Business Rule Engine

Ruleengine Sample

"Rent a Ferrari"

Uwe Geercken – last update: 2016-08-06

Content

Overview

This document shows an example of a Business Logic and how it can be implemented using the Business Rules Maintenance Tool.

The tool is a web application which allows the user to construct and orchestrate business logic in an easy fashion.

Once the logic is setup, it can be exported into a single file and used with the Business Rule Engine "JaRE" - the Java Rule Engine - to evaluate if a given set of data from a file or database conforms to the rules or not.

Sample

The sample is an imaginary sample but related to real life so that it is easier to understand and follow.

A young girl – Maria – wants to rent a Ferrari for a fun ride. She is 25 years old, has a valid drivers license. She is willing to pay an extra fee if necessary (details later…).

The following pages show in detail how to construct the business logicusing the Business Rule Maintenance Tool.

Prerequisites

If you would like to implement this sample yourself, you need a running instance of the Business Rules Maintenance Tool.

The tool stores it's data in a MySQL/MariaDB database, so you need a running instance of MySQL/MariaDB.

A running instance of Apache Tomcat is required which runs the Business Rules Maintenance Tool.

First, install the MySQL/MariaDB database server if you don't already have a running instance. Make sure the database server is running and you can access it. Create a database named "ruleengine_rules". You may also use a different name, but this is the default name.

Next install Apache Tomcat and make sure, that it is running correctly.

Next download the Business Rules Maintenance Tool and the database schema from GitHub:

http://github.com/uwegeercken

Now, import the database schema you downloaded into the MySQL/MariaDB server. This will create the required tables and fields for the Business Rules Maintenance Tool. An example command to do this would be:

mysql -u [username] -p [databasename] < [schema filename]

If the database user name is "harry", the database name is "ruleengine_rules" and the filename (the database schema you downloaded) is "ruleengine_rules.sql", then the command is as follows:

mysql -u harry -p ruleengine_rules < ruleengine_rules.sql

Press [enter] after you have entered the command. You will be asked for a password. Provide the relevant password that belongs to user "harry". If no password is configured for the user, then leave away the "-p" parameter.

Go and check if the database schema was correctly imported and that tables and fields have been created.

Finally, copy (drop) the file of the Business Rules Maintenance Tool that you downloaded (the ".war" file) into the Apache Tomcat "webapps" folder.

Now everything is setup and the Business Rules Maintenance Tool is ready to use. Go and open your Web Browser (preferably Firefox) and enter the address, where the Apache Tomcat instance is running. A default installation of Apache Tomcat on your local computer is usually available at:

http://localhost:8080

The port (the number at the end) might have a different number – That depends on your installation.

Check if entering the above address will show a default Apache Tomcat start page.

To access the Business Rules Maintenance Tool enter following address in your Web Browser:

http://localhost:8080/rule_maintenance

You should now see a page with the title "Business Rules Maintenance Tool" at the top. At initial startup of the tool a configuration page is shown. You will need to enter the connection details to the database. Optionally you may provide details of a LDAP server, if you want the authentication of the users to be done via this server.

Save the configuration details. A message will indicate if the connection to the database server was successful.

Now you are ready to use the tool to construct your business logic.

Note: To login to the tool use:

userid = admin

password = admin

This is the default user and password that comes with the Business Rules Maintenance Tool installation.

It is highly recommended to change this password once you logged in the first time.

Inside the tool you will be able to create additional groups (roles) and users.

User Details

Let's assume that the data I want to check with the business logic comes from a database. It stores the details of the user that wants to rent a car. It has already been entered in the database at the reception of the rental car company and looks like this:

Last_Name	First_Name	Age	Drivers_License	Extra_Fee
Tall	Maria	25	Yes	Yes

The first row shows the names of the fields as they appear in the database. The second row shows the details of the user.

Business Rules of the car rental company

The company that rents cars has following business rules in place for renting a Ferrari:
- Somebody who wants to rent a Ferrari needs to have a valid drivers license
- Somebody who wants to rent a Ferrari needs to be of age 26 or above
- If the user is younger than 26 he/she might rent the Ferrari if an extra fee is paid

Sample - Graphical Overview

On the next page there is a graphical overview of the sample for which we will construct the business logic.

Ruleengine - Rule Sample

Maria, female
25 years old
Has drivers license
Wants to rent a Ferrari
Willing to pay extra fee

Data

Ruleengine

Result

rule 1) passes
rule 2) fails
rule 3) passes

Rulegroup: **passes**

Business logic

Rulegroup**:**
rule 1) Must have drivers license
   and
rule 2) Must be 26+ to rent a Ferrari
   or
rule 3) if younger – must pay extra fee

Logic:
rule 1 and (rule 2 or rule 3)
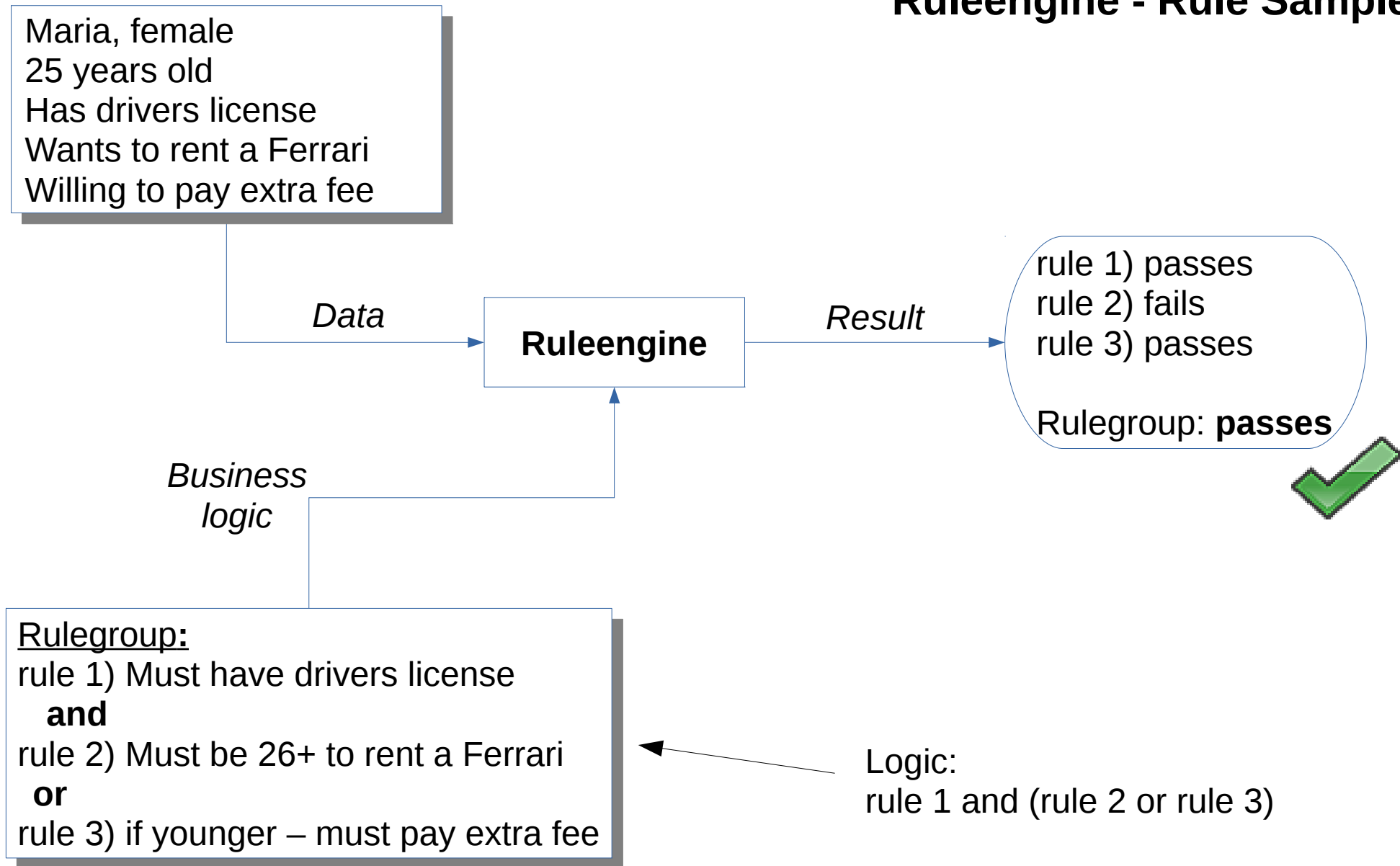
Step 1 – Create a Project

The first step is to create a project. It acts as a container for all the business logic. On the left side in the menu, click on "Project".
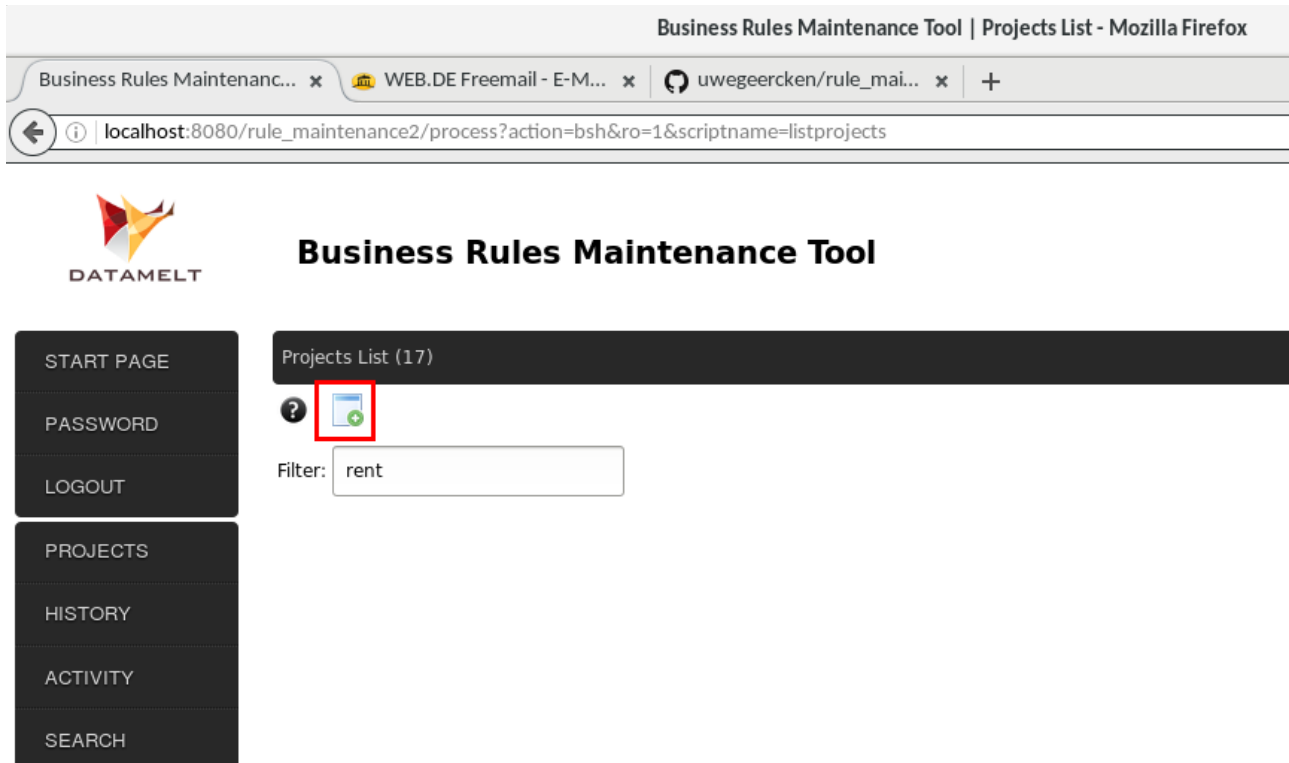
Click on the icon to add a new project. Enter a name a description for the project and select to which groups (roles) the project shall belong. Only users that are members of the selected group will be able to modify the project content. If you select the project as "Private project", users that are not in the same group will not see this project.

Click on "save" to save the project and then click on "close".

The project will now be shown on the project page.
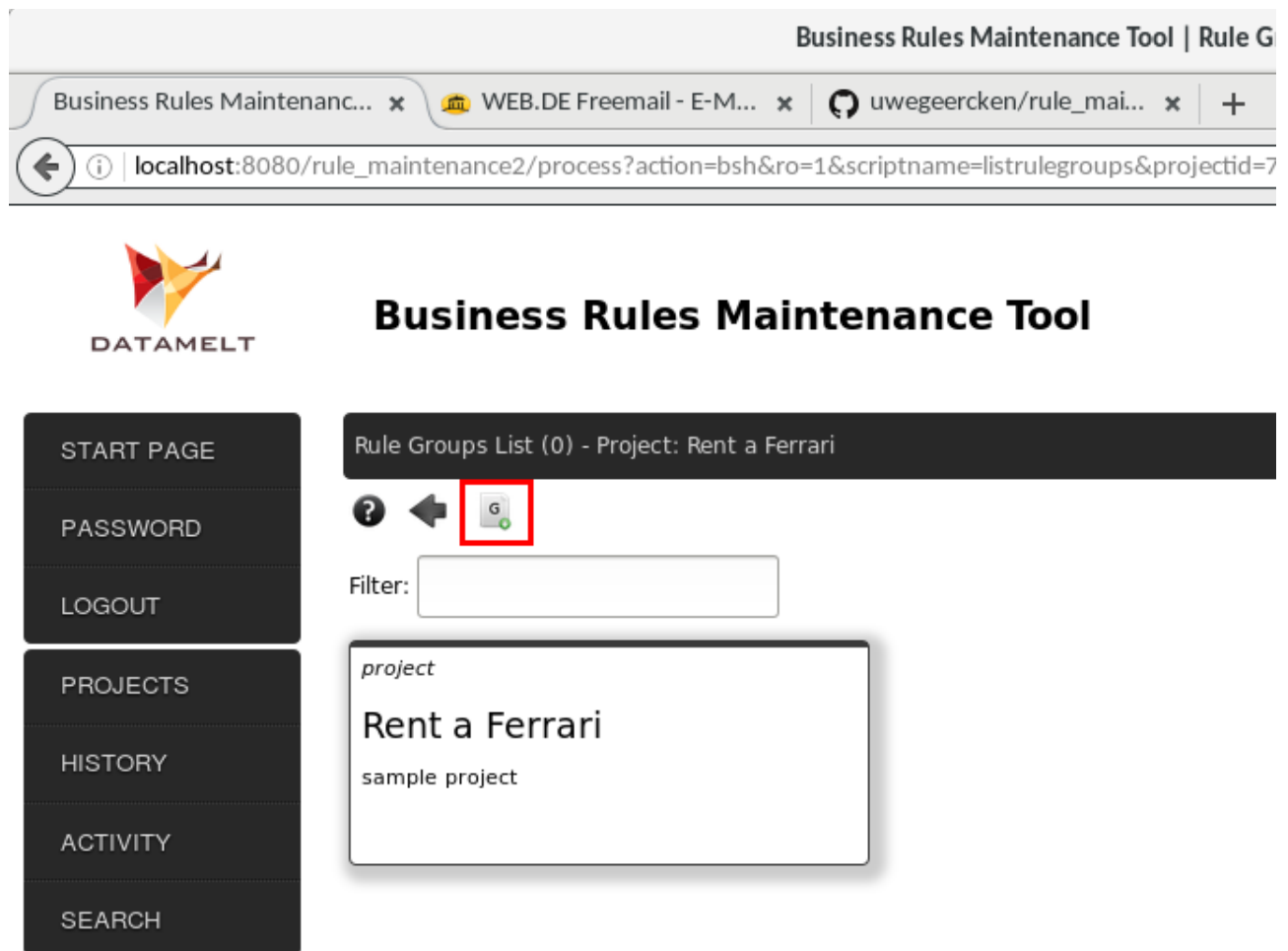
Step 2 – Create a Rulegroup

A rulegroup is part of a project and contains all rules that belong together and define a certain business logic. You may have multiple rulegroups in a project.

Click on the displayed name of the project and a page listing a rulegroups will be presented.

Click on the icon to create a new rulegroup.

A page is presented where the details of the rulegroup have to be entered:

Enter a name and description for the rulegroup. Select a date range from – to when the group shall be valid.

Click on "save" and then "close" to go back to the page listing all rulegroups for the project.

Ruleengine Sample - "Rent a Ferrari"

Your page should look similar to the one shown below.

Step 3 – Create the Rules

Inside a rulegroup the are subgroups. Per default one subgroup is created when a new rulegroup is created. Subgroups contain rules that belong together. Within one subgroup you can either connect the rules with an "and" or with an "or" condition. Multiple subgroups can also be connected using an "and" or with an "or" condition. This way you can construct the complex business logic in any shape or form.

Click on the name of the rulegroup we just created to go to the page which lists subgroups and rules.

Click on the icon to create a new rule.

A page is presented to enter the details of the rule.

According to the User Details and the Business Rules of the car rental company as listed above we will define the rules details.

Enter a name and description for the rule.

Next enter the field name of your data source that you want to check. In our case we want to check if the user has a valid drivers license. Enter "Drivers_License" for "Field to check" and for the "Type" next to it select "string" (the field in the database contains textual data).

From the drop-down list for "Check", select "is equal to". We want to check if one value (the data from the field) is equal to another value.

In "Value to check against" enter "Yes" and for the "Type" next to it select "string". We want to test the content of the field "Drivers_License" against this value.

Next to the fields "Message passed" and "Message failed" there are buttons. Click on both to auto-generate a message that will be generated once this rule runs. The messages contain placeholders which are filled with the actual values when the rule runs.

Your rule should look like this:

Click on "save" and then "close".

You are back on the page which lists the details for the rulegroup. You can see the first rule we just created.

Click to add a new subgroup. We create a new subgroup that contains two rules: one checks for the age of the user and one checks for the extra fee.

Ruleengine Sample - "Rent a Ferrari"

You are presented with a page to enter the details of the subgroup:

Enter a name and description for the subgroup. We will create two rules inside this subgroup and they will be logically connected to each other with an "or" condition: "Age is at or above 26 **or** user is willing to pay an extra fee". So set the drop-down value of "Rules are connected using" to "or".

The subgroup should look like this:

Click on "save" and then "close".

Back to the rulegroup page it now shows the first subgroup with the rule we created and the second subgroup we just created now.

Now we create two more rules: one for the age check and one for the check if the user would be willing to pay an extra fee (in case younger than 26).

Create the two additional rules similar as before. Click the relevant icon (marked in red above) to create the rules in the second subgroup. The first rule will use the check "is greater or equal than" and the other rule will use "is equal to".

Here is the first rule. The value in the database is numeric, so we use "integer" for the type.

Click on "save" and then "close".

Create the next rule:

Click on "save" and then "close".

We have created now all rules that are required according to the business logic of the car rental company:
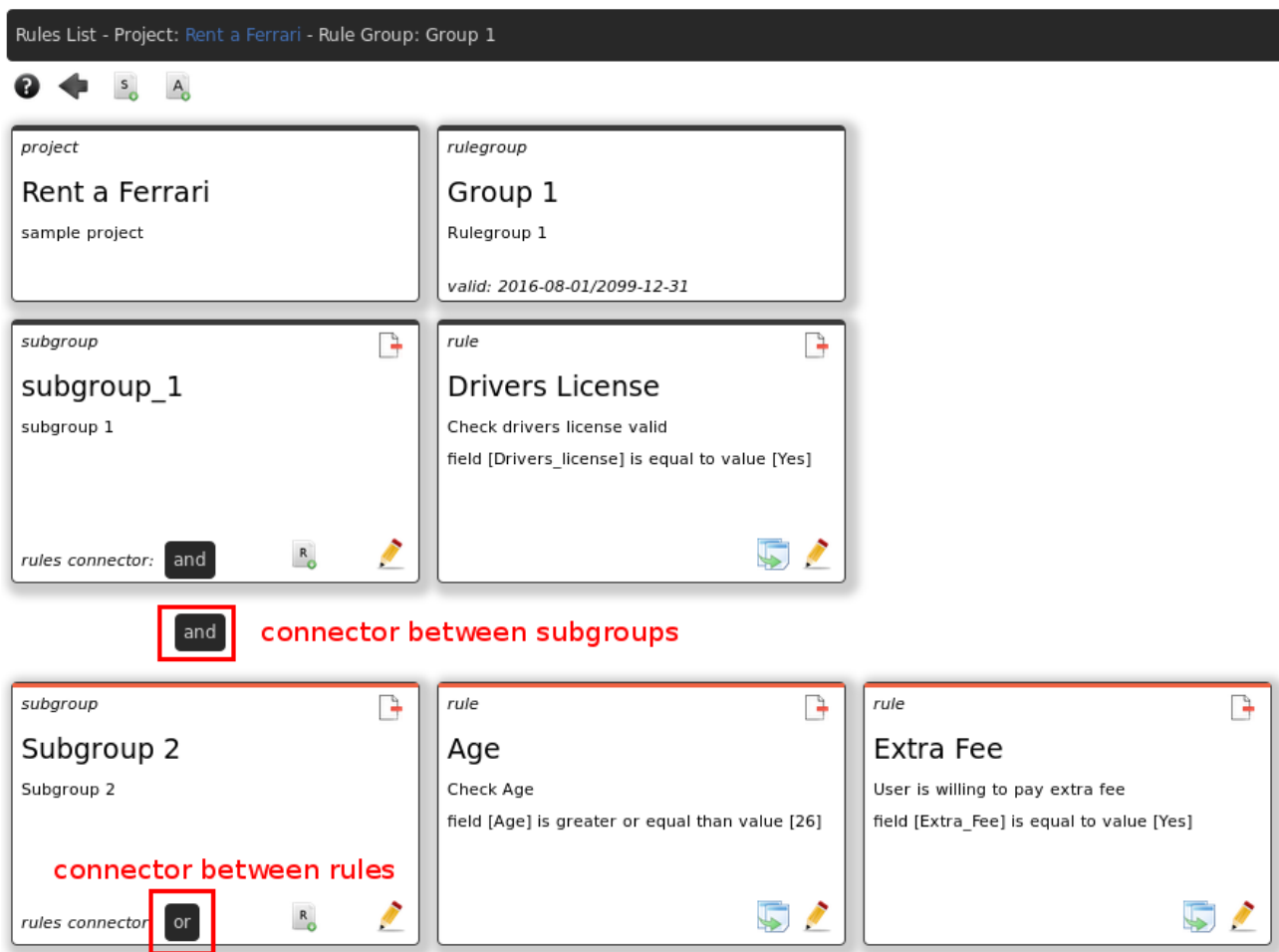
So the logic we have created goes as follows:
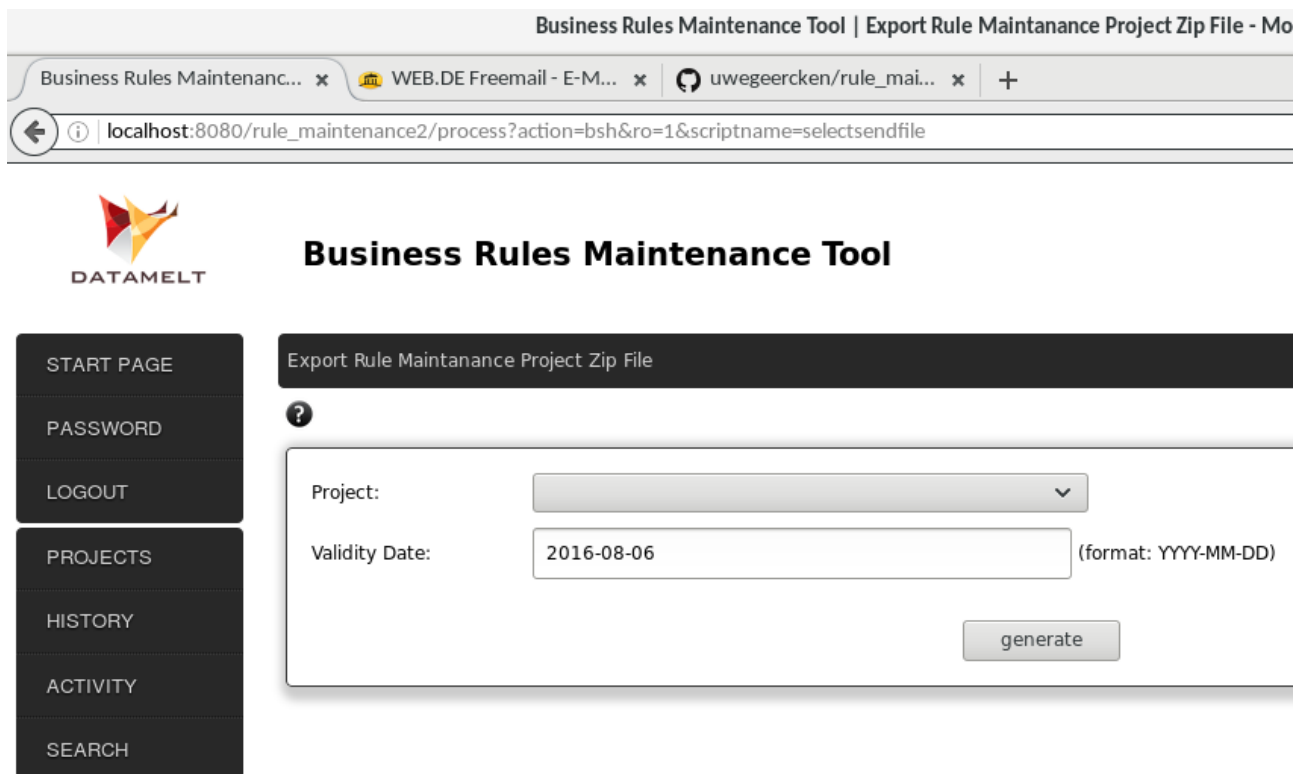
"Check if the user has a drivers license **and** check if the users age is 26+ **or** if the user is willing to pay an extra fee".

The "and" comes from the connection between the subgroups. The "or" comes from the connector between the rules of that subgroup.

Step 4 – Export the project

Our business logic is defined. The final step in the Business Rules Maintenance Tool is to export the complete project. This will create a zip file which contains all the logic we have defined before.

In the menu on the left click on "export".

Select the project you created from the drop-down list. Then select (or enter in the correct format) a validity date. Only rulegroups valid on this date will be included in the exported file. This functionality allows you to plan changes to projects – and thus changes to your business logic – ahead of time.

Ruleengine Sample - "Rent a Ferrari"

Finally click on "generate" to create the project zip file. The Web Browser will ask you where in the filesystem you want to store the file.

When the rule engine runs

The rule engine will check the logic of the rulegroup against the data. It determines that the user has a valid drivers license – the rule passes the check. It then determines that the user is **not** at or above the age of 26 – the rule fails. Next it determines that the user is willing to pay an extra fee. This is the "or" connection between the rule for the age and the rule for the extra fee.

The overall logic is:

> "rule drivers license passes and (rule age passes or rule extra fee passes)"

As the user is below 26 but is willing to pay an extra fee, the complete logic as such passes the business requirements – the whole rulegroup passes and this means that Maria will be able to rent the Ferrari.

What now?

What can you now do with the file you created?

You can run it with the Java Rule Engine "JaRE". This means that you have to download the rule engine from the GitHub address shown at the beginning. You will have to write a Java class that reads the data from the database and provides it to the rule engine. You will have to indicate to the ruleengine to use the project zip file you created. And then you can run the ruleengine and retrieve the results.

Using this approach you could also embed the ruleengine in your Java project – a regular application or a web application. Or you could use it from Java related script languages such as Groovy or Beanshell.

Alternatively you can use the Pentaho PDI Tool. PDI (also called "Kettle") is an ETL tool to create **E**xtract-**T**ransform-**L**oad processes. There is a plugin available that allows to connect the Pentaho ETL (transformation) to the rule engine. So you can read the data from the database, pipe it through the rule engine and display or output the results. No hand coding is required for this alternative!

And why?

When you define your rules outside, separately from your IT code or processes then this allows for a clear division of responsibilities: IT users maintain IT code/processes and the Business users manage the business rules. The IT code is cleaner and this is good for quality. Each expert works in his domain of responsibility.

If the business logic is not mixed with the IT code or processes, then the IT code will be easier to maintain and changes are done in a more agile way without affecting business logic.

Extending or changing the business logic will not require to change the IT logic and having a tool to maintain the business logic is more transparent for the business user.

Final words

I hope this is a good example to start with. The ruleengine and the Business Rules Maintenance Tool, as well as the Pentaho PDI plugin have proven to work in a production scenario.

All software is free to use and open source at **NO** cost. It can be downloaded from the given GitHub address.

Please send your feedback. This helps to detect issues and to enhance the tools.

Contact:

Uwe Geercken – uwe.geercken@web.de