-generated method stub
        content = removeQutationMark(content);
        content = removeComma(content);
        if (content.trim().equals(""))
            return;
        String[] objs = content.split(",");
        for (int j = 0; j < objs.length; j++) {
            String temp = objs[j].trim();
            if (temp.equals(""))
                return;
            char firstChar = temp.charAt(0);
            if (Character.isLetter(firstChar)
                    && Character.isUpperCase(firstChar)) {
                if (objs[j].indexOf(":") != -1) {
                    int pos = objs[j].indexOf(":");
                    list.add(kb.generateVariable(objs[j].substring(0, pos)
                            + "." + objs[j].substring(pos + 1) + "^^"
                            + objs[j].substring(0, pos)));

                } else
                    list.add(kb.generateVariable(objs[j]));
            } else if (Character.isLetter(firstChar)
                    && Character.isLowerCase(firstChar))
                list.add(ProvaConstantImpl.create(objs[j]));
            else if (firstChar == ('\"') || firstChar == ('\''))
                list.add(ProvaConstantImpl.create(removeQutationMark(objs[j])));
            else if (isNumeric(objs[j]))
                list.add(ProvaConstantImpl.create(objs[j]));
            else if (firstChar == '@')
                list.add(kb.generateVariable(removeQutationMark(objs[j])));
        }
    }

    private boolean isNumeric(String str) {
        for (int i = str.length(); --i >= 0;) {
            int chr = str.charAt(i);
            if (chr == 46)
                continue;
            if (chr < 48 || chr > 57)
                return false;
        }
        return true;
    }
}
```

15.4.2 Prova2RuleML

In Prova 2.x, the body of RMessage consists of ComplexTerm, ConstantTerm and VariableTerm as same as ComlexTerm itself. And in Prova 3.0, ProvaList object consists of ProvaList itself, ProvaConstant and ProvaVariable. Following shows the main translation method doTransform(Object src, String encoding) and other related methods (the main changes are marked in blue).

Prova 3.0

```
...
        public Object doTransform(Object src, String encoding) {
                if (src instanceof ProvaList) {
                        try {
                                ProvaList pmes = (ProvaList) src;
                                ProvaObject[] provaObjects = pmes.getFixed();
                                boolean isAnswer;
                                if (pmes.performative().equals("stop_communication"))
                                        return pmes;
                                else if (pmes.performative().equals("answer"))
                                        isAnswer = true;
                                else
                                        isAnswer = false;
                                ArrayList<ProvaObject> payload = new ArrayList<ProvaObject>();
                                for (int i = 0; i < provaObjects.length; i++) {
                                        if (provaObjects[i] instanceof ProvaList) {
                                                ProvaObject[] objects = ((ProvaList) provaObjects[i])
                                                                .getFixed();
                                                for (int j = 0; j < objects.length; j++) {
                                                        payload.add(objects[j]);
                                                }
                                        }
                                }
                                Object conv_id = ((ProvaConstantImpl) provaObjects[0])
                                                .toString();
                                String ruleMLpayload = "";
                                if (payload.size() > 0) {
                        if (payload.get(0).toString().indexOf(" ") == -1) {
                                ruleMLpayload = ruleMLpayload + spacerItem + spacerItem
                                                + "<Atom>\n";
                                for (int i = 0; i < payload.size(); i++) {
                                                if (i == 0)
                                                ruleMLpayload = ruleMLpayload + spacerItem
                                                        + spacerItem + spacerItem + "<Rel>"
                                                        + payload.get(i).toString()
                                                        + "</Rel>\n";
                                        else {
                                                        Object t = payload.get(i);
                                                        ruleMLpayload += serializedProvaObject(t,
                                                                        isAnswer, spacerItem + spacerItem
                                                                                        + spacerItem);
                                                }
                                        }
                                }
                                ruleMLpayload = ruleMLpayload + spacerItem + spacerItem
```

```
                                                + "</Atom>\n";
                        } else {
                                ruleMLpayload = ruleMLpayload + "<Ind>\"";
                                ruleMLpayload = ruleMLpayload + payload.get(0);
                                ruleMLpayload = ruleMLpayload + "\"</Ind>\n";
                        }
                        String rulemlMes = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n"
                                + "<RuleML "
                                + "xmlns=\"http://www.ruleml.org/0.91/xsd\" "
                                + "xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" "
                                + "xsi:schemaLocation=\"http://www.ruleml.org/0.91/xsd "
                                + "http://ibis.in.tum.de/research/ReactionRuleML/0.2/rr.xsd\">\n"
                                + "<Message mode=\"outbound\" directive=\""+ pmes.performative()+ "\">\n"
                                + spacerItem+ "<oid>\n"+ spacerItem+ "<Ind>"+ conv_id.toString()+ "</Ind>\n"+
spacerItem+ "</oid>\n"
                                + spacerItem+ "<protocol>\n"+ spacerItem+ spacerItem+ "<Ind>esb</Ind>\n"+
spacerItem+ "</protocol>\n"
                                + spacerItem+ "<sender>\n"
                                + spacerItem+ spacerItem+ "<Ind>"+ ((ProvaConstant) provaObjects[2]).toString()+
"</Ind>\n"
                                + spacerItem+ "</sender>\n"
                                + spacerItem+ "<content>\n"+ ruleMLpayload+ spacerItem+ "</content>\n"
                                + "</Message>\n" + "</RuleML>\n";

                        return rulemlMes; // return serialized RuleML XML message

                } catch (Exception e) {
                        logger.error("Error during translation of RMessage into RuleML - send RMessage instead: "+ src);
                        logger.error(e);
                        return src; // simply return untranslated message
                }
        }
        return src;
}

private String serializedProvaObject(Object t, boolean isAnswer,
                        String spacer) {
                String serializedContent = "";
                if ((t instanceof ProvaVariable) && !isAnswer)
                        serializedContent = serializeProvaVariable((ProvaVariable) t,
                                        spacer);
                else if (t instanceof ProvaList)
                        serializedContent = serializeProvaList((ProvaList) t, spacer,
                                        isAnswer);
                else if (t instanceof ProvaConstant)
                        serializedContent = serializeProvaConstant(t, spacer);
                return serializedContent;
}
....
```

There are also changes between translation of ComplexTerm, ConstantTerm, VariableTerm and ProvaList, ProvaConstant, ProvaVariable. Following are methods involved in Prova 3.0.

Prova 3.0

```
private String serializeProvaVariable(ProvaVariable ct, String spacer) {
		String serializedVariable = "";
		String variable = ct.toString();
		String type = "";
		if (variable.indexOf(":") != -1 && variable.indexOf("://") == -1) {
			type = variable.substring(variable.lastIndexOf(":") + 1);
			variable = variable.substring(0, variable.lastIndexOf(":"));
		}

		if (variable.indexOf(".") != -1) {
			type = variable.substring(0, variable.lastIndexOf("."));
			variable = variable.substring(variable.lastIndexOf(".") + 1);
		}
		if (isNumeric(variable))
			variable = "@" + variable;
		if (variable.startsWith("<")) {
			variable = variable.substring(1, variable.length() - 1);
			variable = "@" + variable;
		}
		if (type.equals(""))
			serializedVariable = serializedVariable + spacer + spacerItem
					+ "<Var>" + variable + "</Var>\n";
		else
			serializedVariable = serializedVariable + spacer + spacerItem
					+ "<Var type=\"" + type + "\">" + variable + "</Var>\n";
		return serializedVariable;
	}

private String serializeProvaConstant(Object ct, String spacer) {
		String serializedConstant = "";
		String constant = ct.toString();
		String type = "";
		if (constant.indexOf(":") != -1 && constant.indexOf("://") == -1) {
			int pos = constant.indexOf(":");
			type = constant.substring(0, pos);
			constant = constant.substring(pos + 1);
		}

		if (!type.equalsIgnoreCase(""))
			serializedConstant = serializedConstant + spacer + spacerItem
					+ "<Ind type=\"" + type + "\">" + constant + "</Ind>\n";
		else
			serializedConstant = serializedConstant + spacer + spacerItem
					+ "<Ind>" + constant + "</Ind>\n";
		return serializedConstant;
	}

private String serializeProvaList(ProvaList ct, String spacer, boolean isAnswer) {
		ProvaObject[] tokens = ct.getFixed();

		String serializedProvaList = "";
		if (tokens.length > 0) {
			if (tokens[0].toString().indexOf(" ") == -1) { // Expr complex
```

```
                                serializedProvaList = serializedProvaList + spacer + spacerItem
                                                + "<Expr>\n";
                                for (int i = 0; i < tokens.length; i++) {
                                        if (i == 0)
                                                serializedProvaList = serializedProvaList + spacer
                                                                + spacerItem + spacerItem + "<Fun>" + tokens[i]
                                                                + "</Fun>\n";
                                        else {
                                                Object t = tokens[i];
                                                serializedProvaList = serializedProvaList
                                                                + serializedProvaObject(t, isAnswer, spacer
                                                                                + spacerItem);
                                        }
                                }
                        }
                        if (tokens.length > 0)
                                serializedProvaList = serializedProvaList + spacer + spacerItem
                                                + "</Expr>\n";
                } else { // Plex complex term
                        serializedProvaList = serializedProvaList + spacer + spacerItem
                                        + "<Plex>\n";
                        for (int i = 0; i < tokens.length; i++) {
                                Object t = tokens[i];
                                serializedProvaList = serializedProvaList
                                                + serializedProvaObject(t, isAnswer, spacer
                                                                + spacerItem);
                        }
                        if (tokens.length > 0)
                                serializedProvaList = serializedProvaList + spacer + spacerItem
                                                + "</Plex>\n";
                }
                return serializedProvaList;
        }
```