Loading and managing rules dynamically from a database

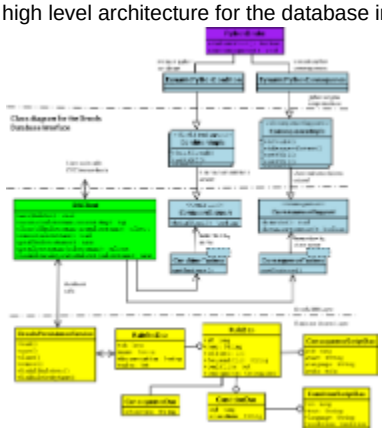
Storing and managing Rules in a database
(note that this example uses the Drools SPI API - which will be changing significantly with version 3 - but it shows how you can use custom front ends to drive the rule engine, in particular using a dynamic language like python).

Thanks to Sujit Pal for this article (sujit.pal@comcast.net) and the contribution.

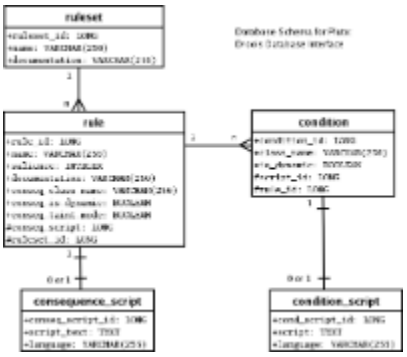
(Note that this is being adapted into a rule repository for Drools 3, adding in versioning, history, searching etc.)

Introduction

- Why use Rules - Allows us to abstract changeable business logic out of an application. Changing application behavior by changing the rules is faster and cheaper than having to make changes in the code to accomodate the new rules
- Why put Rules in a Database - Putting rules into the database allows us to change business rules while the application is running.
  Architecture

- The application is divided into two main parts - the Java database interface and the web interface. The Java interface can be used to programmatically manipulate rulesets in the database and to execute named rulesets. The web interface can be used to create and manage the rulesets in the database.
- The high level architecture for the database interface is shown below (click on images to enlarge):
  
- Conditions and Consequences can be written either as Java classes (extending ConditionSupport and ConsequenceSupport respectively) or as Python scriptlets (using DynamicPythonCondition and DynamicPythonConsequence respectively). Some examples of Java conditions and consequences are available in the test/com/cnwk/pluto/conditions and test/com/cnwk/pluto/consequences directoryof the download.
  Database configuration

- The rulesets need a database to store them. The program has been tested with MySQL and PostgreSQL.
  ER diagram for the schema is shown below.



- MySQL and PostgreSQL versions of the schema are available in doc/ruledb_mysql.sql and doc/ruledb_pgsql.sql of the download respectively.
- To modify the database connection parameters, the etc/hibernate.properties file needs to be updated.
  Web interface configuration
  Web interface configuration

- The web based rule management interface has been designed with Caucho resin, but should work with any standard servlet container.
- To deploy to your container of choice, please update the value of deploy.dir in the build.xml of the download to point to the desired location for the web application.

- Run ant deploy to deploy the code to this location.
- Start up the servlet container at port 8080 (for example) on localhost.
- Point your browser at the following URL:

http://localhost:8080/pluto/list.html

Database interface usage examples

- To execute a stored ruleset, the client instantiates and interacts with the DBClient object, and calls the execute method with a named ruleset and a context object (a Map) of parameters to pass to the ruleset.

```
DBClient client = new DBClient();
Map results = client.execute("MyRuleSet", context);
```

- To list all rulesets in the database, call the getAllRuleSetNames() method.

```
List names = client.getAllRuleSets();
```

- To load a particular ruleset to work on it from your application, call the getRuleSetByName() method.

```
RuleSet ruleset = client.getRuleSetByName("MyRuleSet");
```

- Writing a Java Condition - extend the ConditionSupport class, provide a constructor and setters for all context data that is needed by the Condition, and override the doIsAllowed() method. The ConditionSupport provides the functionality to pull all information from the context for which setters are declared into the Condition bean, and uses the value returned from doIsAllowed() in its isAllowed() method. An example of a Java condition follows:

```
public class MyCondition extends ConditionSupport {


            public MyCondition(Rule rule) {
                super(rule);
            }


            /**
             * There should be something called "data" in the Context
             */
            public void setData(Data data) {
                this.data = data;
            }


            public boolean doIsAllowed() throws ConditionException {
                // do something with data
                return someBoolean; // a true if condition passed, else false
            }
        }
```

- Writing a Java Consequence - extend the ConsequenceSupport class, provide a constructor, setters for everything that needs to be pulled from the Context, getters for everything that needs to be set back in the Context, and override the doReassertContext() and the doInvoke() methods. The doReassertContext() method needs to be overriden only if the Consequence changes the WorkingMemory. The ConsequenceSupport will take care of pulling the named data values from the Context before the invoke() call and putting them back in the Context after the invoke() call. An example of a Java consequence follows:

```
public class MyConsequence extends ConsequenceSupport {


            public MyConsequence(Rule rule) {
                super(rule);
            }


            public Data getData() {
                return data;
            }


            public void setData(Data data) {
                this.data = data;
            }


            // other getter and/or setters


            public boolean doReassertContext() {
                // override only if the consequence taints the working
                // memory so the facts need to be re-asserted.
                return false;
            }


            public void doInvoke() throws ConsequenceException {
                // do something with data
            }
        }
```

- Writing a Python Condition - instantiate a DynamicPythonCondition, which is a subclass of ConditionSupport. Set the script text and context into the object. An example of a Dynamic Python Condition follows:

```
Condition condition = new DynamicPythonCondition(rule);
        condition.setScriptText("cart.getState() == \"Exploded\"");
        condition.setContext(context);
```

- Writing a Python Consequence - instantiate a DynamicPythonConsequence, which is a subclass of ConsequenceSupport. Set the script text, context and the taint mode (true if facts need to be re-asserted into working memory). Here is an example:

```
Consequence consequence = new DynamicPythonConsequence(rule);
        consequence.setScriptText("cart.setState(\"Exploded\")");
        consequence.setTaintMode(false);
        consequence.setContext(context);
```

Web interface usage examples

- Listing all rulesets in database - this is the start page of the web interface. Each Ruleset has an Edit, Clone and Delete links. Clicking the Edit link will lead to a detailed view of the Ruleset and clicking Delete will delete the Ruleset from the database. Here is a screenshot:

- Adding a ruleset - a new ruleset can be added by filling in the form at the bottom of the list page and clicking submit. Alternatively, clicking on the Clone link next to each Ruleset will clone the ruleset for modifications.
- Editing a ruleset - Clicking on the Edit link will bring up the detail screen for that Ruleset. The name of a Ruleset can be edited by clicking the small Edit link near the Ruleset name. A list of Rules in the Ruleset follows the Ruleset name. Here is a screenshot:

Expanding a rule for viewing or editing - Clicking on the Expand link next to each Rule will bring up a detail view of the Rule. Here is a screenshot:

Adding a rule to a ruleset - Clicking on the Add New Rule link at the bottom of the edit screen brings up a form to add a new Rule to the ruleset. Here is a screenshot:

Editing a rule - Clicking the Edit link in the expanded Rule view will allow you to edit a Rule. Here is a screenshot:

- Conclusion

The web interface allows an easy way to manage rulesets in the database, in some cases it is less intimidating than writing DRL files in an editor.

The database interface takes away much of the complexity (and tunability) of understanding and writing code against the Drools API.

The web interface currently does not support versioning of rulesets, or locking rulesets while they are being modified, but this should be fairly easy to do if needed.

- Acknowlegements

Many thanks to Mark Proctor and the Drools team for the software.
Many thanks to Michael Neale for his support during the writing of this article.

The code for described in this article can be downloaded from here.