Dynamic UServ Implementation based on TAKE

Jens Dietrich - 07/11/07

This is an application implementing the UServ Rule Derby Scenario using TAKE. See http://www.businessrulesforum.com/2005_Product_Derby.pdf for a description of rules. This implementation is dynamic in the sense that the actual rules are compiled at runtime and can therefore be reloaded by the application. A static version of the UServ example is part of the standard TAKE distribution. A webstart enabled version of the static version can be launched by clicking on the link on the TAKE project homepage (http://code.google.com/p/take/).

License

Apache License version 2.0

Requirements

1. Java version 1.6 (or better) is required. This can be checked by running the following command: java -version
2. The JRE (Java runtime environment) is not sufficient, the full JDK (Java development kit) is required as the compiler is needed.
3. This distribution contains the web application **server/DUILookup.war**. This application has to be deployed on a local Java web server (such as Tomcat) running on port 8080. The application will try to connect to this web service using the following URL:
`http://localhost:8080/DUILookup/DUILookup`
You can check whether the web application is installed correctly by pointing your browser to this URL. You should see the word false in your browser window.
If you want to change the URL, you must also change the value of the variable URL in example.nz.org.take.compiler.userv.main.DUIConvictionInfoSource and recompile the application.

Starting the Application

The jar file in the root folder is executable. Alternatively, try executing the following command:

java -jar userv-dynamic-<version>.jar . Running the application from the console has the advantage that the log statements recording compilation, database and web service access are visible.

Interesting Features

1. The application uses external fact stores accessing a web service (DUI Conviction Lookup)

2. The application uses external fact stores accessing a relational database (a list of risky locations is stored in a HSQL database located in **data/locations** and accessed using JDBC)
3. The rules can be dynamically reloaded and are recompiled on the fly (there are two rule sets in **data/rules** to try this)
4. The user interface supports semantic reflection, that is, the user can inspect the rules used to compute a result