Business Labs « Drools »

Pre-requisites:

- Good internet connection
- JDK11+
- Git
- VSCode and Red Hat provided DMN Editor extension
- Maven

Solutions on Github
https://github.com/dthibau/drools-solutions

Table of contents

Lab 1 : Set up the IDE

1.1 Archetype Maven

With the command line, create a Maven project :

```
$ mvn archetype:generate -DgroupId=org.formation -DartifactId=tp0-maven -
DarchetypeArtifactId=kie-drools-archetype -DarchetypeVersion=7.73.0.Final
-DarchetypeGroupId=org.kie -DinteractiveMode=false
```

Then build the package with :

```
$ cd tp0-maven
$ mvn package
```

Observe the goals performed by the kie-plugin

Import the maven project in Eclipse

Optional: Eclipse m2Eclipse Maven plugin doesn't know how to handle kie Maven plugin; this leads to a red cross or warnings
If you want to remove this red cross ; add the following lines in the ***pom.xml*** in th
e<pluginManagement> element

```
<plugins>
  <plugin>
    <groupId>org.eclipse.m2e</groupId>
    <artifactId>lifecycle-mapping</artifactId>
    <version>1.0.0</version>
    <configuration>
      <lifecycleMappingMetadata>
        <pluginExecutions>
          <pluginExecution>
            <pluginExecutionFilter>
            <groupId>org.kie</groupId>
            <artifactId>kie-maven-plugin</artifactId>
            <version>${drools-version}</version>
            <goals>
              <goal>build</goal>
            </goals>
          </pluginExecutionFilter>
          <action>
            <ignore />
          </action>
        </pluginExecution>
      </pluginExecutions>
      </lifecycleMappingMetadata>
    </configuration>
  </plugin>
</plugins>
</pluginManagement>
```

View and run the test class

Lab 2 : Stateless and stateful sessions

2.1 Stateless Session

Import the provided Maven project

Implementing an Helper class
The ***org.formation.helper.RuleRunner*** class is a utility that should allow to start a stateless or stateful session.

Complete the provided source file

Interaction with a stateless session
The problem is to write a rules file to calculate the price of car insurance.

The base price is calculated as follows
- If the driver is a young driver (between 18 and 25 years old), the basic price is 500€
- If the driver is a confirmed driver (> 25 years old), the basic price is 300€

A discount is made based on the seniority of the customer:
- If the client has 5 years of seniority, he is granted a 10% discount
- If the customer has 10 years of seniority, he is granted a 20% discount

Finally, a penalty of 5% is applied per number of incidents the driver has had.

Implement the rule file and test it with the class org.formation.drools.test.AssuranceTest

2.2 Stateful Session

This part takes the example of the course material

The rules to implement are:
- As soon as a fire breaks out in a room, an alarm must be triggered. If a sprinkler exists for this room, the sprinkler must activate.
- If there is no fire in any of the rooms, a message indicating that all is well must be displayed

Work to do

Import or Opening the project :
BusinessLabs_Data\2_Sessions\2.2_Stateful

Write the previously described rules in ***src/main/resources/org/formation/rules/Fire.drl***

Test class
Use ***org.formation.test.TestFire*** to validate your rules

Lab 3 : Conflicts, ordering rules, type declaration, timers...

Problem
This lab continues the stateful part of the previous lab

We want to complete the previous lab by adding the following rules

- The triggering of the alarm must be delayed by 5s in case the fire alert is a false alarm
- When 2 sprinklers are activated, we would like to go into *panic mode* and activate all available sprinklers
- Panic mode should display a general evacuation message
- We would like to display a message when all Sprinklers are activated
- As long as the alarm is in progress, we would like to periodically send an SMS to the building manager

Rule's organization and elements of solution

Rules are partitioned into several groups:

- The rules in *normal mode*: you activate as many Sprinklers as fires
- The rules in *panic mode* : we force the activation of all the Sprinklers
- The rules displaying the status of the building (Everything is fine or all the Sprinklers are on
- The rules managing the alarm (Recurrent alarm triggering and notification)

The first two groups are exclusive, only 1 type of rule applies depending on the mode.

They will therefore be implemented with the attribute ***agenda-group***

Rules based on the number of fires will determine the mode. They must be applied first

The mode could be a type declared type in the DRL file:

```
//declare any global variables here
declare PanicMode
	on : boolean
end
```

A rule guarantees that there is always a singleton fact of *PanicMod* in the working memory:

```
rule "always panic object"
salience 100
	when
		not PanicMode()
	then
		insert(new PanicMode(false));
		channels["console-channel"].send("Immeuble initialisé");
end
```

2 other rules will determine the mode of the building based on the number of fire present, these rules will be responsible for activating the appropriate *agenda group*.

Rules will be triggered by ***fireUntilHalt()*** to use the timers.

The new class ***RuleRunner*** and test classes are provided.

Lab4 : Decision tables and templates

4.1 Decision table

We take again the stateless part of Lab1, we implement the rules via a decision table .

Use the Eclipse wizard to start with a pre-filled spreadsheet
File → New → Other → Drools → Rules Resource

A sample is also provided in :
BusinessLabData/4.1_DecisionTable/sample.xls

4.2 Templates

In the 2nd part of this lab, we will generate a DRL file from a template and a data collection (stored in a Map)

Open a new project

Retrieve the provided sources, look at :
`src/main/java/org/formation/test/TestAssuranceTemplate`

and the method

`_generateFromTemplate()`

Complete the template ***age.drt*** which computes the basic price

Run the test and look at the genetrated .drl files

Lab5 : DMN

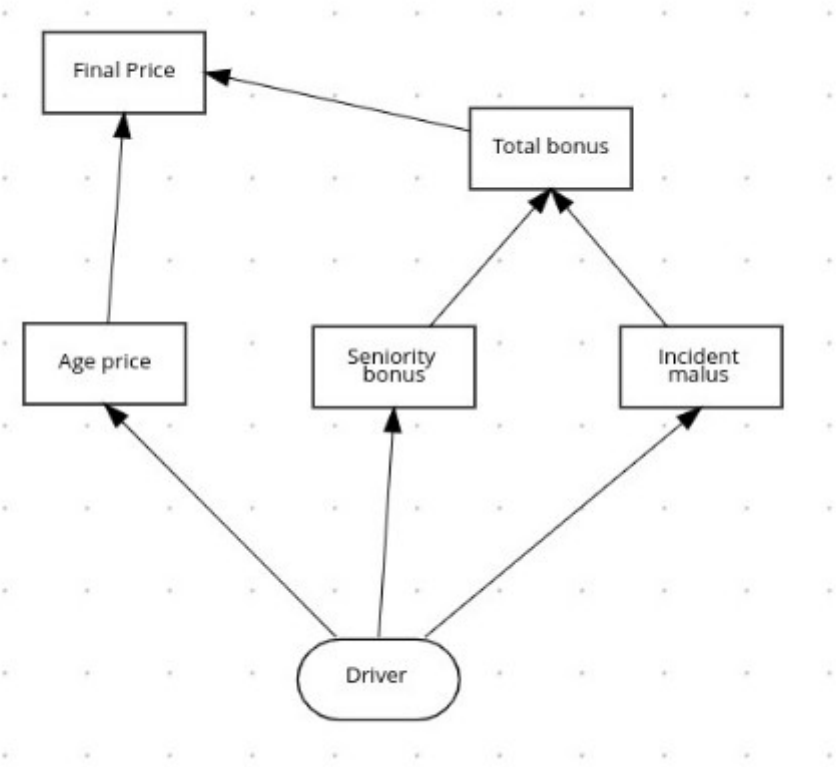
Retreive the provided project *5_DMN*

Create the folder ***dmn*** under ***src/main/resources***

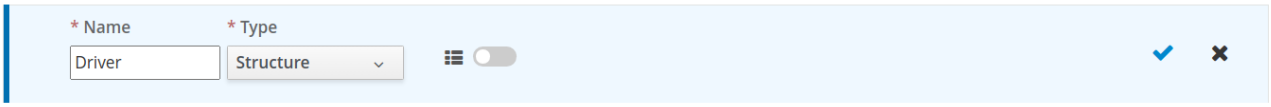
Create inside a file named ***insurance.dmn*** and edit it with VSCode

The objective of this lab is to implement the stateless rules of Lab2 (Insurance car price) with a DMN Model

The final result should be :

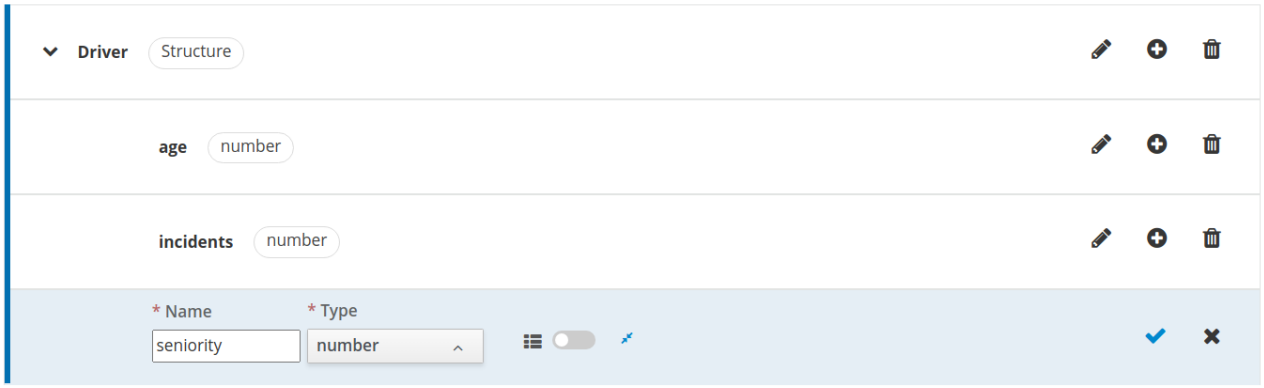


In the DMNEditor, first create a new type named ***Driver*** as follows :



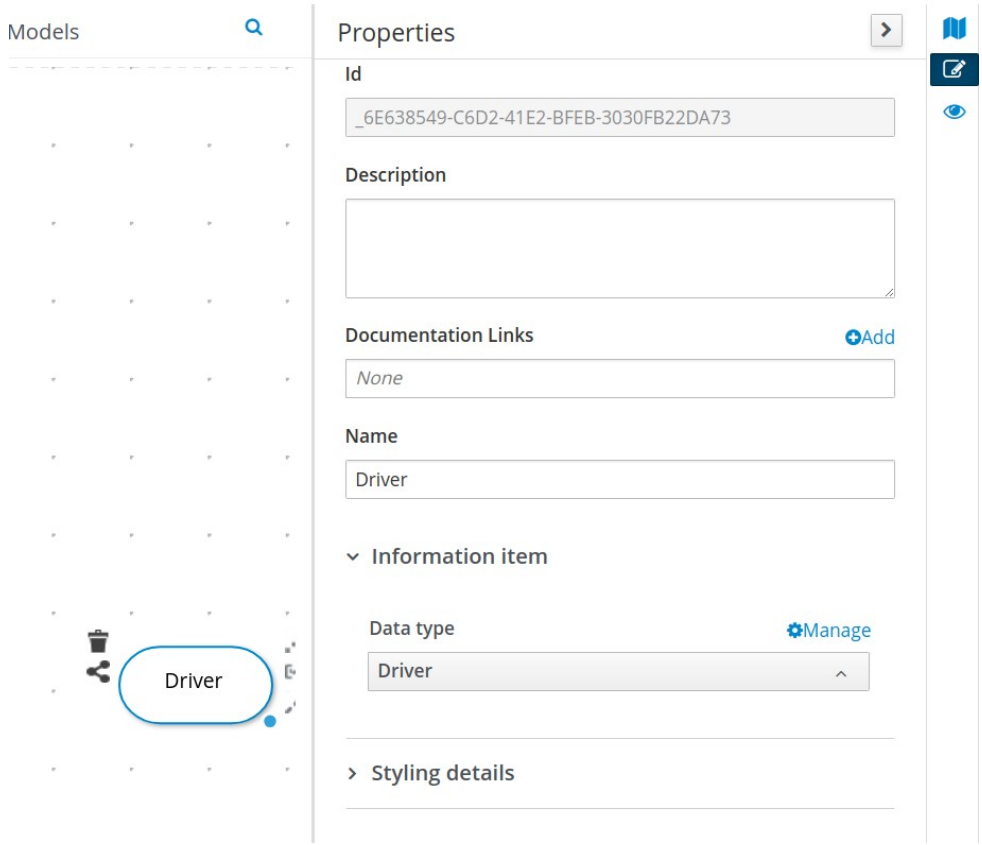
Then declare the fields of the Driver type :

- age
- incidents
- seniority



Then go to the editor tab :

Insert an *Input Data Node* in th diagram
Name it Driver
And display its properties panel.
Select as Data Type the previous custom Data Type



Then Create a DMN Decision
Name it Age Price
Edit it and choose Decision Table

And Edit the table in order to obtain this result :

U	Driver.Age *(number)*	Age price *(number)*	annotation-1
1	[18..25]	500.0	
2	> 25	300.0	

Then complete the diagram to implement all the decision nodes

Lab6 : CEP

Import the provided Maven project

Inspect the code

Write a rule that detects an ***HeartAttack***:

- If there is no heartbeat within a sliding window of 5 seconds

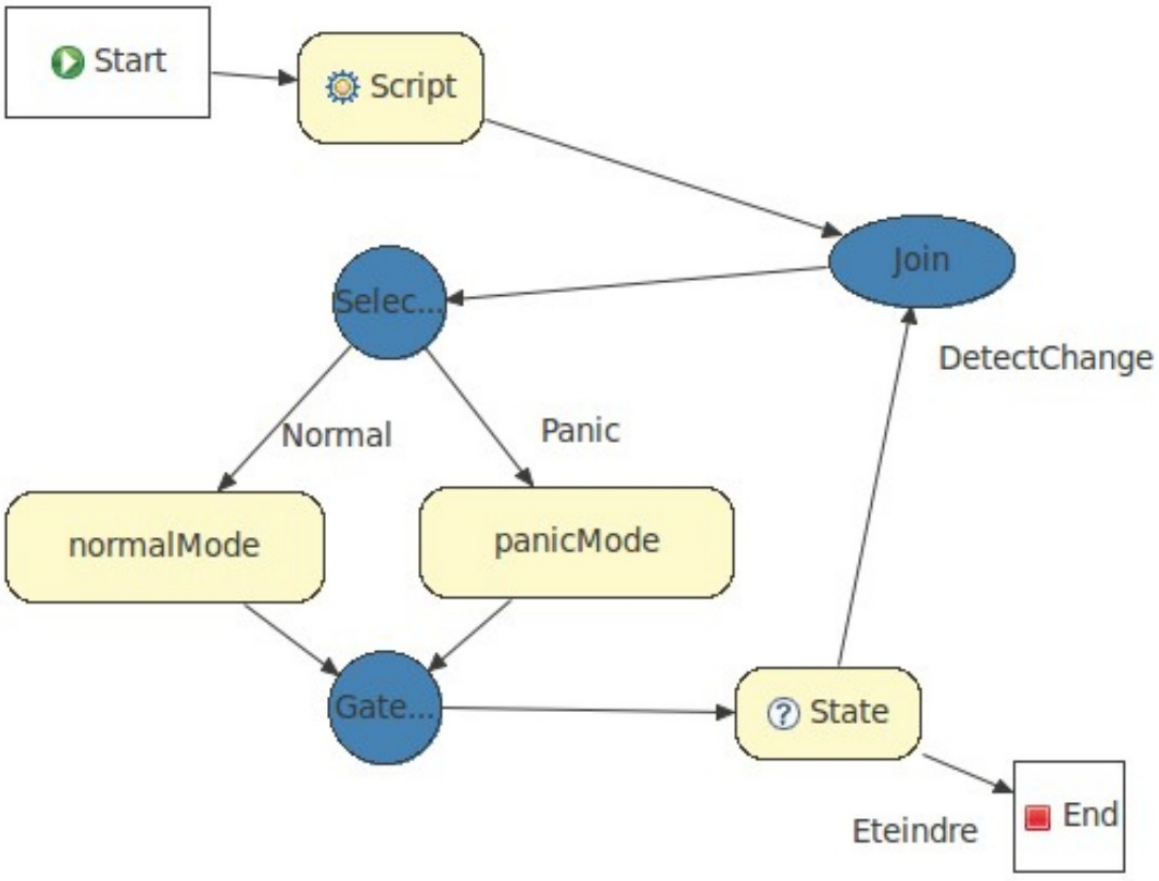
Run the tests, they should detect an event ***HeartAttackEvent***

Lab7 : Drools and processes

In this exercise, we will reproduce the Lab2 using a *jbpm process* and rule task.

The problem
The building's alarm system has two modes: the normal mode and the panic mode. Instead of implementing this behavior with group calendars we implement it via the following process:

- The alarm system when it is started performs an *Initialization Task* (Script) and then according to the facts present in the working memory is positioned on one of the nodes of type ***RuleTask*** (*normalMode* or *panicMode*).
- The entry into one of these nodes triggers the application of the corresponding rules, then the process goes into a pending state (*signal catch*).
- When receiving a *DetectChange* event, the process loops and evalutes again a the alarm mode
- When the Shut Down event is received, the alarm system is disconnected.



Implement the ***fire.bpmn*** process and the *fireMode.drl* rules to get the desired behavior.
The other necessary classes are provided.

Lab Annex : DSL

Rule file
We complete the stateless part of Lab1.
We wan to express the rules in natural langage :

```
règle "Prix de base Jeune conducteur"
si
L age du conducteur est compris entre 18 et 25
Pour une assurance
alors
Le prix de base est de 500.00
fin
```

The complete rules file is provided

Implementing the DSL file
After creating the Drools Lab3 project, use the wizard to create the DSL file, put it in the same directory as the rule file whose extension has been changed to ***.dslr***

Write the necessary associations

Editing the .dslr file
Add the expander statement to the top of the file that references the DSL file.

Rewrite the rules using the "business" syntax

The editor allows to have the business view and the technical view

Run the test class to verify your work

Lab Annex : Performance

4.1 Rete network

Prerequisites: Graphviz installation ***https://graphviz.org/***
Get the sources provided.

The ***PhreakInspector*** class generates a graph in graphviz format
Run it on your drl files.
Visualize the graph with *graphviz*:
```
$ dot -Tsvg /tmp/phreak.dot > /tmp/phreak.svg
```

4.2 Drools Metric

Reference tutorial:
https://blog.kie.org/2021/07/how-to-find-a-bottle-neck-in-your-rules-for-performance-analysis.html

Import the provided project and view:
- Dependence on drools-metric
- The JoinTest test class

Run the test class and observe the console:
- ***dumpRete*** output : Represents the Rete network
- ***dumpAssociatedRulesRete*** output : Allows you to correlate segments to rules

Then activate the provided logback file by removing the *.bak* extension, run the test

You notice that
```
[ AccumulateNode(8) ], evalCount:4950000, elapsedMicro:1277578
```
is anormal

The output of *dumpAssociatedRules()*. Show that *AccumulateNode(8)]* is associated with a rule *[Combinaison de collecte des commandes coûteuses]*
=> We must review this rule

If evalCount is abnormally large, look at the previous node's log:
```
[JoinNode(7) - [ClassObjectType class=com.sample.Order]],
evalCount:100000,
elapsedMicro:205274.
```
Accumulate is evaluated for each ***$o1/$o2*** combination produced by the JoinNode. Which is not good.

This accumulation is only used to calculate the maximum price of a customer's orders.
It should therefore only be assessed once per client.

Correct this defect, rerun the test and identify the next bottleneckMake a correction to optimize the overall time