

Drools Introduction and General User Guide

.....	v
1. Welcome	1
2. Installation and Setup (Core and IDE)	3
2.1. Installing and using	3
2.1.1. Dependencies and jars	3
2.1.2. Runtime	4
2.1.3. Installing IDE (Rule Workbench)	4
2.2. Building from source	14
2.2.1. Getting the sources	14
2.2.2. Building the sources	15
2.3. Eclipse	15
2.3.1. Importing Eclipse Projects	15
3. Drools Release Notes	23
3.1. What is New and Noteworthy in Drools 5.4.0.Beta1	23
3.1.1. Drools Expert	23
3.1.2. Guvnor	30
3.1.3. Planner	41
3.1.4. Known Issues	41
3.2. What is New and Noteworthy in Drools 5.3.0	42
3.2.1. Drools Expert	42
3.2.2. Guvnor	45
3.2.3. Drools Planner	59
3.2.4. Drools Integration	61
3.3. What is New and Noteworthy in Drools 5.2.0	62
3.3.1. Knowledge API (renamed from Drools API)	62
3.3.2. Drools Expert and Fusion	62
3.3.3. Drools and jBPM integration	69
3.3.4. Merging Drools Flow into jBPM5	70
3.3.5. Guvnor	70
3.3.6. Eclipse	81
3.3.7. Maven artifactId changes	81
3.4. What is New and Noteworthy in Drools 5.1.0	82
3.4.1. Drools API	82
3.4.2. Core	83
3.4.3. Expert	89
3.4.4. Flow	92
3.4.5. Guvnor	93
3.4.6. Eclipse	109
3.4.7. Known Issues	112
3.5. What is New and Noteworthy in Drools 5.0.0	112
3.5.1. Drools API	112
3.5.2. Drools Guvnor	115
3.5.3. Drools Expert	120
3.5.4. Drools Flow	128

3.5.5. Drools Fusion	136
3.5.6. Eclipse IDE	140
3.6. What is new in Drools 4.0	141
3.6.1. Language Expressiveness Enhancements	142
3.6.2. Core Engine Enhancements	142
3.6.3. IDE Enhancements	143
3.6.4. Business Rules Management System - BRMS	143
3.6.5. Miscellaneous Enhancements	143
3.7. Upgrade tips from Drools 3.0.x to Drools 4.0.x	143
3.7.1. API changes	143
3.7.2. Rule Language Changes	144
3.7.3. Drools Update Tool	145
3.7.4. DSL Grammars in Drools 4.0	146
3.7.5. Rule flow Update for 4.0.2	146
4. Drools compatibility matrix	149
Index	151



G

E

F

F

Chapter 1. Welcome

I've always stated that end business users struggle understanding the differences between rules and processes, and more recently rules and event processing. For them they have this problem in their mind and they just want to model it using some software. The traditional way of using two vendor offerings forces the business user to work with a process oriented or rules oriented approach which just gets in the way, often with great confusion over which tool they should be using to model which bit.

PegaSystems and Microsoft have done a great job of showing that the two can be combined and a behavioural modelling approach can be used. This allows the business user to work more naturally where the full range of approaches is available to them, without the tools getting in the way. From being process oriented to rule oriented or shades of grey in the middle - whatever suites the problem being modelled at that time.

Drools 5.0 takes this one step further by not only adding BPMN2 based workflow with Drools Flow but also adding event processing with Drools Fusion, creating a more holistic approach to software development. Where the term holistic is used for emphasizing the importance of the whole and the interdependence of its parts.

Drools 5.0 is now split into 5 modules, each with their own manual - Guvnor (BRMS/BPMS), Expert (Rules), Fusion (CEP), Flow (Process/Workflow) and Planner. Guvnor is our web based governance system, traditionally referred to in the rules world as a BRMS. We decided to move away from the BRMS term to a play on governance as it's not rules specific. Expert is the traditional rules engine. Fusion is the event processing side, it's a play on data/sensor fusion terminology. Flow is our workflow module, Kris Verlaenen leads this and has done some amazing work; he's currently moving flow to be incorporated into jBPM 5. The fifth module called Planner, authored by Geoffrey De Smet, solves allocation and scheduling type problem and while still in the early stage of development is showing a lot of promise. We hope to add Semantics for 2011, based around description logic, and that is being work on as part of the next generation Drools designs.

I've been working in the rules field now for around 7 years and I finally feel like I'm getting to grips with things and ideas are starting to gel and the real innovation is starting to happen. To me It feels like we actually know what we are doing now, compared to the past where there was a lot of wild guessing and exploration. I've been working hard on the next generation Drools Expert design document with Edson Tirelli and Davide Sottara. I invite you to read the document and get involved, <http://community.jboss.org/wiki/DroolsLanguageEnhancements>. The document takes things to the next level pushing Drools forward as a hybrid engine, not just a capable production rule system, but also melding in logic programming (prolog) with functional programming and description logic along with a host of other ideas for a more expressive and modern feeling language.

I hope you can feel the passion that my team and I have while working on Drools, and that some of it rubs off on you during your adventures.

Mark Proctor
Drools Creator and Lead



Chapter 2. Installation and Setup

(Core and IDE)

2.1. Installing and using

Drools provides an Eclipse-based IDE (which is optional), but at its core only Java 1.5 (Java SE) is required.

A simple way to get started is to download and install the Eclipse plug-in - this will also require the Eclipse GEF framework to be installed (see below, if you don't have it installed already). This will provide you with all the dependencies you need to get going: you can simply create a new rule project and everything will be done for you. Refer to the chapter on the Rule Workbench and IDE for detailed instructions on this. Installing the Eclipse plug-in is generally as simple as unzipping a file into your Eclipse plug-in directory.

Use of the Eclipse plug-in is not required. Rule files are just textual input (or spreadsheets as the case may be) and the IDE (also known as the Rule Workbench) is just a convenience. People have integrated the rule engine in many ways, there is no "one size fits all".

Alternatively, you can download the binary distribution, and include the relevant jars in your projects classpath.

2.1.1. Dependencies and jars

Drools is broken down into a few modules, some are required during rule development/compiling, and some are required at runtime. In many cases, people will simply want to include all the dependencies at runtime, and this is fine. It allows you to have the most flexibility. However, some may prefer to have their "runtime" stripped down to the bare minimum, as they will be deploying rules in binary form - this is also possible. The core runtime engine can be quite compact, and only requires a few 100 kilobytes across 3 jar files.

The following is a description of the important libraries that make up JBoss Drools

- knowledge-api.jar - this provides the interfaces and factories. It also helps clearly show what is intended as a user api and what is just an engine api.
- knowledge-internal-api.jar - this provides internal interfaces and factories.
- drools-core.jar - this is the core engine, runtime component. Contains both the RETE engine and the LEAPS engine. This is the only runtime dependency if you are pre-compiling rules (and deploying via Package or RuleBase objects).
- drools-compiler.jar - this contains the compiler/builder components to take rule source, and build executable rule bases. This is often a runtime dependency of your application, but it need not be if you are pre-compiling your rules. This depends on drools-core.

- drools-jsr94.jar - this is the JSR-94 compliant implementation, this is essentially a layer over the drools-compiler component. Note that due to the nature of the JSR-94 specification, not all features are easily exposed via this interface. In some cases, it will be easier to go direct to the Drools API, but in some environments the JSR-94 is mandated.
- drools-decisiontables.jar - this is the decision tables 'compiler' component, which uses the drools-compiler component. This supports both excel and CSV input formats.

There are quite a few other dependencies which the above components require, most of which are for the drools-compiler, drools-jsr94 or drools-decisiontables module. Some key ones to note are "POI" which provides the spreadsheet parsing ability, and "antlr" which provides the parsing for the rule language itself.

NOTE: if you are using Drools in J2EE or servlet containers and you come across classpath issues with "JDT", then you can switch to the janino compiler. Set the system property "drools.compiler": For example: -Ddrools.compiler=JANINO.

For up to date info on dependencies in a release, consult the released poms, which can be found on the maven repository.

2.1.2. Runtime

The "runtime" requirements mentioned here are if you are deploying rules as their binary form (either as KnowledgePackage objects, or KnowledgeBase objects etc). This is an optional feature that allows you to keep your runtime very light. You may use drools-compiler to produce rule packages "out of process", and then deploy them to a runtime system. This runtime system only requires drools-core.jar and knowledge-api for execution. This is an optional deployment pattern, and many people do not need to "trim" their application this much, but it is an ideal option for certain environments.

2.1.3. Installing IDE (Rule Workbench)

The rule workbench (for Eclipse) requires that you have Eclipse 3.4 or greater, as well as Eclipse GEF 3.4 or greater. You can install it either by downloading the plug-in or, or using the update site.

Another option is to use the JBoss IDE, which comes with all the plug-in requirements pre packaged, as well as a choice of other tools separate to rules. You can choose just to install rules from the "bundle" that JBoss IDE ships with.

2.1.3.1. Installing GEF (a required dependency)

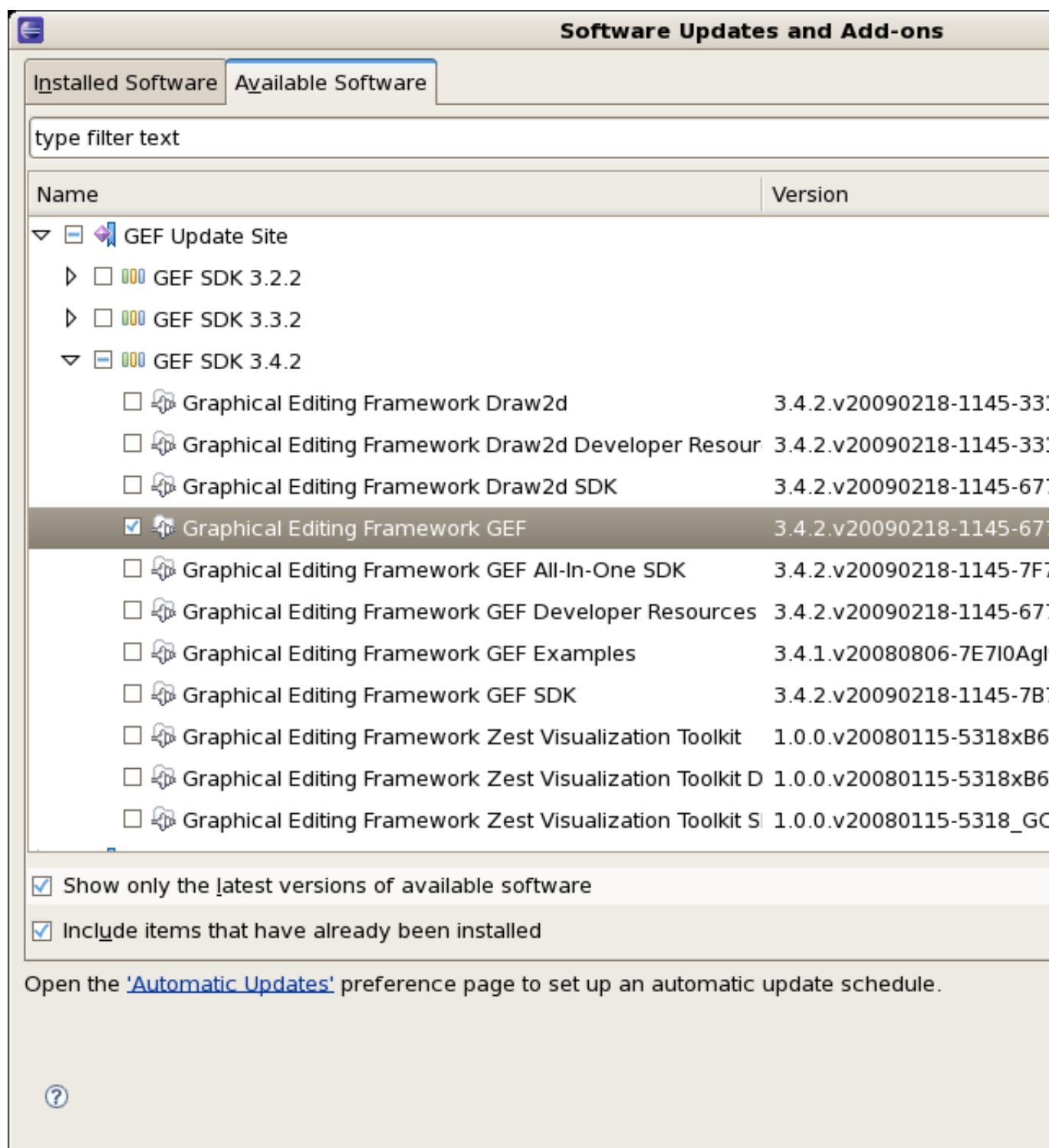
GEF is the Eclipse Graphical Editing Framework, which is used for graph viewing components in the plug-in.

If you don't have GEF installed, you can install it using the built in update mechanism (or downloading GEF from the Eclipse.org website not recommended). JBoss IDE has GEF already, as do many other "distributions" of Eclipse, so this step may be redundant for some people.

Open the Help->Software updates...->Available Software->Add Site... from the help menu.
Location is:

```
http://download.eclipse.org/tools/gef/updates/releases/
```

Next you choose the GEF plug-in:



Press next, and agree to install the plug-in (an Eclipse restart may be required). Once this is completed, then you can continue on installing the rules plug-in.

2.1.3.2. Installing GEF from zip file

To install from the zip file, download and unzip the file. Inside the zip you will see a plug-in directory, and the plug-in jar itself. You place the plug-in jar into your Eclipse applications plug-in directory, and restart Eclipse.

2.1.3.3. Installing Drools plug-in from zip file

Download the Drools Eclipse IDE plugin from the link below. Unzip the downloaded file in your main eclipse folder (do not just copy the file there, extract it so that the feature and plugin jars end up in the features and plugin directory of eclipse) and (re)start Eclipse.

<http://www.jboss.org/drools/downloads.html>

To check that the installation was successful, try opening the Drools perspective: Click the 'Open Perspective' button in the top right corner of your Eclipse window, select 'Other...' and pick the Drools perspective. If you cannot find the Drools perspective as one of the possible perspectives, the installation probably was unsuccessful. Check whether you executed each of the required steps correctly: Do you have the right version of Eclipse (3.4.x)? Do you have Eclipse GEF installed (check whether the `org.eclipse.gef_3.4.*.jar` exists in the plugins directory in your eclipse root folder)? Did you extract the Drools Eclipse plugin correctly (check whether the `org.drools.eclipse_*.jar` exists in the plugins directory in your eclipse root folder)? If you cannot find the problem, try contacting us (e.g. on irc or on the user mailing list), more info can be found on our homepage here:

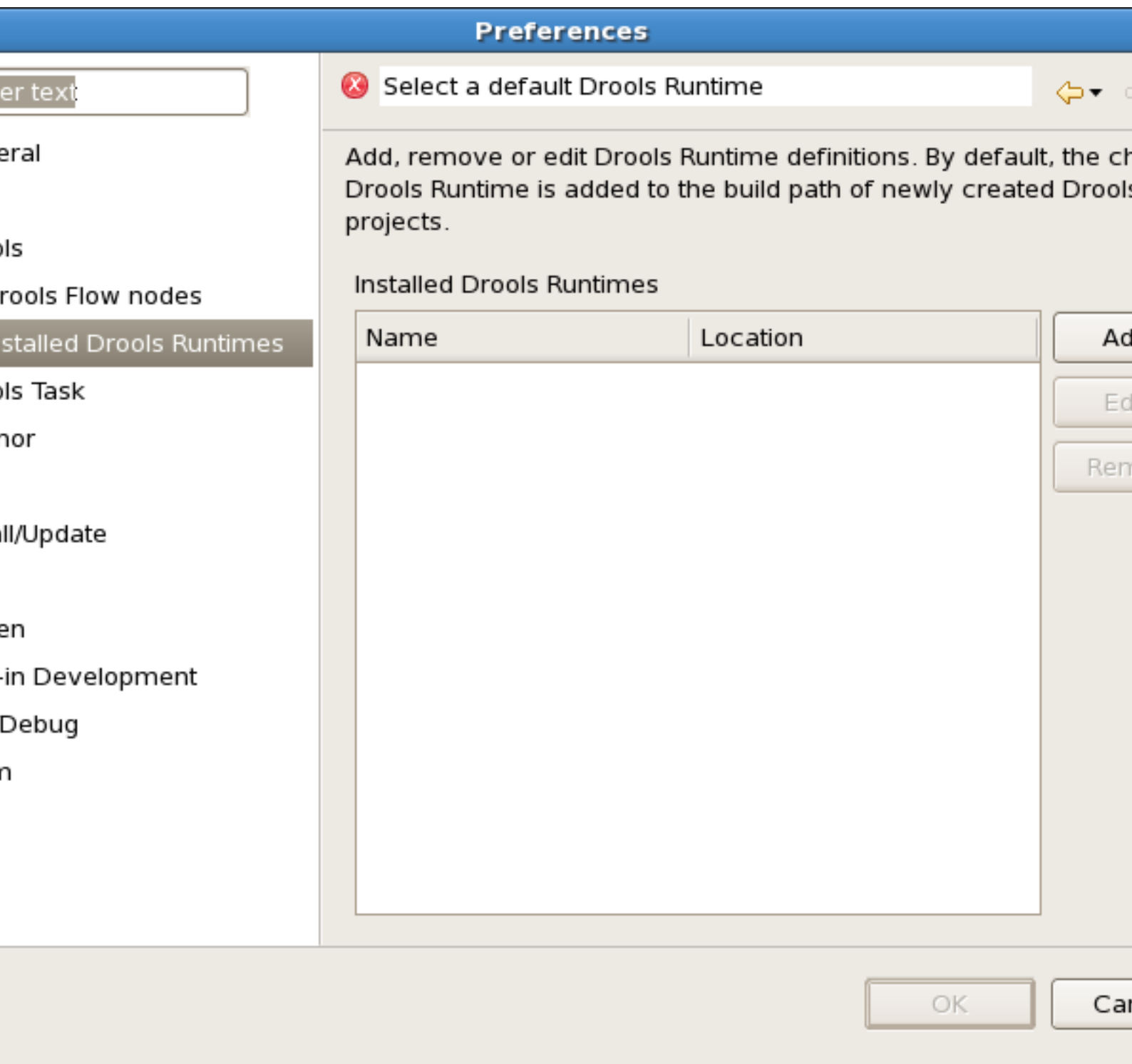
<http://www.jboss.org/drools/>

2.1.3.4. Drools Runtimes

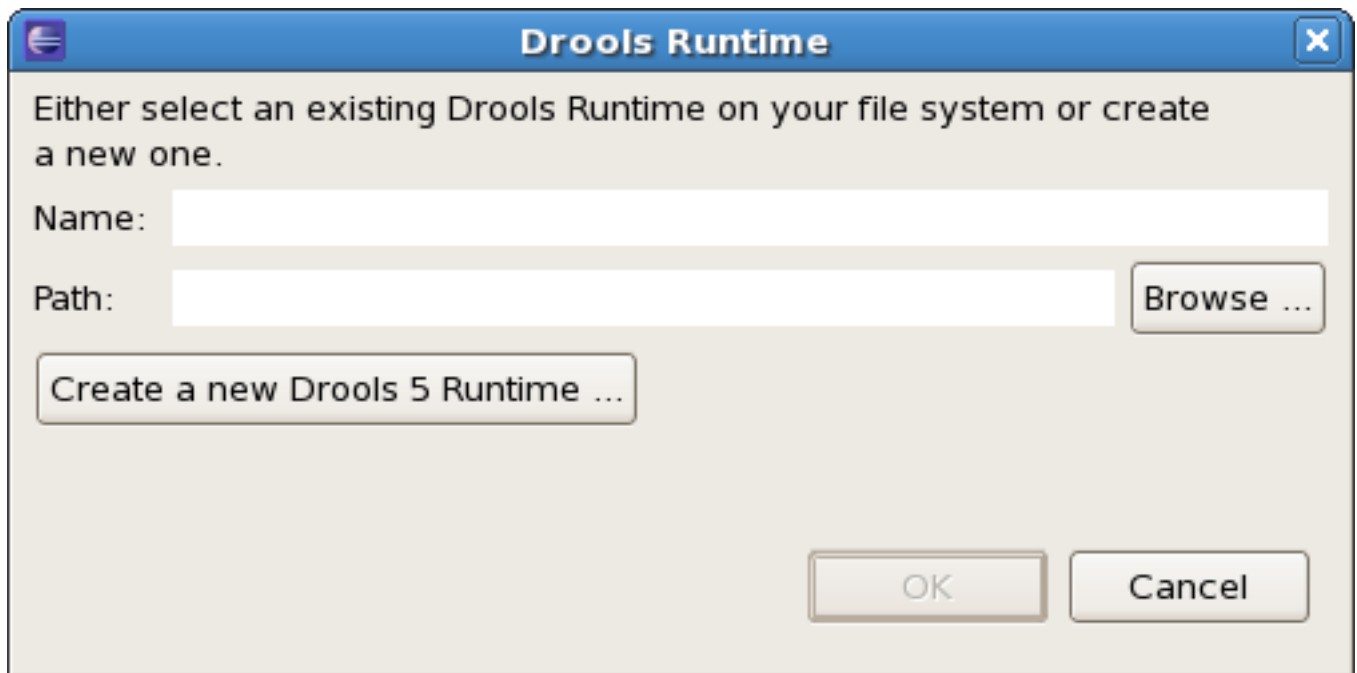
A Drools runtime is a collection of jars on your file system that represent one specific release of the Drools project jars. To create a runtime, you must point the IDE to the release of your choice. If you want to create a new runtime based on the latest Drools project jars included in the plugin itself, you can also easily do that. You are required to specify a default Drools runtime for your Eclipse workspace, but each individual project can override the default and select the appropriate runtime for that project specifically.

2.1.3.4.1. Defining a Drools runtime

You are required to define one or more Drools runtimes using the Eclipse preferences view. To open up your preferences, in the menu Window select the Preferences menu item. A new preferences dialog should show all your preferences. On the left side of this dialog, under the Drools category, select "Installed Drools runtimes". The panel on the right should then show the currently defined Drools runtimes. If you have not yet defined any runtimes, it should look something like the figure below.

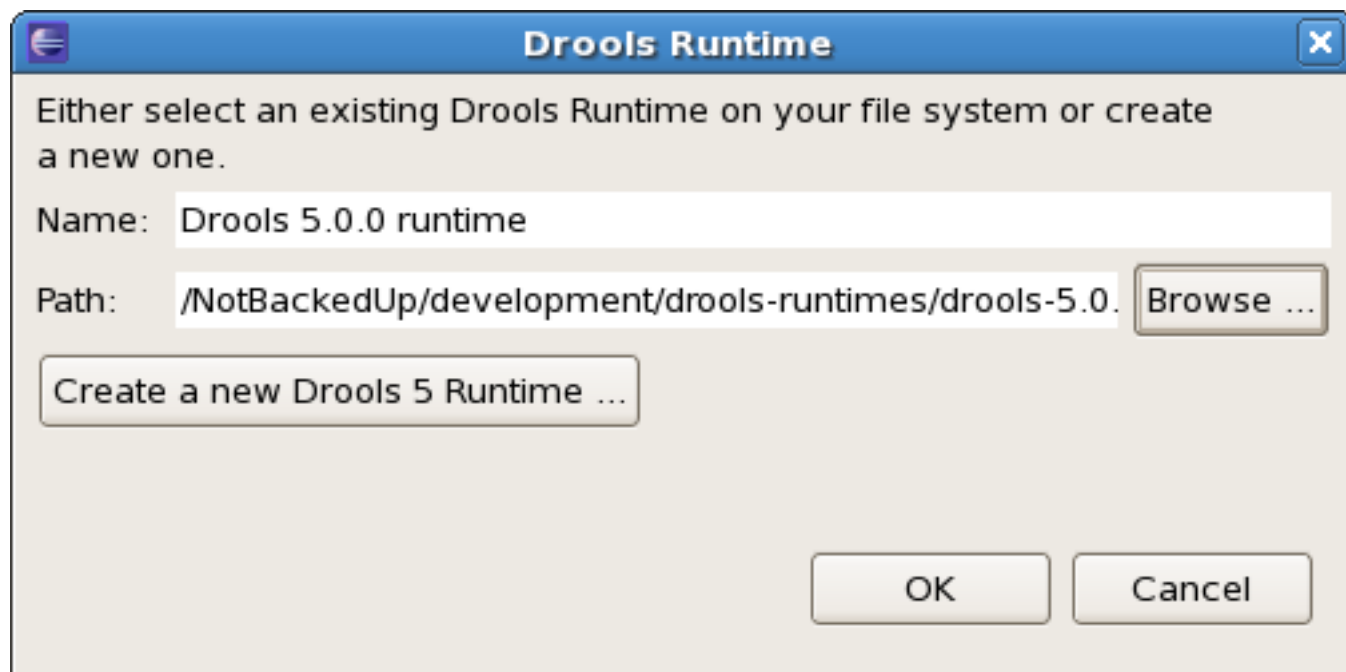


To define a new Drools runtime, click on the add button. A dialog as shown below should pop up, requiring the name for your runtime and the location on your file system where it can be found.

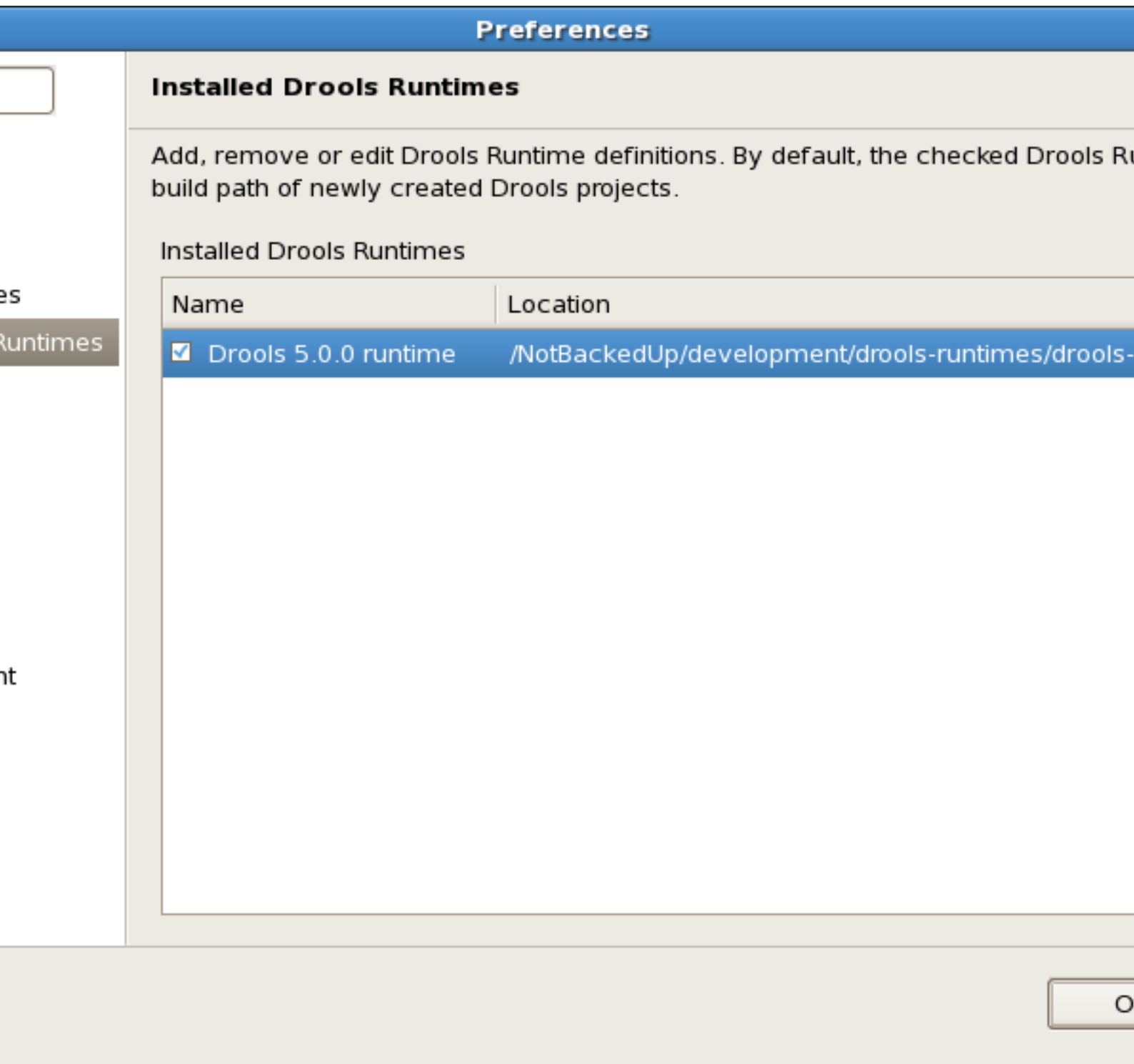


In general, you have two options:

1. If you simply want to use the default jars as included in the Drools Eclipse plugin, you can create a new Drools runtime automatically by clicking the "Create a new Drools 5 runtime ..." button. A file browser will show up, asking you to select the folder on your file system where you want this runtime to be created. The plugin will then automatically copy all required dependencies to the specified folder. After selecting this folder, the dialog should look like the figure shown below.
2. If you want to use one specific release of the Drools project, you should create a folder on your file system that contains all the necessary Drools libraries and dependencies. Instead of creating a new Drools runtime as explained above, give your runtime a name and select the location of this folder containing all the required jars.



After clicking the OK button, the runtime should show up in your table of installed Drools runtimes, as shown below. Click on checkbox in front of the newly created runtime to make it the default Drools runtime. The default Drools runtime will be used as the runtime of all your Drools project that have not selected a project-specific runtime.



You can add as many Drools runtimes as you need. For example, the screenshot below shows a configuration where three runtimes have been defined: a Drools 4.0.7 runtime, a Drools 5.0.0 runtime and a Drools 5.0.0.SNAPSHOT runtime. The Drools 5.0.0 runtime is selected as the default one.

Preferences

Installed Drools Runtimes

Add, remove or edit Drools Runtime definitions. By default, the checked Drools Runtime is used by newly created Drools projects.

Installed Drools Runtimes

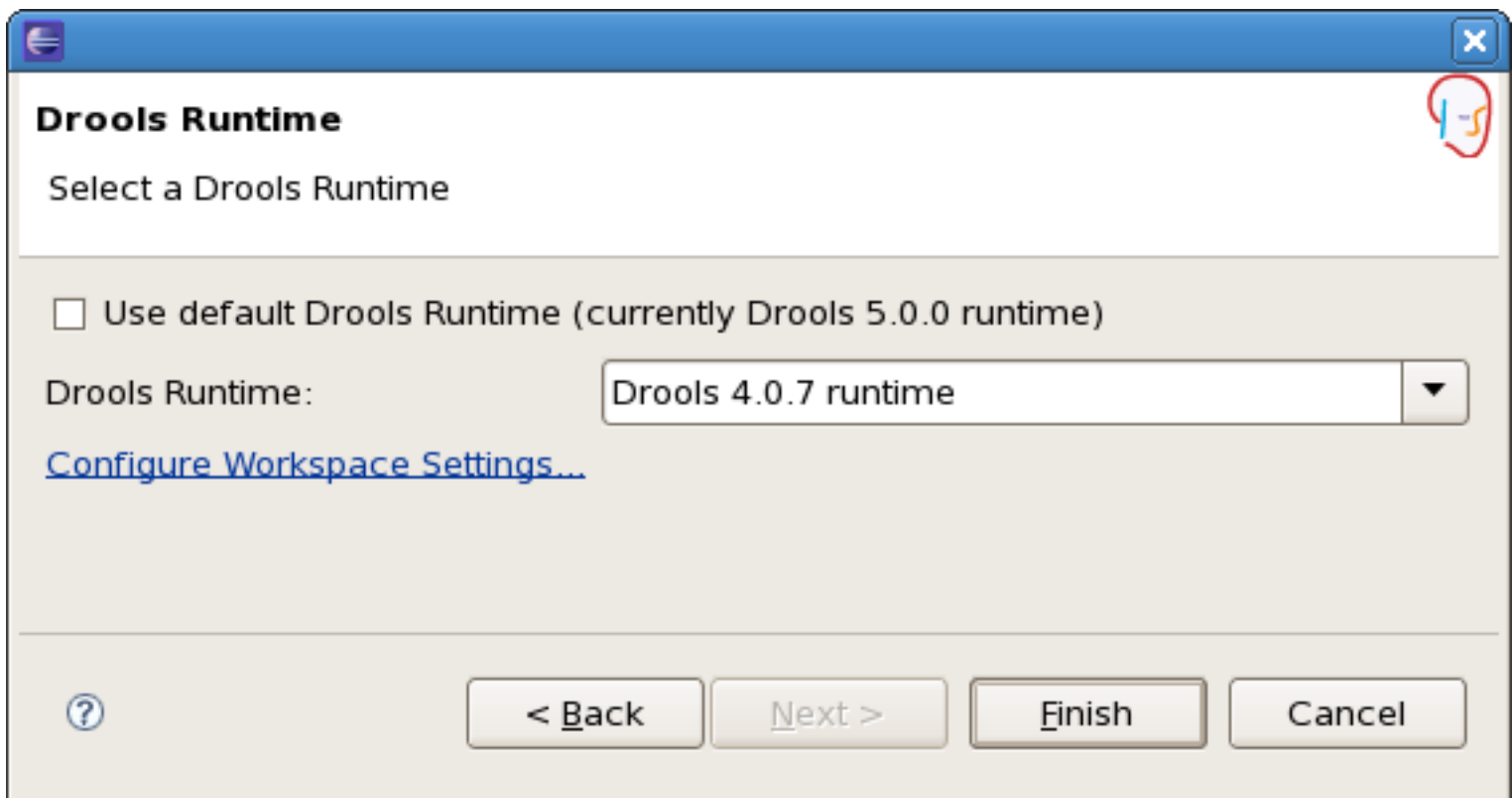
Name	Location
<input checked="" type="checkbox"/> Drools 5.0.0 runtime	/NotBackedUp/development/drools-runtimes/drools-5.0.0
<input type="checkbox"/> Drools 4.0.7 runtime	/NotBackedUp/development/drools-runtimes/drools-4.0.7
<input type="checkbox"/> Drools 5.0.0.SNAPSHOT	/NotBackedUp/development/drools-runtimes/drools-5.0.0.S

Note that you will need to restart Eclipse if you changed the default runtime and you want to make sure that all the projects that are using the default runtime update their classpath accordingly.

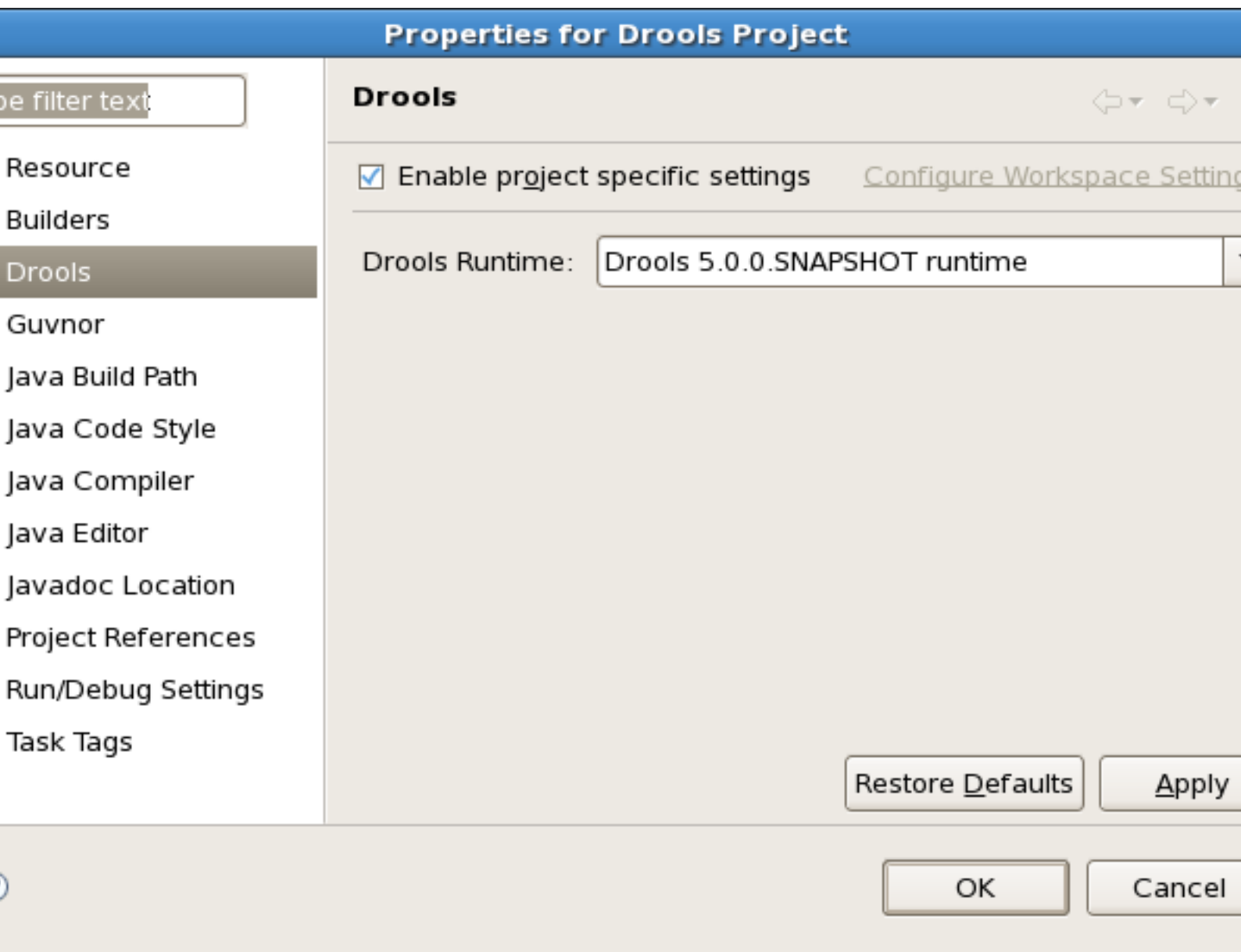
2.1.3.4.2. Selecting a runtime for your Drools project

Whenever you create a Drools project (using the New Drools Project wizard or by converting an existing Java project to a Drools project using the "Convert to Drools Project" action that is shown when you are in the Drools perspective and you right-click an existing Java project), the plugin will automatically add all the required jars to the classpath of your project.

When creating a new Drools project, the plugin will automatically use the default Drools runtime for that project, unless you specify a project-specific one. You can do this in the final step of the New Drools Project wizard, as shown below, by deselecting the "Use default Drools runtime" checkbox and selecting the appropriate runtime in the drop-down box. If you click the "Configure workspace settings ..." link, the workspace preferences showing the currently installed Drools runtimes will be opened, so you can add new runtimes there.



You can change the runtime of a Drools project at any time by opening the project properties (right-click the project and select Properties) and selecting the Drools category, as shown below. Check the "Enable project specific settings" checkbox and select the appropriate runtime from the drop-down box. If you click the "Configure workspace settings ..." link, the workspace preferences showing the currently installed Drools runtimes will be opened, so you can add new runtimes there. If you deselect the "Enable project specific settings" checkbox, it will use the default runtime as defined in your global preferences.



2.2. Building from source

2.2.1. Getting the sources

The source code of each maven artifact is available in the JBoss maven repository as a source jar. The same source jars are also included in the download zips. However, if you want to build from source, it's highly recommended to get our sources from our source control.

Drools and jBPM use [Git](http://git-scm.com/) [http://git-scm.com/] for source control. The blessed git repositories are hosted on [Github](https://github.com) [https://github.com]:

- <https://github.com/droolsjbpm>

Git allows you to fork our code, independently make personal changes on it, yet still merge in our latest changes regularly and optionally share your changes with us. To learn more about git, read the free book [Git Pro](http://progit.org/book/) [http://progit.org/book/].

2.2.2. Building the sources

In essence, building from source is very easy, for example if you want to build the *guvnor* project:

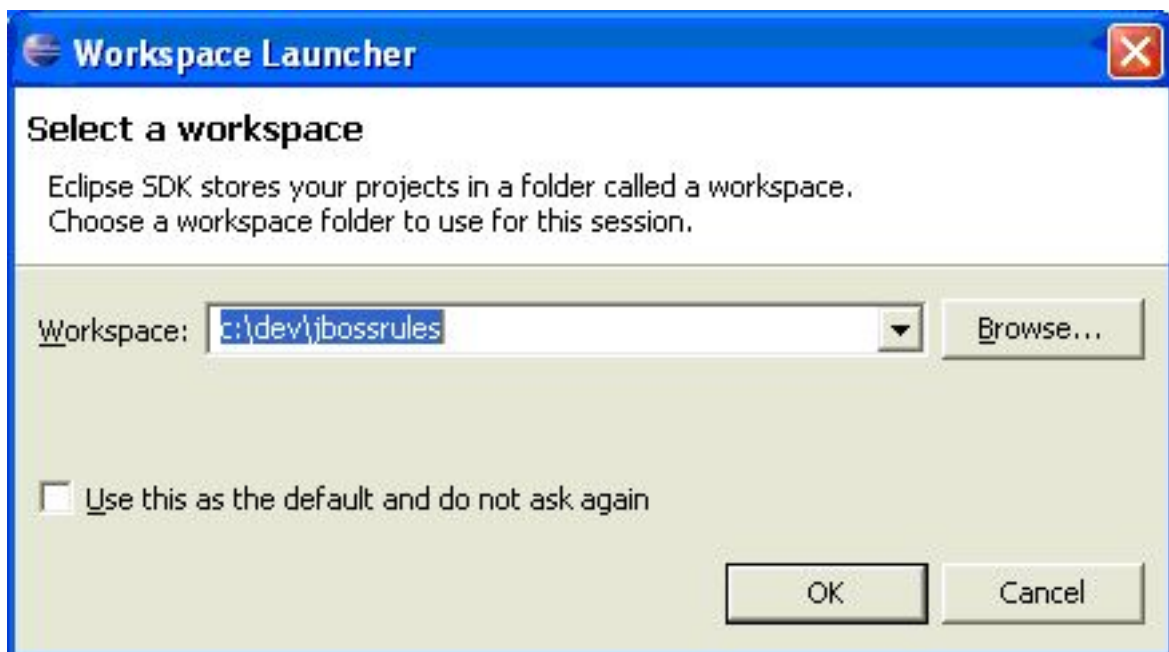
```
$ git clone git@github.com:droolsjbpm/guvnor.git
...
$ cd guvnor
$ mvn clean install -DskipTests -Dfull
...
```

However, *there are a lot potential pitfalls*, so if you're serious about building from source and possibly contributing to the project, **follow the instructions in the README file in droolsjbpm-build-bootstrap** [<https://github.com/droolsjbpm/droolsjbpm-build-bootstrap/blob/master/README.md>].

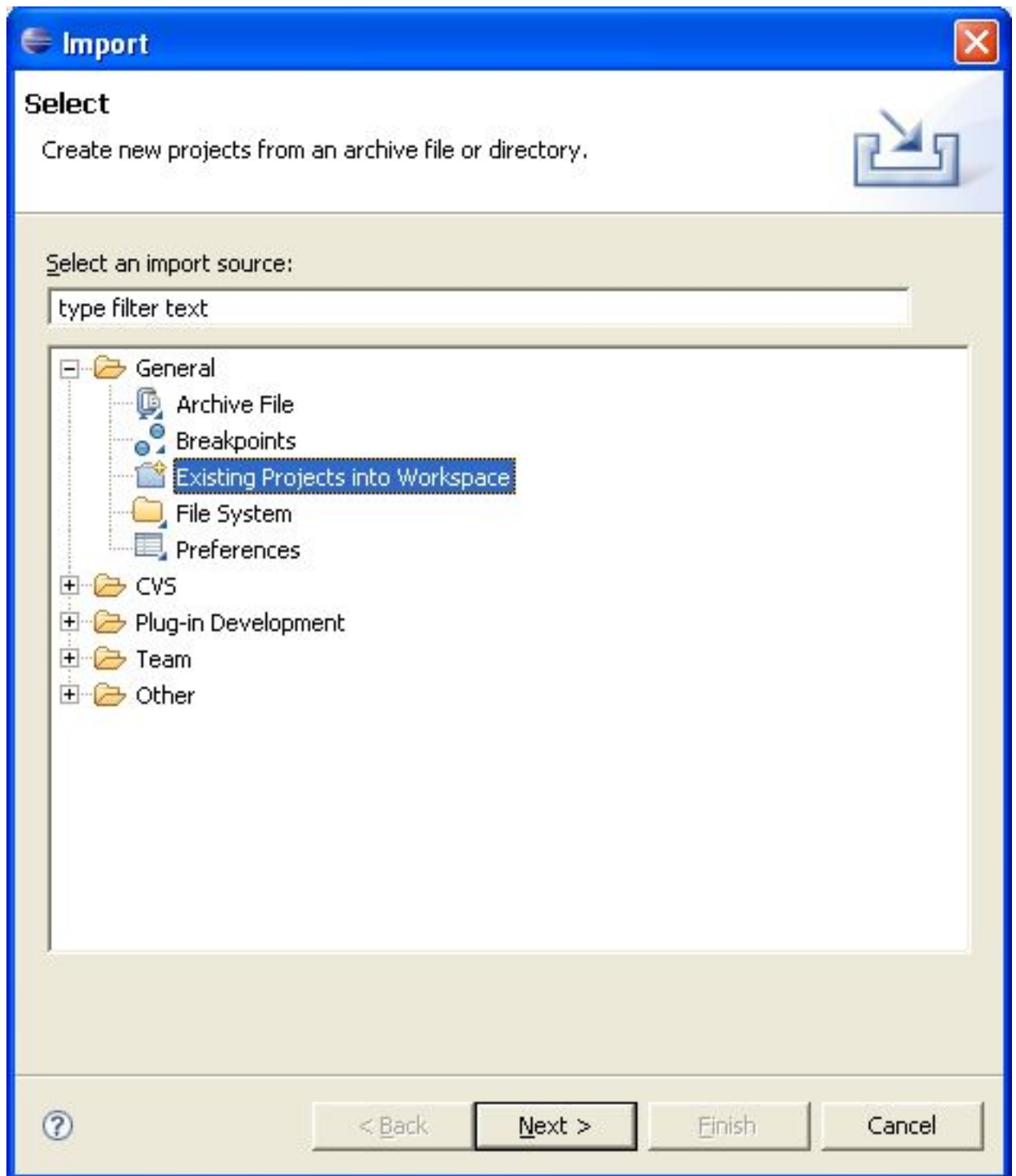
2.3. Eclipse

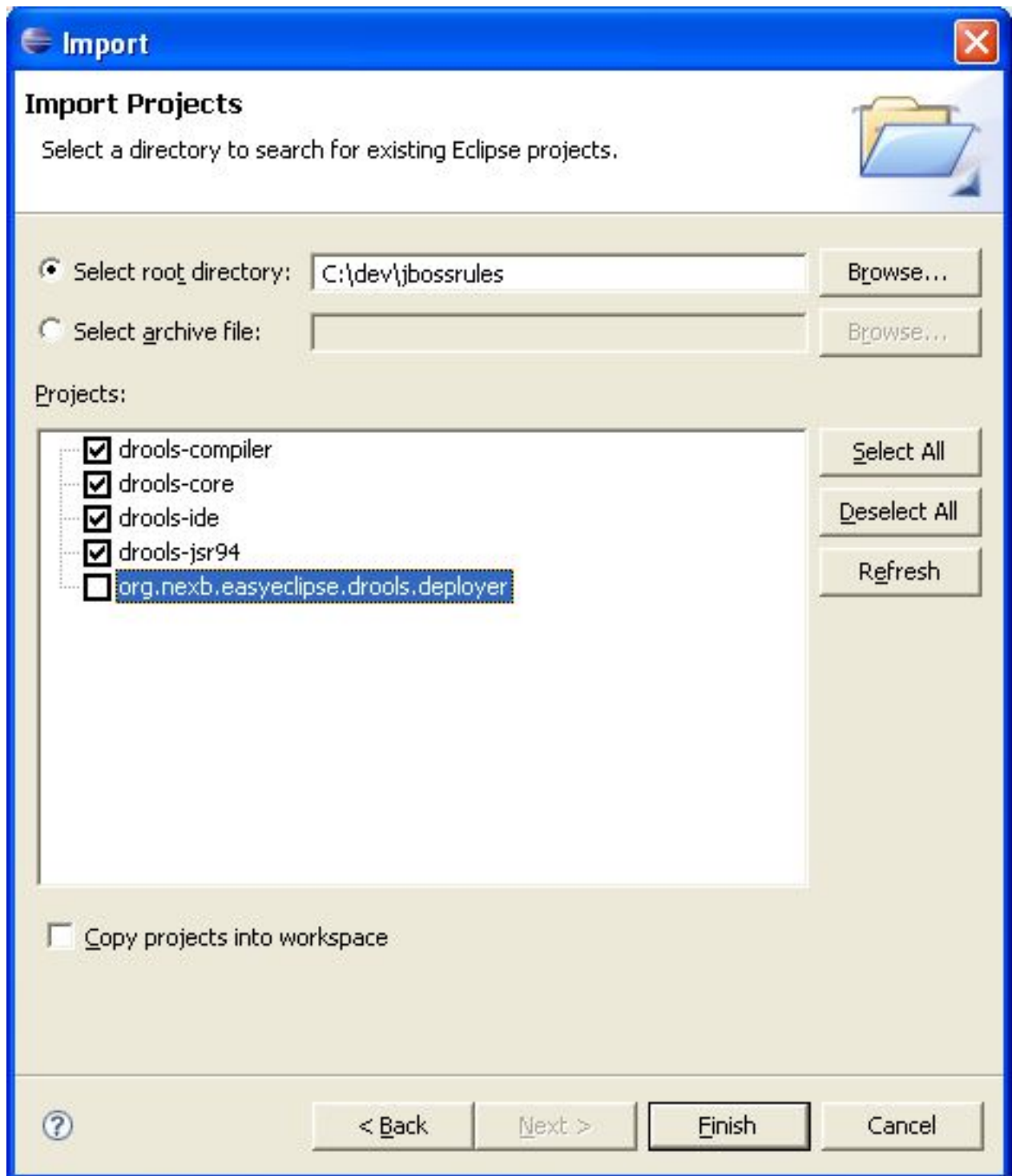
2.3.1. Importing Eclipse Projects

With the Eclipse project files generated they can now be imported into Eclipse. When starting Eclipse open the workspace in the root of your subversion checkout.

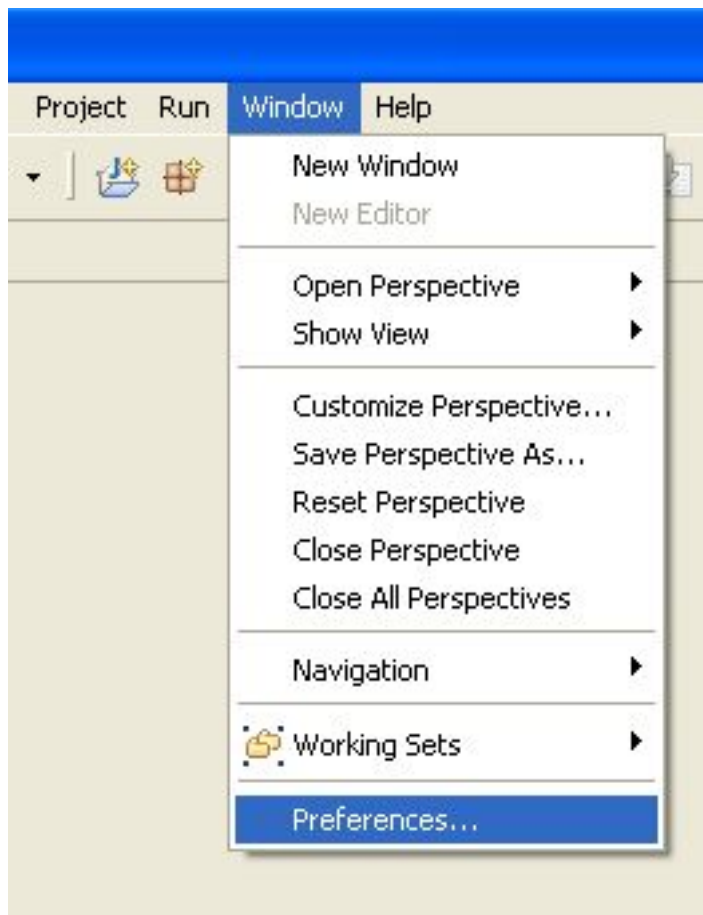


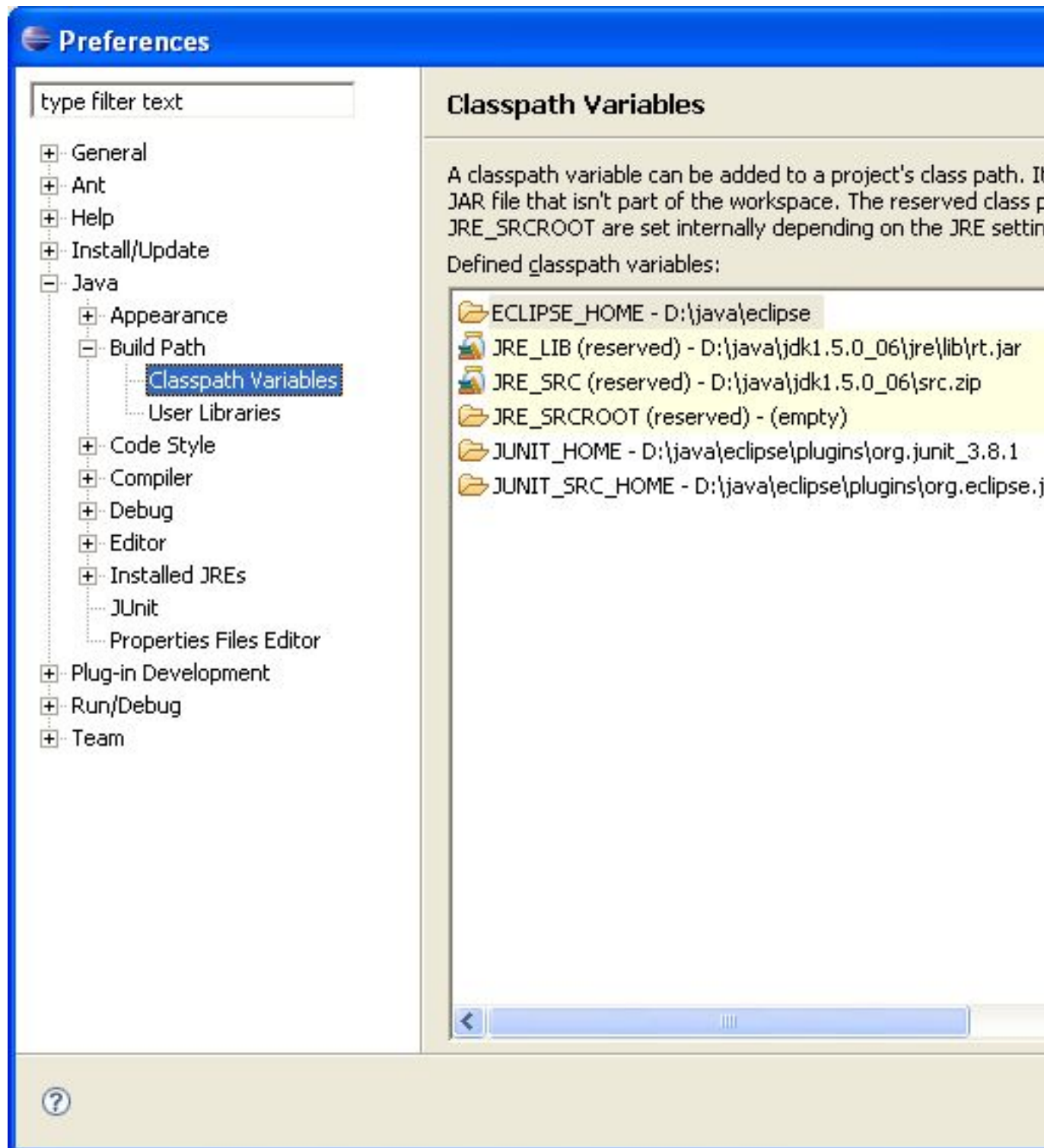


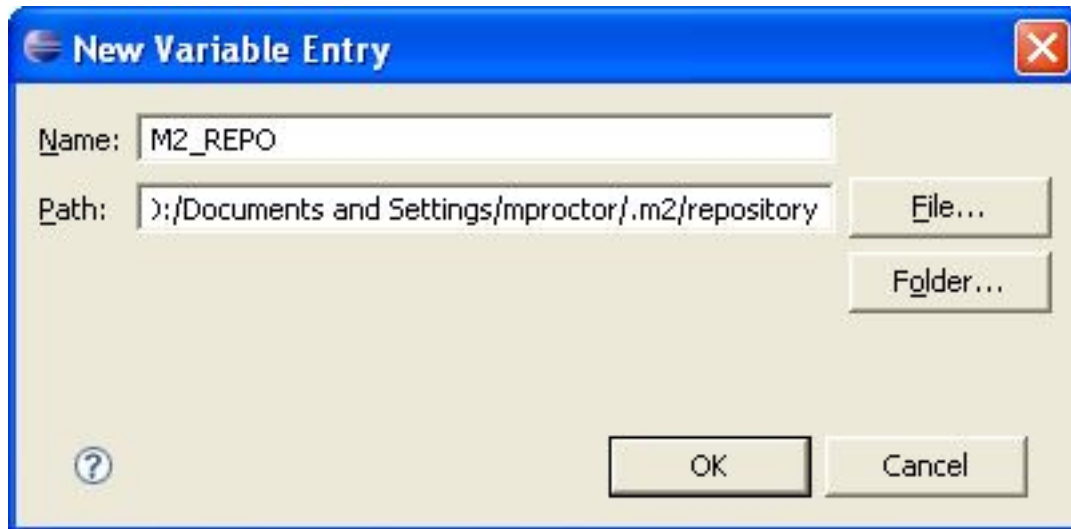


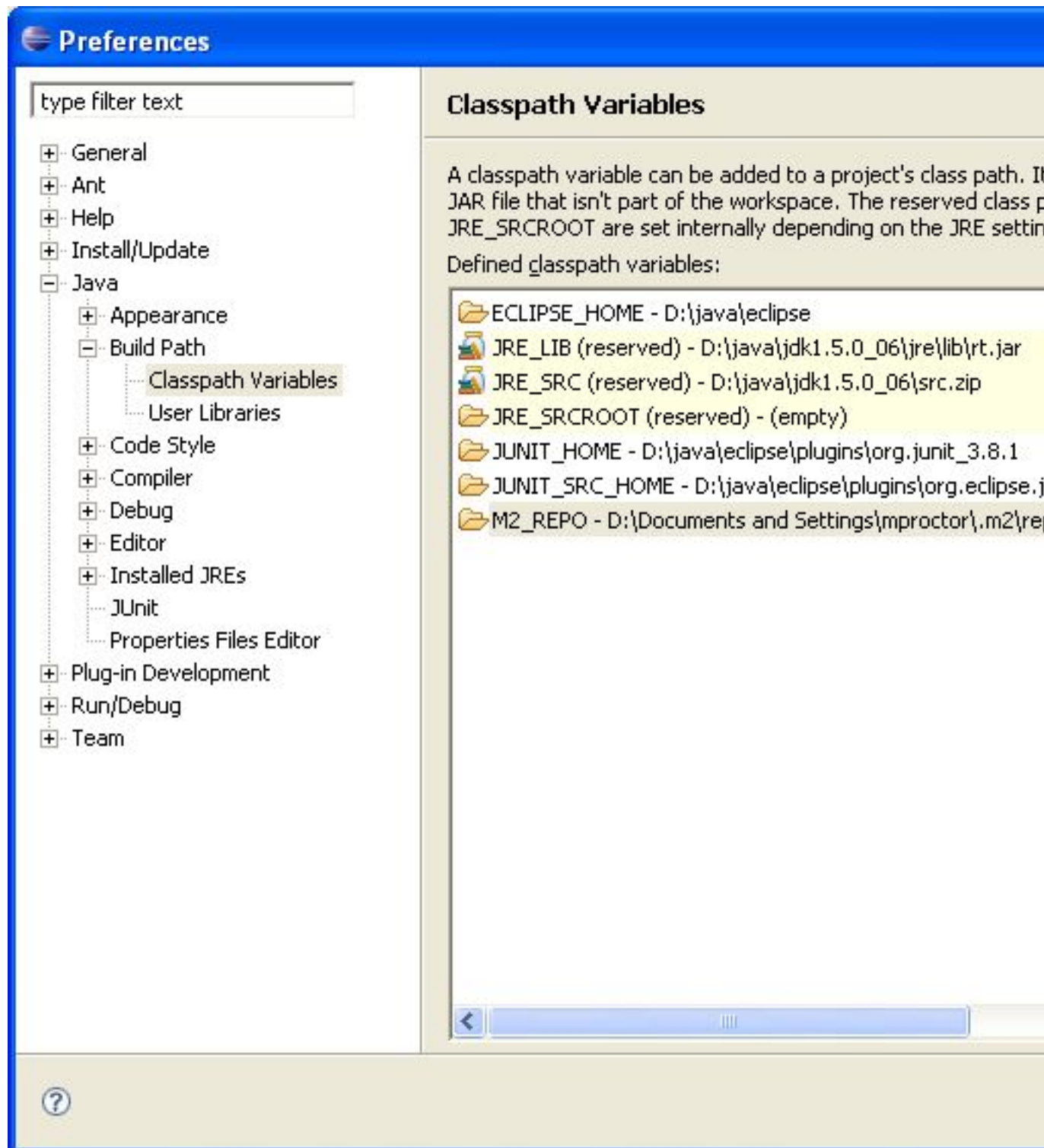


When calling `mvn install` all the project dependencies were downloaded and added to the local Maven repository. Eclipse cannot find those dependencies unless you tell it where that repository is. To do this setup an `M2_REPO` classpath variable.









Chapter 3. Drools Release Notes

3.1. What is New and Noteworthy in Drools 5.4.0.Beta1

3.1.1. Drools Expert

3.1.1.1. Traits

Traits were introduced in the 5.3 release, and details on them can be found in the N&N for there. This release adds an example so that people have something simple to run, to help them understand. In the drools-examples source project open the classes and drl for the namespace `"/org/drools/examples/traits"`. There you will find an example around classifications of students and workers.

```
rule "Students and Workers" no-loop when
    $p : Person( $name : name,
                 $age : age < 25,
                 $weight : weight )
then
    IWorker w = don( $p, IWorker.class, true );
    w.setWage( 1200 );
    update( w );

    IStudent s = don( $p, IStudent.class, true );
    s.setSchool( "SomeSchool" );
    update( s );
end

rule "Working Students" salience -10 when
    $s : IStudent( $school : school,
                  $name : name,
                  this isA IWorker,
                  $wage : fields[ "wage" ] )
then
    System.out.println( $name + " : I have " + $wage + " to pay the fees at
    " + $school );
end
```

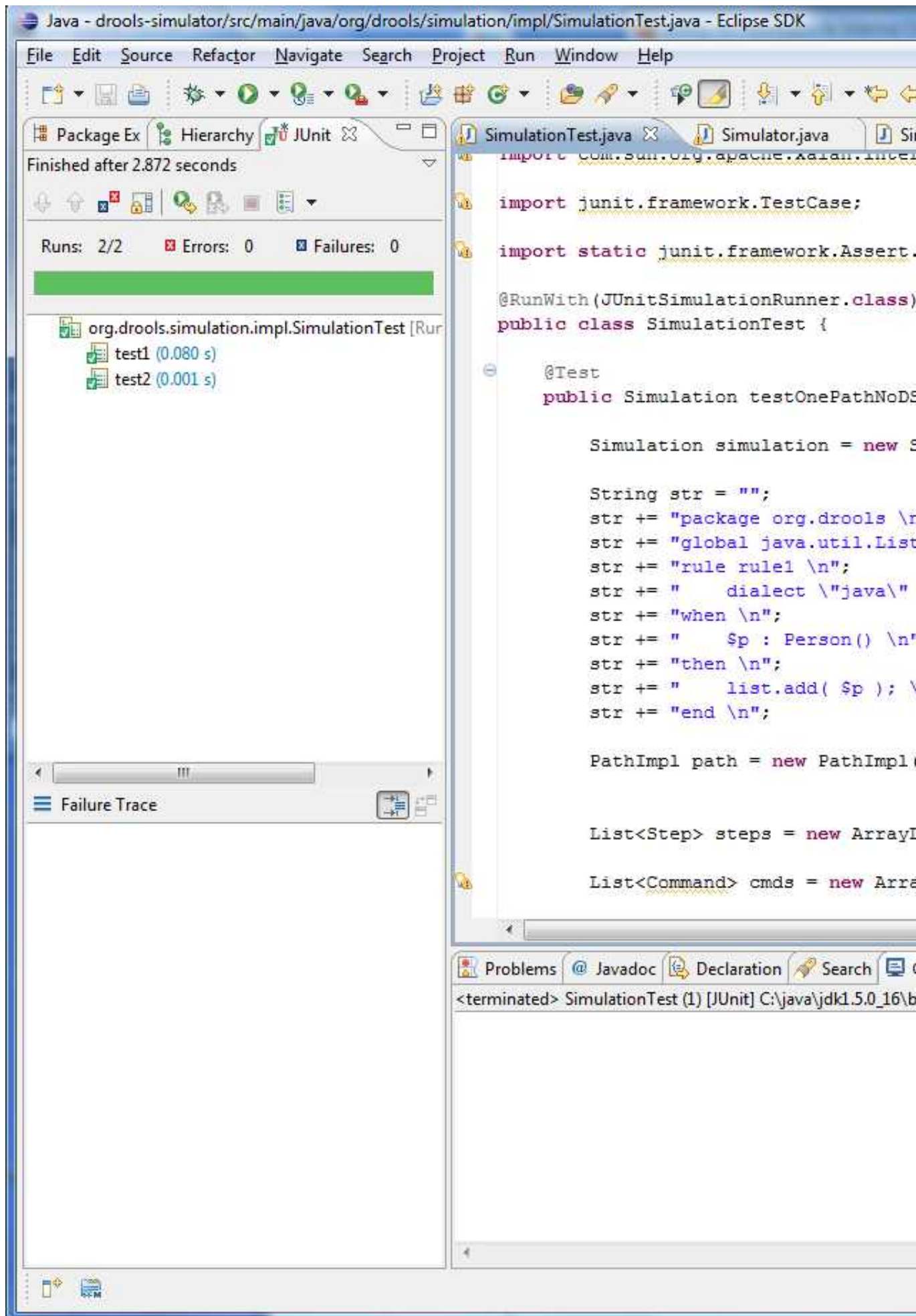
A two part detailed article has been written up at a blog, which will later be improved and rolled into the main documentation. For now you can read them here.

- <http://blog.athico.com/2011/12/new-feature-spotlight-traits-part-1.html> [???

- <http://blog.athico.com/2011/12/dynamic-typing-in-rules-traits-part-2.html>

3.1.1.2. Simulation and Test

The simulation project that was first started in 2009, <http://blog.athico.com/2009/07/drools-simulation-and-test-framework.html>, has undergone an over haul and is now in a usable state. We have not yet promoted this to -api, so it's considered unstable and will change during the beta process. For now though the adventurous people can take a look at the unit tests and start playing.



The Simulator runs the Simulation. The Simulation is your scenario definition. The Simulation consists of 1 to n Paths, you can think of a Path as a sort of Thread. The Path is a chronological line on which Steps are specified at given temporal distances from the start. You don't specify a time unit for the Step, say 12:00am, instead it is always a relative time distance from the start of the Simulation. Each Step contains one or more Commands, i.e. create a StatefulKnowledgeSession or insert an object or start a process. These are the very same commands that you would use to script a knowledge session using the batch execution, so it's re-using existing concepts.

- 1.1 Simulation

- 1..n Paths

- 1..n Steps

- 1..n Commands

All the steps, from all paths, are added to a priority queue which is ordered by the temporal distance, and allows us to incrementally execute the engine using a time slicing approach. The simulator pops of the steps from the queue in turn. For each Step it increments the engine clock and then executes all the Step's Commands.

Here is an example Command (notice it uses the same Commands as used by the CommandExecutor):

```
new InsertObjectCommand( new Person( "darth", 97 ) )
```

Commands can be grouped together, especially Assertion commands, via test groups. The test groups are mapped to JUnit "test methods", so as they pass or fail using a specialised JUnit Runner the Eclipse GUI is updated - as illustrated in the above image, showing two passed test groups named "test1" and "test2".

Using the JUnit integration is trivial. Just annotate the class with `@RunWith(JUnitSimulationRunner.class)`. Then any method that is annotated with `@Test` and returns a Simulation instance will be invoked executing the returned Simulation instance in the Simulator. As test groups are executed the JUnit GUI is updated.

When executing any commands on a KnowledgeBuilder, KnowledgeBase or StatefulKnowledgeSession the system assumes a "register" approach. To get a feel for this look at the [org.drools.simulation.impl.SimulationTest](#) at github (path may change over time).

```
cmds.add( new NewKnowledgeBuilderCommand( null ) );
cmds.add( new SetVariableCommandFromLastReturn( "path1",
                                                KnowledgeBuilder.class.getName() ) );
```



```
cmds.add(
    KnowledgeBuilderAddCommand( ResourceFactory.newByteArrayResource( str.getBytes() ),
                                ResourceType.DRL, null );
```

Notice the set command. "path1" is the context, each path has it's own variable context. All paths inherit from a "root" context. "KnowledgeBuilder.class.getName() " is the name that we are setting the return value of the last command. As mentioned before we consider the class names of those classes as registers, any further commands that attempt to operate on a knowledge builder will use what ever is assigned to that, as in the case of KnowledgeBuilderAddCommand. This allows multiple kbuilders, kbases and ksessions to exist in one context under different variable names, but only the one assigned to the register name is the one that is currently executed on.

The code below show the rough outline used in SimulationTest:

```
Simulation simulation = new SimulationImpl();
PathImpl path = new PathImpl( simulation,
                               "path1" );
simulation.getPaths().put( "path1",
                           path );
List<Step> steps = new ArrayList<Step>();
path.setSteps( steps );
List<Command> cmds = new ArrayList<Command>();
.... add commands to step here ....
// create a step at temporal distance of 2000ms from start
steps.add( new StepImpl( path,
                        cmds,
                        2000 ) );
```

We know the above looks quite verbose. SimulationTest just shows our low level canonical model, the idea is that high level representations are built on top of this. As this is a builder api we are currently focusing on two sets of fluents, compact and standard. We will also work on a spreadsheet UI for building these, and eventually a dedicated textual dsl.

The compact fluent is designed to provide the absolute minimum necessary to run against a single ksession. A good place to start is [org.drools.simulation.impl.CompactFluentTest](https://github.com/droolsjbpm/droolsjbpm-integration/blob/8100e28668537aa5fe04ce5dd0a62f3dc779b30f/drools-simulator/src/test/java/org/drools/simulation/impl/CompactFluentTest.java) [https://github.com/droolsjbpm/droolsjbpm-integration/blob/8100e28668537aa5fe04ce5dd0a62f3dc779b30f/drools-simulator/src/test/java/org/drools/simulation/impl/CompactFluentTest.java] , a snippet of which is shown below. Notice we set "yoda" to "y" and can then assert on that. Currently inside of the test string it executes using mvel. The eventual goal is to build out a set of [hamcrest](http://code.google.com/p/hamcrest/) [http://code.google.com/p/hamcrest/] matchers that will allow assertions against the state of the engine, such as what rules have fired and optionally with with data.

```
FluentCompactSimulation f = new FluentCompactSimulationImpl();
```

```
f.newStatefulKnowledgeSession()
    .getKnowledgeBase()

    .addKnowledgePackages( ResourceFactory.newByteArrayResource( str.getBytes() ),
                          ResourceType.DRL )

    .end()
    .newStep( 100 ) // increases the time 100ms
    .insert( new Person( "yoda",
                        150 ) ).set( "y" )

    .fireAllRules()
    // show testing inside of ksession execution
    .test( "y.name == 'yoda'" )
    .test( "y.age == 160" );
```

Note that the test is not executing at build time, it's building a script to be executed later. The script underneath matches what you saw in `SimulationTest`. Currently the way to run a simulation manually is shown below. Although you already saw in `SimulationTest` that JUnit will execute these automatically. We'll improve this over time.

```
SimulationImpl sim = (SimulationImpl) ((FluentCompactSimulationImpl)
f).getSimulation();
Simulator simulator = new Simulator( sim,
                                   new Date().getTime() );

simulator.run();
```

The standard fluent is almost a 1 to 1 mapping to the canonical path, step and command structure in `SimulationTest`- just more compact. Start by looking in [org.drools.simulation.impl.StandardFluentTest](#). [???]. This fluent allows you to run any number of paths and steps, along with a lot more control over multiple kbuilders, kbases and ksessions.

```
FluentStandardSimulation f = new FluentStandardSimulationImpl();
f.newPath("init")
    .newStep( 0 )
        // set to ROOT, as I want paths to share this
        .newKnowledgeBuilder()
            .add( ResourceFactory.newByteArrayResource( str.getBytes() ),
                ResourceType.DRL )
        .end(ContextManager.ROOT, KnowledgeBuilder.class.getName() )
        .newKnowledgeBase()
            .addKnowledgePackages()
        .end(ContextManager.ROOT, KnowledgeBase.class.getName() )
    .end()
    .newPath( "path1" )
        .newStep( 1000 )
            .newStatefulKnowledgeSession()
```

```

        .insert( new Person( "yoda", 150 ) ).set( "y" )
        .fireAllRules()
        .test( "y.name == 'yoda'" )
        .test( "y.age == 160" )
        .end()
    .end()
    .newPath( "path2" )
    .newStep( 800 )
        .newStatefulKnowledgeSession()
            .insert( new Person( "darth", 70 ) ).set( "d" )
            .fireAllRules()
            .test( "d.name == 'darth'" )
            .test( "d.age == 80" )
        .end()
    .end()
    .end

```

There is still an awful lot to do, this is designed to eventually provide a unified simulation and testing environment for rules, workflow and event processing over time, and eventually also over distributed architectures.

- Flesh out the api to support more commands, and also to encompass jBPM commands
- Improve out of the box usability, including moving interfaces to knowlege-api and hiding "new" constructors with factory methods
- Commands are already marshallable to json and xml. They should be updated to allow full round tripping from java api commands and json/xml documents.
- Develop hamcrest matchers for testing state
 - What rule(s) fired, including optionally what data was used with the executing rule (Drools)
 - What rules are active for a given fact
 - What rules activated and de-activated for a given fact change
 - Process variable state (jBPM)
 - Wait node states (jBPM)
- Design and build tabular authoring tools via spreadsheet, targetting the web with round tripping to excel.
- Design and develop textual DSL for authoing - maybe part of DRL (long term task).

3.1.2. Guvnor

3.1.2.1. Guided Decision Table - Limited Entry

The Guided Decision Table editor and wizard now support the creation of "Limited Entry" tables.

New Rule

☒ Create new:
☐ Import asset from global area:

Name:

Initial category: Home Mortgage
 Commercial Mortgage

Type (format) of rule: Decision Table (Web - guided editor)

☒ Use Wizard

Options: ☐ Extended entry, values defined in table body
☒ Limited entry, values defined in columns

☒ Create in Package: insurance
☐ Create in Global area

Initial description:

OK

Figure 3.1. Selecting the table format

The screenshot shows the 'Guided Decision Table Wizard' window. On the left, a sidebar lists steps: Summary, Add Fact Patterns, Add Constraints (selected), Add Actions to update Facts, Add Actions to insert Facts, and Columns to expand. The main area is titled 'Define constraints on the Facts\Patterns fields.' It contains three panels: 'Available patterns' with '\$a : Applicant' and '\$v : Vehicle'; 'Available fields' with 'this : this', 'name : Text', and 'age : Numeric'; and 'Conditions' with '[Age < 35] age'. Between the panels are '>>' and '<<' buttons. To the right of the 'Conditions' panel are up and down arrow buttons. Below these panels is a green box with the following fields: 'Column header (description):' with 'Age < 35', 'Operator:' with a dropdown set to 'less than', and 'Constraint value:' with '35'. At the bottom are buttons for '<- Previous', 'Next ->', 'Cancel', and 'Finish'.

Figure 3.2. Defining Limited Entry constraints with the Wizard

The screenshot shows the 'Guided Decision Table Wizard' window. On the left, a sidebar lists steps: Summary, Add Fact Patterns, Add Constraints, Add Actions to update Facts, Add Actions to insert Facts (selected), and Columns to expand. The main area is titled 'Define actions to insert new Facts\Patterns.' It contains four panels: 'Available patterns' with 'Applicant', 'Policy', and 'Vehicle'; 'Chosen patterns' with '\$p : Policy'; 'Available fields' with 'this : this', 'type : Text', and 'premium : Numeric'; and 'Chosen fields' with '[Premium 1000] premium'. Between the panels are '>>' and '<<' buttons. Below these panels is a green box with the following fields: 'Binding:' with '\$p', a checkbox for 'Logically assert a fact - the fact will be retracted when the supporting evidence is removed.' (unchecked), 'Column header (description):' with 'Premium 1000', and 'Constraint value:' with '1000'. At the bottom are buttons for '<- Previous', 'Next ->', 'Cancel', and 'Finish'.

Figure 3.3. Defining Limited Entry actions with the Wizard

File Edit Source

Attributes Edit

+ Decision table















	#	Description	Age < 35	BMW	Audi	Premium 1000	
			Applicant [\$a]	Vehicle [\$v]			
			age [<35]	make [==BMW]	make [==Audi]		
	1		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
	2		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
	3		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
	4		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
	5		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
	6		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
	7		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
	8		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Figure 3.4. An example expanded, limited entry table

Condition column configuration 

Pattern: Vehicle [\$v] 

Field:  

Operator: equal to 

From Entry Point:

Column header (description):

Constraint value:

Hide column: ☐

Apply changes

Figure 3.5. Editing a constraint in the editor

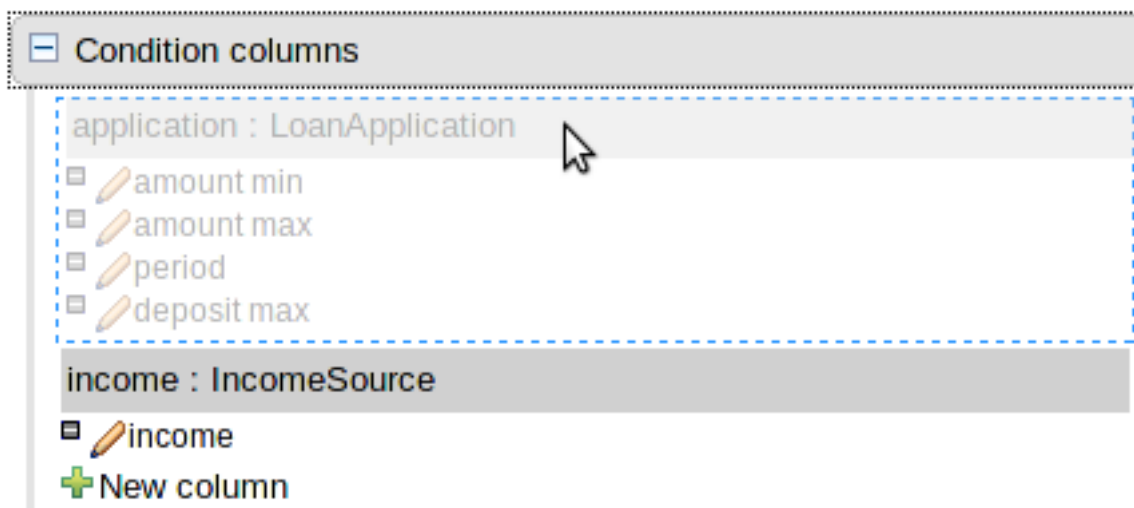


Figure 3.8. Re-arranging Condition patterns

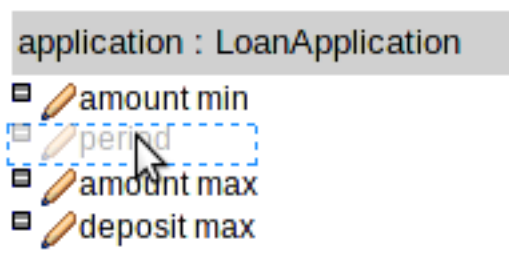


Figure 3.9. Re-arranging columns

3.1.2.4. Guided Decision Table - Actions to retract Facts

This release brings the ability to define Action columns to retract Facts.

If you are authoring an Extended Entry decision table the column definition contains basic information and the fact being retracted is held in the table itself.

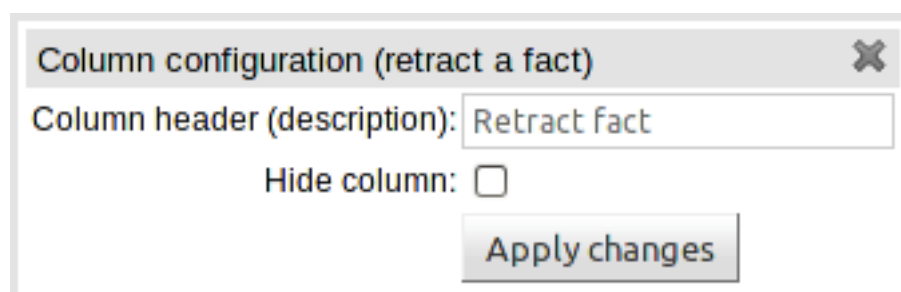
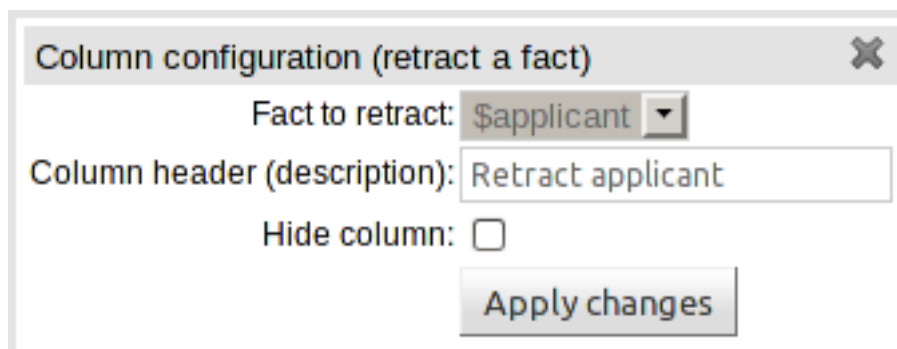


Figure 3.10. Defining an Extended Entry retraction

If however you are authoring a Limited Entry decision table the Fact being retracted is defined in the column definition.



Column configuration (retract a fact) [X]

Fact to retract: \$applicant [v]

Column header (description): Retract applicant

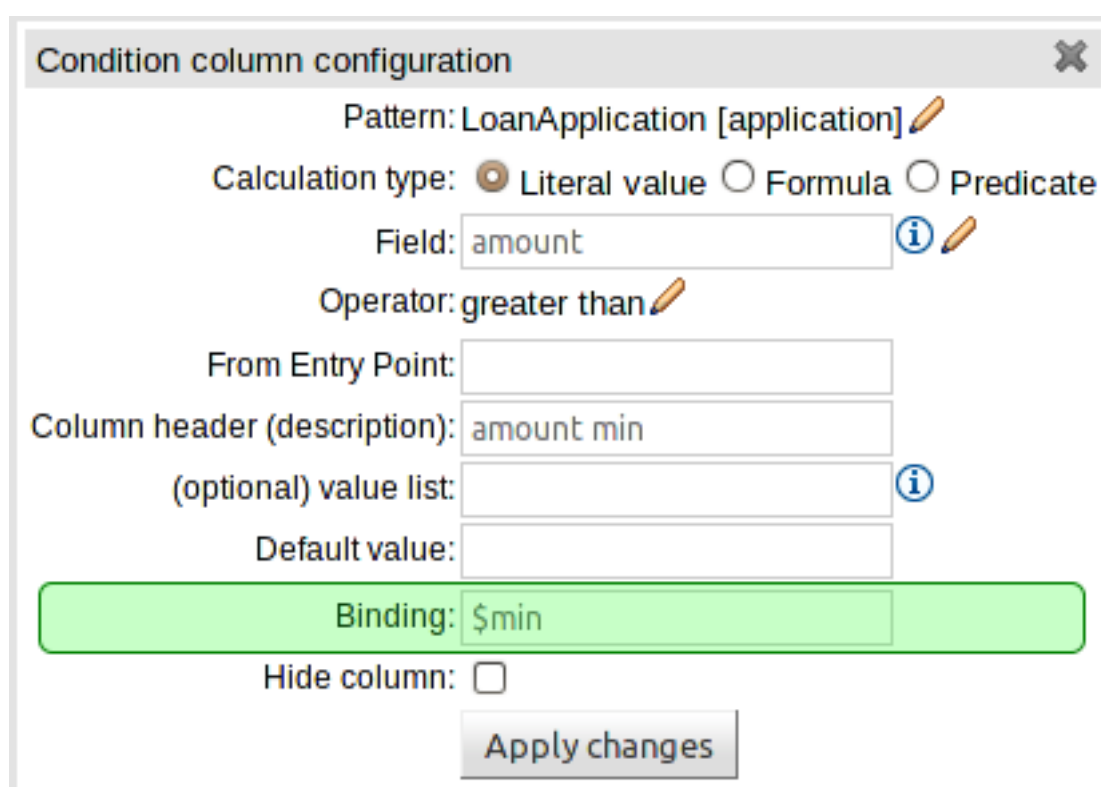
Hide column: ☐

Apply changes

Figure 3.11. Defining a Limited Entry retraction

3.1.2.5. Guided Decision Table - Binding condition fields

You can now bind fields in Conditions to variables. These variables can then be used in Predicate or Formula conditions, and Work Item actions.



Condition column configuration [X]

Pattern: LoanApplication [application] [pencil]

Calculation type: ☒ Literal value ☐ Formula ☐ Predicate

Field: amount [i] [pencil]

Operator: greater than [pencil]

From Entry Point: [text box]

Column header (description): amount min

(optional) value list: [text box] [i]

Default value: [text box]

Binding: \$min

Hide column: ☐

Apply changes

Figure 3.12. Binding a field

Decision table										
	#	Description	\$min : amount min	amount max	period	deposit max	income	Loan approved	LMI	rate
			LoanApplication [application]				IncomeSource	application	application	application
			amount [>]	amount [<]	lengthYears [==]	deposit [<]	type [==]	approved	insuranceCost	approvedRate
+	1		131000	\$min * 2	30	20000	Asset	true	0	2
+	2		10000	\$min * 2	20	2000	Job	true	0	4
+	3		100001	\$min * 2	20	3000	Job	true	10	6

Figure 3.13. A Decision Table using bound fields

3.1.2.6. Guided Decision Table - Using jBPM Work Item actions

jBPM Work Items can now be used as Actions; and the corresponding Work Item Handler invoked at runtime. Work Item Handlers should be added to the runtime session as normal.

Work Item input parameters can either be defined as a literal value in the column definition or as a Fact or Fact Field binding.

New Actions have been created to perform the following functions:-

- Execute a Work Item
- Set the value of a field on an existing Fact to the value of a Work Item output (result) parameter.
- Set the value of a field on a new Fact to the value of a Work Item output (result) parameter.

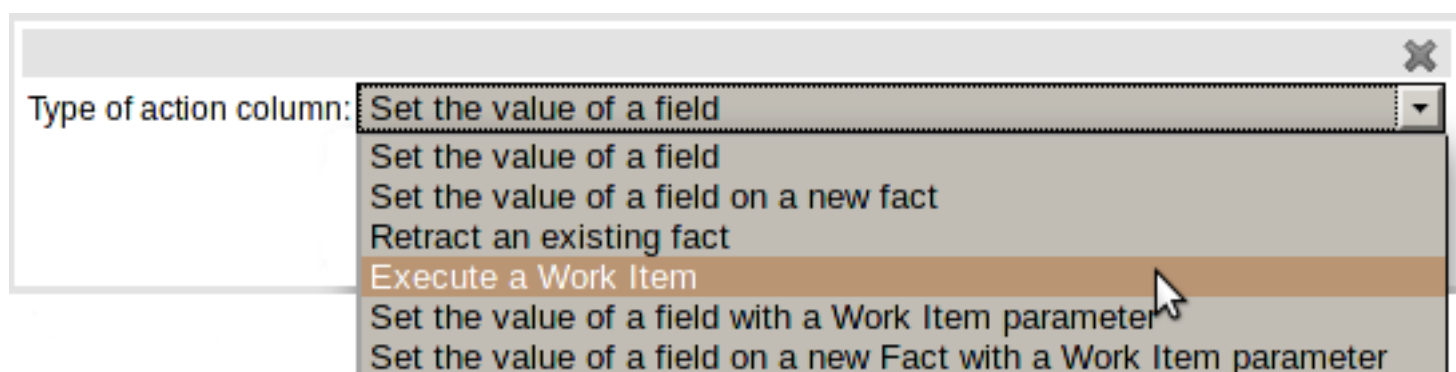
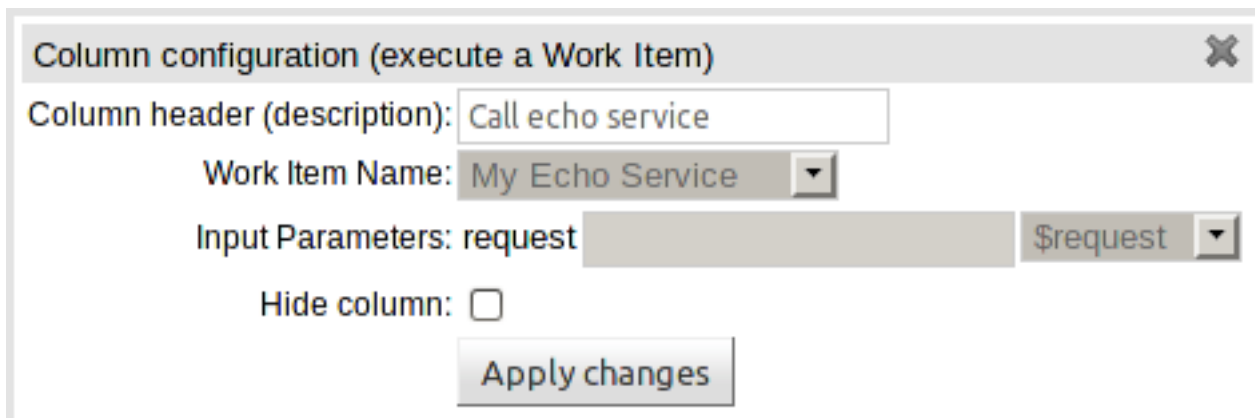


Figure 3.14. New Actions



Column configuration (execute a Work Item) [X]

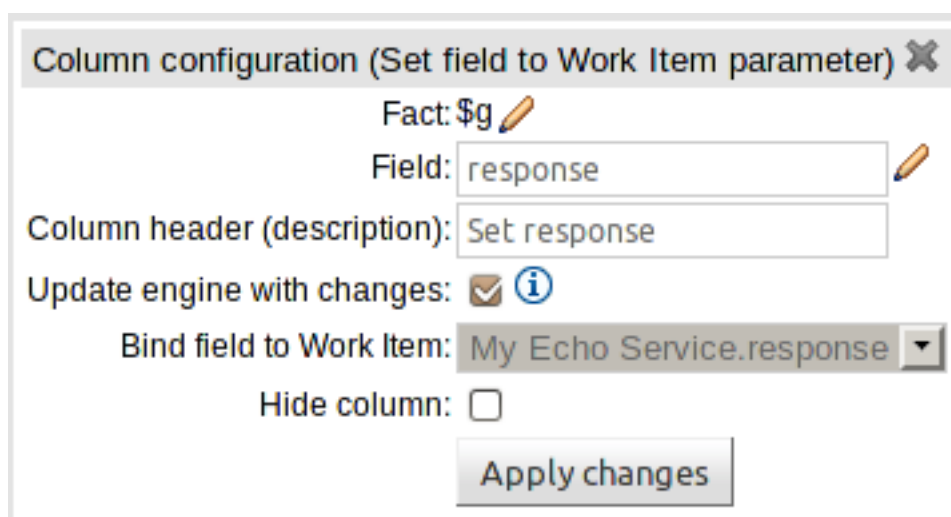
Column header (description):

Work Item Name: [v]

Input Parameters: request [v]

Hide column: ☐

Figure 3.15. Calling a Work Item



Column configuration (Set field to Work Item parameter) [X]

Fact: \$g [pencil icon]

Field: [pencil icon]

Column header (description):

Update engine with changes: ☒ [i icon]

Bind field to Work Item: [v]

Hide column: ☐





Figure 3.16. Setting a field from a Work Item output (result) parameter

3.1.2.7. Decision table analysis

On the bottom of Web decision tables, there's now a button **Analyze...** which will check if there are issues in your decision table, such as impossible matches and conflicting matches.

3.1.2.7.1. Impossible match detection

An impossible match is a row in a decision table that can never match. For example:





Decision table							
	#	Description	Minimum age	Maximum age	First offence?	Fee	Ana
	1			20	<input checked="" type="checkbox"/>	150	
	2		21	20	<input checked="" type="checkbox"/>	200	Impossible m
	3		65		<input checked="" type="checkbox"/>	100	
	4				<input type="checkbox"/>	1000	

[Add row...](#)
[Otherwise](#)
[Analyze...](#)

In the decision table above, row 2 is an impossible match, because there is no Person with a minimum age of 21 and a maximum age of 20.

3.1.2.7.2. Conflicting match detection

A conflicting match are 2 rows in a decision table that can both match.

Decision table							
	#	Description	Minimum age	Maximum age	First offence?	Fee	Ana
	1			20	<input checked="" type="checkbox"/>	150	
	2		21	70	<input checked="" type="checkbox"/>	200	Conflicting ma
	3		65		<input checked="" type="checkbox"/>	100	Conflicting ma
	4				<input type="checkbox"/>	1000	

[Add row...](#)
[Otherwise](#)
[Analyze...](#)

In the decision table above, row 2 and 3 are a conflicting match, because a Person of age 67 will match both rows: it's unclear whether that Person has to pay a fee of 200 or 100.

3.1.2.8. REST api changes

- Added support to create (PUT) and delete (DELETE) a Category with `rest/categories/{categoryName}`.
- Removed buggy `category` property from `Package`: it was `null` upon GET and ignored upon PUT.
- The REST resource to get the assets for a certain category has been moved from `rest/categories/{categoryName}` to `rest/categories/{categoryName}/assets`.
- The REST resources now properly encode and decode the URL's. Note that URL path encoding is different than URL query encoding: the former does not accept `+` as encoding for space.

3.1.2.9. JBoss AS 6 war is no longer shipped in the release

This does not mean that Guvnor can not be made to work in AS 6. We are just focusing on JBoss AS 7 and Tomcat.

3.1.2.10. Rule Templates - Interpolation keys in free-form DRL

When using Rule Templates you can now define interpolation variables in free-form DRL blocks. Template Keys are formatted as documented in Drools Expert User Guide; i.e. `@{keyName}`. All Template Keys in 'free-form' LHS or RHS elements are considered 'Text' (i.e. you'll get a TextBox in the Template data screen). Data-types should be correctly escaped in the 'free-form' entry. For example: `System.out.println("@{key}");`

3.1.2.11. Relational operators for Strings

All guided rule editors (BRL, Rule Templates and Decision Tables) now support relation operators for String values.

3.1.3. Planner

3.1.3.1. Build-in move factories

Planner now comes with 2 build-in move factories: `GenericChangeMoveFactory` and `GenericSwapMoveFactory`. Here's an example that uses both of them:

```
<localSearch>
  <selector>
    <selector>

moveFactoryClass>
  </selector>
  <selector>

moveFactoryClass>
  </selector>
</selector>
...
</localSearch>
```

It's no longer required to write your own Move and MoveFactory implementations, but you still can (and mix those in too).

3.1.4. Known Issues

3.1.4.1. Breakpoints in Rule RHS are not working with Java dialect

Debugging with Java is not working. Debugging with MVEL is not affected. This will be fixed in the future releases before 5.4.0.Final

3.1.4.2. OSGi Modules are not stable

They will be fixed for 5.4.0.Beta2

3.1.4.3. Test Scenarios in Guvnor are not working

When running the test scenarios. An error pops up saying that the work items could not be loaded. This will be fixed in Beta2.

3.2. What is New and Noteworthy in Drools 5.3.0

3.2.1. Drools Expert

3.2.1.1. Core and Compiler compatility will no longer be maintained

This is the last release where compatibility in core/compiler and other submodules will be maintained, as we refactor for cleanliness in the following releases. All developers should be using the knowledge-api.

3.2.1.2. Queries and Backward Chaining

Queries and Backward Chaining have had extensive bug fixing as it had many cases that didn't work, especially with "open" queries. Anyone using backward chaining queries in 5.2 should stop straight away and start using this.

3.2.1.3. Declarative Agenda

5.3 Introduces the declarative agenda, where rules can be used to control which rules can fire and when. While this will add a lot more overhead than the simple use of salience, the advantage is it is declarative and thus more readable and maintainable and should allow more use cases to be achieved in a simpler fashion.

This feature is off by default and must be explicitly enabled, that is because it is considered highly experimental for the moment and will be subject to change.

Example 3.1. Enabling the Declarative Agenda

```
KnowledgeBaseConfiguration kconf =
    KnowledgeBaseFactory.newKnowledgeBaseConfiguration();
kconf.setOption( DeclarativeAgendaOption.ENABLED );
KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase( kconf );
```

The basic idea is:

- All matched rule's Activations are inserted into WorkingMemory as facts. So you can now match against an Activation. The rule's metadata and declarations are available as fields on the Activation object.

- You can use the `kcontext.blockActivation(Activation match)` for the current rule to block the selected activation. Only when that rule becomes false will the activation be eligible for firing. If it is already eligible for firing and is later blocked, it will be removed from the agenda until it is unblocked.
- An activation may have multiple blockers and a count is kept. All blockers must become false for the counter to reach zero to enable the Activation to be eligible for firing.
- `kcontext.unblockAllActivations($a)` is an over-ride rule that will remove all blockers regardless
- An activation may also be cancelled, so it never fires with `cancelActivation`
- An unblocked Activation is added to the Agenda and obeys normal salience, agenda groups, ruleflow groups etc.
- `@activationListener('direct')` allows a rule to fire as soon as it's matched, this is to be used for rules that block/unblock activations, it is not desirable for these rules to have side effects that impact elsewhere. The name may change later, this is actually part of the pluggable terminal node handlers I made, which is an "internal" feature for the moment.

Example 3.2. New RuleContext methods

```
void blockActivation(Activation match);
void unblockAllActivations(Activation match);
void cancelActivation(Activation match);
```

Here is a basic example that will block all activations from rules that have metadata `@department('sales')`. They will stay blocked until the `blockerAllSalesRules` rule becomes false, i.e. "go2" is retracted.

Example 3.3. Block rules based on rule metadata

```
rule rule1 @department('sales') when
    $s : String( this == 'go1' )
then
    list.add( kcontext.rule.name + ':' + $s );
end
rule rule2 @department('sales') when
    $s : String( this == 'go1' )
then
    list.add( kcontext.rule.name + ':' + $s );
end
rule blockerAllSalesRules @activationListener('direct') when
    $s : String( this == 'go2' )
    $i : Activation( department == 'sales' )
then
    list.add( $i.rule.name + ':' + $s );
```

```
kcontext.blockActivation( $i );
end
```

This example shows how you can use active property to count the number of active or inactive (already fired) activations.

Example 3.4. Count the number of active/inactive Activations

```
rule rule1 @department('sales') when
    $s : String( this == 'gol' )
then
    list.add( kcontext.rule.name + ':' + $s );
end
rule rule2 @department('sales') when
    $s : String( this == 'gol' )
then
    list.add( kcontext.rule.name + ':' + $s );
end
rule rule3 @department('sales') when
    $s : String( this == 'gol' )
then
    list.add( kcontext.rule.name + ':' + $s );
end
rule countActivateInActive @activationListener('direct') when
    $s : String( this == 'go2' )
    $active : Number( this == 1 ) from accumulate( $a : Activation( department
== 'sales', active == true ), count( $a ) )
    $inactive : Number( this == 2 ) from accumulate( $a : Activation( department
== 'sales', active == false ), count( $a ) )
then
    kcontext.halt( );
end
```

3.2.1.4. Faster compilation

Thanks to many improvements and optimizations made on both MVEL library and Drools internals, the DRL compilation is now at least 3 times faster for both MVEL and Java dialects.

3.2.1.5. Entry-point declarations

Entry points can now be explicitly declared. Syntax is:

```
entryPointDeclaration := DECLARE ENTRY-POINT stringId annotation* END
```

Example use:

Example 3.5.

```
declare entry-point X
    @doc( "This entry point is receives events from the message queue X" )
end
```

3.2.1.6. Traits (experimental)

A Drools Trait is a bean interface which can be attached - and removed - to and from an individual object at runtime. While an object wears a trait, a reference of the trait type is returned, so methods defined in the trait interface can be called normally. A trait, then, adds a type and some fields to an object. If a bean has a field of the given name and type, that field will be used to support the interface. "Virtual" fields, instead, will be stored as entries in a map, or as triples in an in-memory store.

Traits are declared with the attribute `@format(trait)`. Unlike normal beans, declared traits will generate interfaces instead of classes: the declared fields will be mapped to getters/setters. Notice that multiple traits can be worn at the same time.

3.2.1.6.1. IsA operator

The operator `isA` can be used in patterns to check whether an object is wearing a trait or not

E.g. `Worker(this isA Student)`

3.2.1.6.2. Special Classes

- `Thing` : Interface, automatically extended by all traits
- `Entity` : Class without concrete fields, optimized for virtual fields

3.2.1.6.3. Usage

To add/remove a trait to an object, in the RHS:

- `TraitClass traited = don($object, TraitClass)`
- `Thing stripped = shed($traitedObject, TraitClass)`

3.2.2. Guvnor**3.2.2.1. Declarative type extension**

Following the enhancement to Drools Expert type declaration in Guvnor now support 'extends' to inherit from super-classes. Sub-classes can extend either other types declared in the same

package or imported Java classes. In order to extend a type declared in Java by a declared subtype, repeat the supertype in a declare statement without any fields.

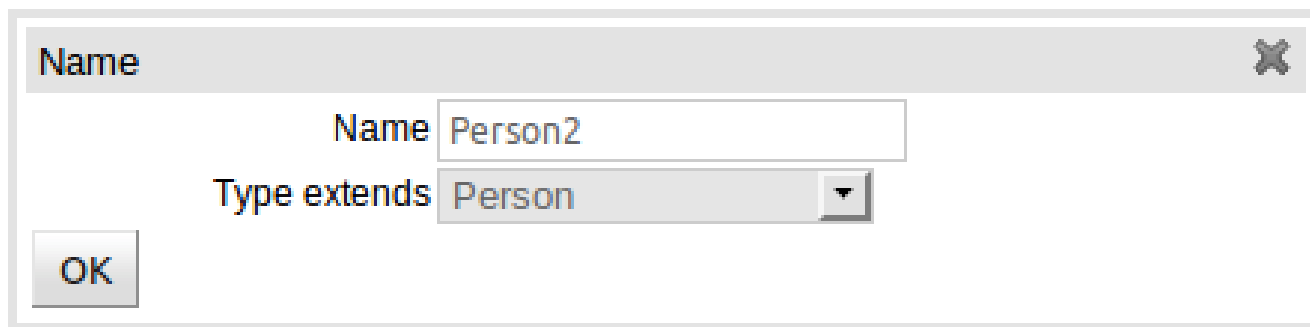


Figure 3.17. Extending a class

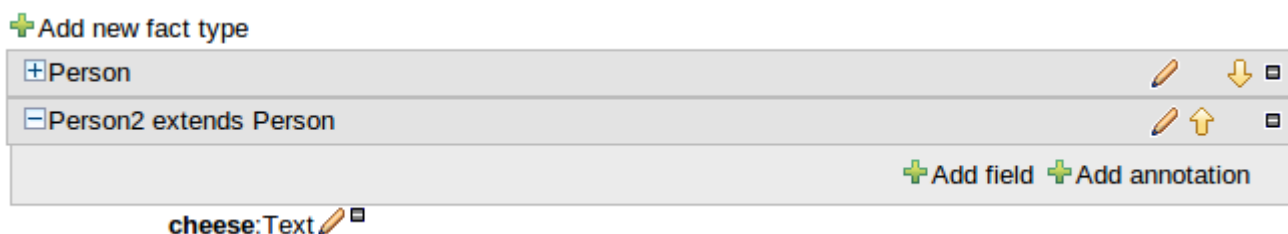


Figure 3.18. Declarative model showing extensions

3.2.2.2. Improved Knowledge Bases view

Thanks to the help of a community member the tree-view shown in the Knowledge Bases view has been improved.

We also took the opportunity to make a few improvements of our own.

- The view no longer repeats intermediate level sub-package names that are empty (community led effort).
- The package view can be viewed hierarchically, as has been the default up to 5.2.0.Final.
- The package view can now also be viewed "flat" with no nesting
- The tree's nodes can be fully expanded or collapsed.

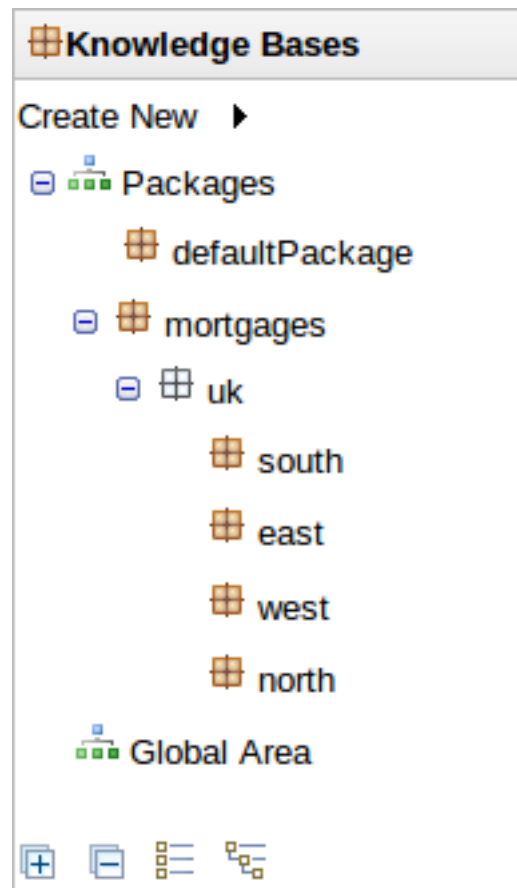


Figure 3.19. Hierarchical view

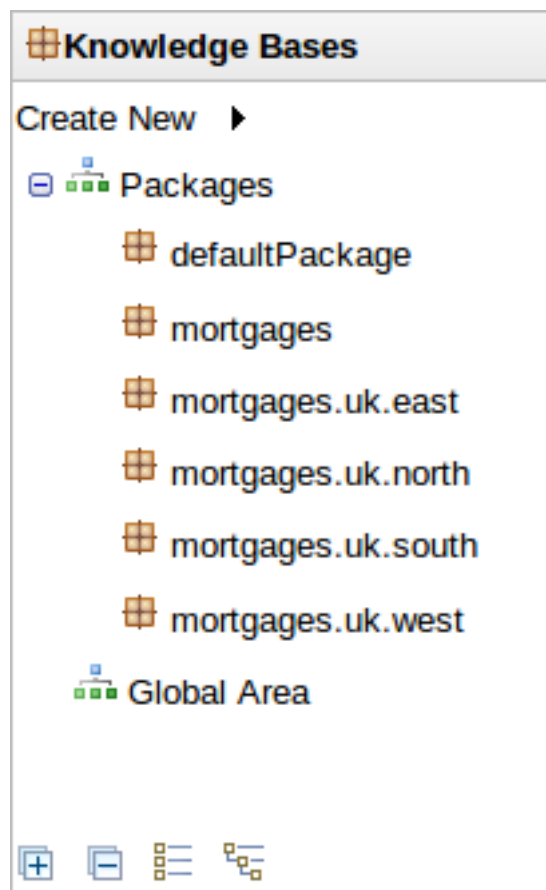


Figure 3.20. Flat view

3.2.2.3. Asset Viewer

Guvnor continues to grow and improve with every passing release. One of the requirements we have is to move Guvnor away from a pure "Rules" or "Knowledge" repository management and authoring environment to a more generic one, in which consumers of different technologies can tailor their experience to their domain's requirements.

One of the more noticeable changes we are making to support this has been to de-couple asset types and groups from the code. What was once previously static code is now defined by compile-time configuration. Due to limitations in GWT (no client-side reflection), the technology with which Guvnor is implemented, we are unfortunately not able to offer runtime configuration at this moment.


The asset groups are configurable within source file; `src/main/resources/drools-asseteditors.xml`. This file is used at GWT compile time to wire-up asset types with their respective editor, group and group icon. An example extract from the foregoing file looks like this:-

```
<asseteditor>
  <class>org.drools.guvnor.client.modeldriven.ui.RuleModeller</class>
  <format>brl</format>
```

```
<icon>images.ruleAsset()</icon> <title>constants.BusinessRuleAssets()</title>
</asseteditor>
```

To emphasize the separation, asset groups have become their own "editor" appearing as a tab in Guvnor's main, central panel.



Find **mortgages**



 **Package 'mortgages' assets**

[View package configuration](#)

[+ Business rule assets](#)

[- Technical rule assets](#)

[Refresh list](#) [Open selected](#) [Open selected to single tab](#)  

	Format	Name	Status	Last modified	Open
		Dummy rule	Draft	2008 Oct 2 02:23:51	Open

1 of 1

- [+ Functions](#)
- [+ DSL configurations](#)
- [+ Model](#)
- [+ Processes](#)
- [+ Enumerations](#)
- [+ Test Scenarios](#)
- [+ XML, Properties](#)
- [+ Other assets, documentation](#)
- [+ WorkingSets](#)
- [+ SpringContext](#)
- [+ WorkItemDefinition](#)

Figure 3.21. Asset Viewer

The format of the new screen is being tried for 5.3.0.Beta1. There has been some discussion whether a single table containing all assets would be better - with collapsible rows to group different

types of asset. The immediate problem with this approach is however that finding different asset types on a "paged table" becomes more cumbersome for the user; as they'd have to sort by type and page through.

We therefore ask for community feedback.

3.2.2.4. Guvnor enumeration improvements

Previously Guvnor enumerations that had a "display value" and a "DRL value" (i.e. the value substituted for the display value when DRL was generated) could be defined with "<DRL value>=<Display value>". Various community users have been using Guvnor enumerations to support complex rule definitions in both DSL and the guided Decision Table editor.

```
'Fact.operator' : ['equals=\\=\\=', 'not equals=!\\=']
```

3.2.2.5. Guvnor Guided Decision Table

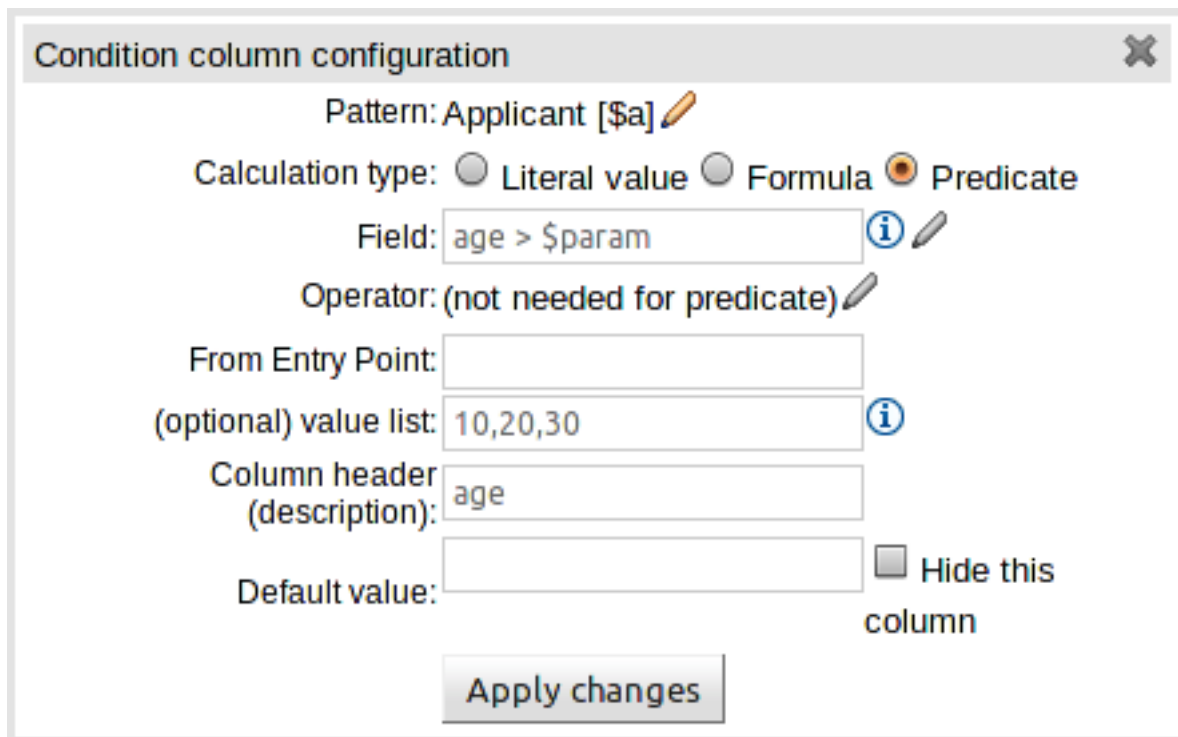
A couple of minor enhancements have been made to Guvnor's guided decision table editor:-

- Removal of sorting

You are now able to restore the original sort order of a column by clicking on the sort icon through: ascending, descending and none.

- Formulae and Predicates support value lists

Up until now only literal value columns could take advantage of value lists; either "Guvnor enums" or the Decision Table's "Optional value list". This been rectified with this release bringing the advantage of predefined choices for these types of fields to the business user.



Condition column configuration

Pattern: Applicant [\$a]

Calculation type: ☐ Literal value ☐ Formula ☒ Predicate

Field: age > \$param

Operator: (not needed for predicate)

From Entry Point:

(optional) value list: 10,20,30

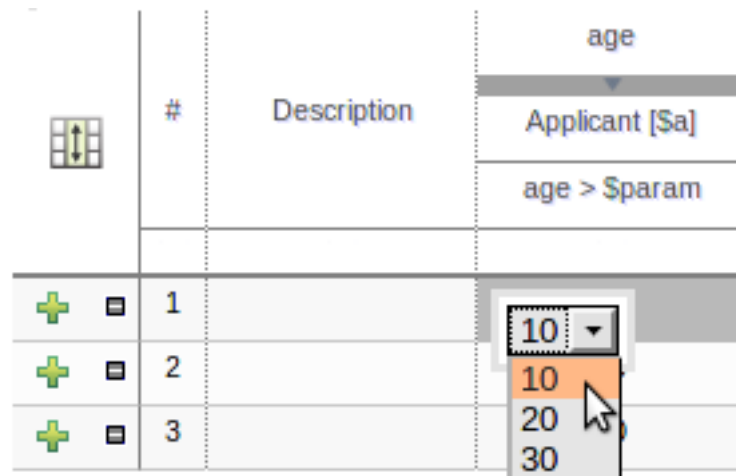
Column header (description): age

Default value:

☐ Hide this column

Apply changes

Figure 3.22. Defining a predicate with a value list



	#	Description	age
			Applicant [\$a]
			age > \$param
+	1		10
+	2		10
+	3		20
			30

Figure 3.23. Using the predicate column

3.2.2.6. Solid URLs and working browser history


Guvnor URLs point to what ever is currently open in the active tab. Because of this you can bookmark the URL or email it to your colleague. Forward and next page directs you to the previously opened tab or to the next tab.

3.2.2.7. Guvnor Guided Decision Table - Wizard

A wizard has been created to assist with the construction of a new table.

The wizard takes the user through the definition process, from adding patterns to creating constraints and generating an expanded form.

New Rule



New Rule

☒ Create new:
☐ Import asset from global area:

Name:

Initial category:

Home Mortgage

Commercial Mortgage

Type (format) of rule:

Decision Table (Web - guided editor)

Use Wizard: ☒

☒ Create in Package:

insurance

☐ Create in Global area

Initial description:

OK

Figure 3.24. Invoke the ewizard

The screenshot shows the 'Guided Decision Table Wizard' window. On the left, a sidebar lists the steps: Summary (checked), Add Fact Patterns, Add Constraints, Add Actions to set fields, Add Actions to insert new Facts, and Columns to expand. The main area is titled 'Summary of fields for the decision table.' and contains three fields: 'Name:' with the value 'availablePolicies' and a red asterisk, 'Initial description:' with an empty text box, and 'Create in Package:' with the value 'insurance'. At the bottom, there are four buttons: '< Previous', 'Next >', 'Cancel', and 'Finish'.

Figure 3.25. Summary page

The screenshot shows the 'Guided Decision Table Wizard' window at the 'Add Fact Patterns' step. The sidebar shows 'Summary' and 'Add Fact Patterns' (checked), with 'Add Constraints' as the next step. The main area is titled 'Define Facts\Patterns on which constraints can be defined.' and contains two lists: 'Available patterns' with 'Applicant', 'Policy', and 'Vehicle' (selected), and 'Chosen patterns' with '\$a : Applicant' and '\$v : Vehicle'. Between the lists are '>>' and '<<' buttons. To the right of the 'Chosen patterns' list are up and down arrow buttons. Below the lists, there are fields for 'Binding:' with the value '\$v' and a red asterisk, and 'From Entry Point:' with an empty text box. At the bottom, there are four buttons: '< Previous', 'Next >', 'Cancel', and 'Finish'.

Figure 3.26. Add facts

The screenshot shows the 'Guided Decision Table Wizard' window at the 'Add Constraints' step. The left sidebar contains a checklist: Summary, Add Fact Patterns, Add Constraints (selected), Add Actions to set fields, Add Actions to insert new Facts, and Columns to expand. The main area is titled 'Define constraints on the Facts\Patterns fields.' It features three panels: 'Available patterns' with '\$a : Applicant' and '\$v : Vehicle'; 'Available fields' with 'this', 'make : Text', 'model : Text', 'age : Whole number (integer)', 'value : Whole number (integer)', and '[New Predicate]'; and 'Conditions' with '[make] make' and '[model] model'. Between the panels are '>>' and '<<' buttons. To the right of the 'Conditions' panel are up and down arrow buttons. Below the panels, the 'Calculation type' is set to 'Literal value'. The 'Column header (description)' is 'make'. The 'Operator' is 'equal to'. The '(optional) value list' is 'BMW,Audi'. The 'Default value' is empty. At the bottom are buttons for '<- Previous', 'Next ->', 'Cancel', and 'Finish'.

Figure 3.27. Add constraints to facts

The screenshot shows the 'Guided Decision Table Wizard' window at the 'Add Actions to set fields' step. The left sidebar contains a checklist: Summary, Add Fact Patterns, Add Constraints, Add Actions to set fields (selected), Add Actions to insert new Facts, and Columns to expand. The main area is titled 'Define actions to set the fields on bound Facts\Patterns.' It features three panels: 'Available patterns' with '\$a : Applicant' and '\$v : Vehicle'; 'Available fields' with 'this', 'make : Text', 'model : Text', 'age : Whole number (integer)', and 'value : Whole number (integer)'; and 'Chosen fields' with the text 'No fields chosen'. Between the panels are '>>' and '<<' buttons. At the bottom are buttons for '<- Previous', 'Next ->', 'Cancel', and 'Finish'.

Figure 3.28. Add actions to set fields on bound facts

The screenshot shows the 'Guided Decision Table Wizard' window. On the left, a sidebar lists steps: Summary, Add Fact Patterns, Add Constraints, Add Actions to set fields, **Add Actions to insert new Facts**, and Columns to expand. The main area is titled 'Define actions to insert new Facts\Patterns.' It contains four panels: 'Available patterns' (Applicant, Policy, Vehicle), 'Chosen patterns' (\$p : Policy), 'Available fields' (this, type : Text, premium : Whole number), and 'Chosen fields' ([type] type, [premium] premium). Navigation buttons '>>' and '<<' are between the panels. Below the panels, there are input fields for 'Binding: \$p', a checkbox for 'Logically assert a fact - the fact will be retracted when the supporting evidence is removed.', 'Column header (description): premium', '(optional) value list:', and 'Default value:'. At the bottom are buttons: '<- Previous', 'Next ->', 'Cancel', and 'Finish'.

Figure 3.29. Add actions to insert new facts

The screenshot shows the 'Guided Decision Table Wizard' window at the 'Columns to expand' step. The sidebar on the left has 'Columns to expand' selected. The main area is titled 'Define the columns from which the generated table will be expanded.' and contains a checked checkbox for 'Fully expand the table, including all columns.' At the bottom are buttons: '<- Previous', 'Next ->', 'Cancel', and 'Finish'.

Figure 3.30. Choose how rows are created - all columns

The screenshot shows the 'Guided Decision Table Wizard' dialog box. On the left is a sidebar with a list of steps, each preceded by a green checkmark: 'Summary', 'Add Fact Patterns', 'Add Constraints', 'Add Actions to set fields', 'Add Actions to insert new Facts', and 'Columns to expand'. The 'Columns to expand' step is currently selected. The main area of the dialog is titled 'Define the columns from which the generated table will be expanded.' and contains a checkbox labeled 'Fully expand the table, including all columns.' which is currently unchecked. Below this, there are two list boxes: 'Available columns' on the left and 'Chosen columns' on the right. The 'Available columns' list contains two items: '[make] make' and '[model] model'. The 'Chosen columns' list is currently empty. Between the two list boxes are two buttons: '>>' (to move a column from available to chosen) and '<<' (to move a column from chosen to available). At the bottom of the dialog is a row of four buttons: '< Previous', 'Next >', 'Cancel', and 'Finish'.

Guided Decision Table Wizard

- ✓ Summary
- ✓ Add Fact Patterns
- ✓ Add Constraints
- ✓ Add Actions to set fields
- ✓ Add Actions to insert new Facts
- ✓ Columns to expand

Define the columns from which the generated table will be expanded.

☐ Fully expand the table, including all columns.

Available columns		Chosen columns
[make] make	>>	
[model] model	<<	

< Previous Next > Cancel Finish

Figure 3.31. Choose how rows are created - select the columns you want to expand upon















Decision table						
	#	Description	age	make	model	type
			Applicant [\$a]	Vehicle [\$v]		
			age [>]	make [==]	model [==]	
	1		18	BMW	M3	
	2		25	BMW	M3	
	3		35	BMW	M3	
	4		55	BMW	M3	
	5		18	Audi	M3	
	6		25	Audi	M3	
	7		35	Audi	M3	
	8		55	Audi	M3	
	9		18	BMW	M5	
	10		25	BMW	M5	
	11		35	BMW	M5	
	12		55	BMW	M5	
	13		18	Audi	M5	
Add row...		Otherwise				

Figure 3.32. An extract of the resulting decision table

3.2.2.8. Asset Viewer

The Asset Viewer now only shows sections that contain Assets.

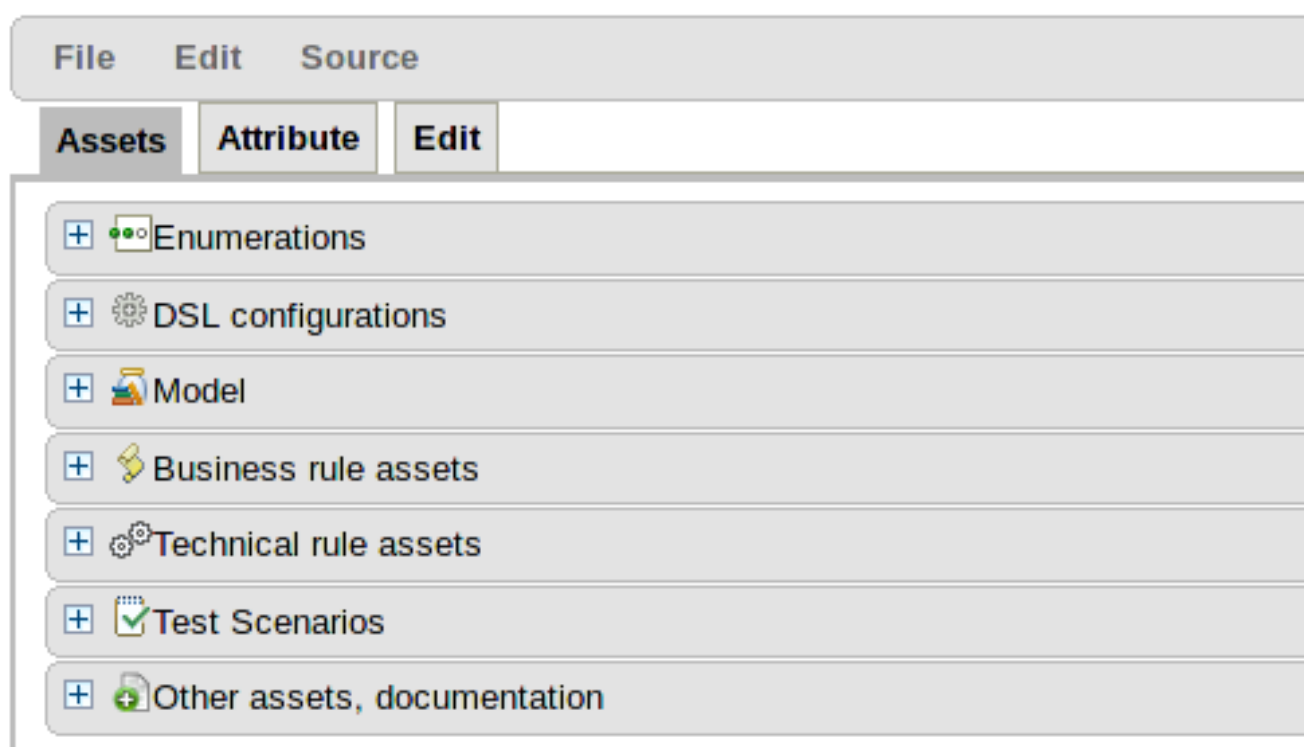


Figure 3.33. A package with assets

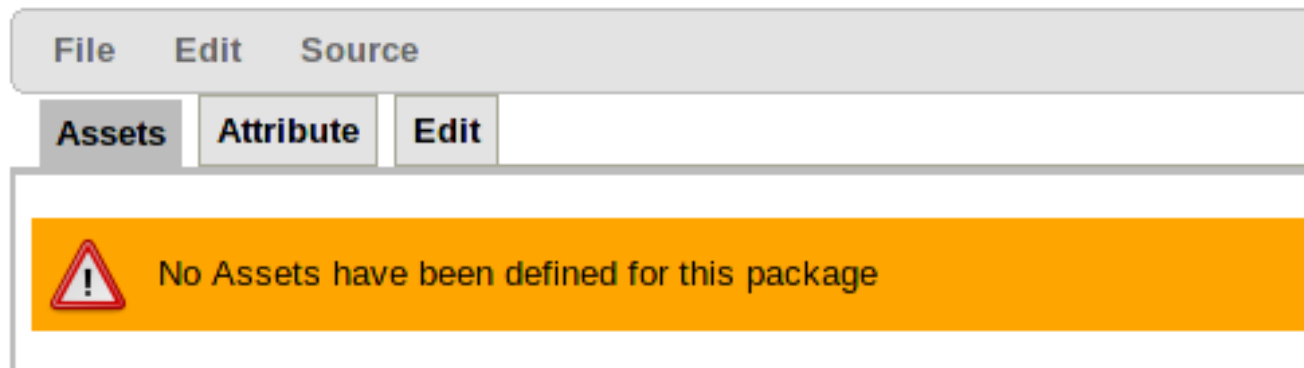


Figure 3.34. An empty package

3.2.2.9. Works on JBoss AS 7

The Guvnor distribution now includes a war that runs on JBoss AS 7.0.1 (not on JBoss AS 7.0.0).

3.2.3. Drools Planner

3.2.3.1. Construction heuristics

As an alternative to writing a long, complex `StartingSolutionInitializer`, you can now simply configure a powerful construction heuristic:

```
<solver>
  ...
  <constructionheuristic>
    <constructionheuristicType>FIRST_FIT</constructionheuristicType>
  </constructionheuristic>
  <localSearch>
    ...
  </localSearch>
</solver>
```

Planner already supports:

- First Fit
- First Fit Decreasing (this is how most of the examples implemented their StartingSolutionInitializer)
- Best Fit
- Best Fit Decreasing

Future versions will support more construction heuristics.

3.2.3.2. Phasing

Planner can now run several solver phases sequentially: each phase is a different optimization algorithm. For example: run First Fit Decreasing, then Simulated Annealing, then Tabu Search:

```
<solver>
  ...
  <constructionheuristic>
    ... <!-- First Fit Decreasing -->
  </constructionheuristic>
  <localSearch>
    ... <!-- Simulated Annealing -->
  </localSearch>
  <localSearch>
    ... <!-- Tabu Search -->
  </localSearch>
</solver>
```

3.2.3.3. Real-time planning

Planner now supports real-time planning. Real-time planning means that the planning problem can change up to a few milliseconds before a solution needs to be executed.

If one of the planning facts change while a planning problem is being solved, Planner can now process such a planning fact change and incrementally continue from the best solution found before that planning fact change. In practice, this means it can find a good solution for a big planning problem only a few milliseconds after a planning fact changes.

Planner's documentation now covers several common techniques such as backup planning, continuous planning and real-time planning.

3.2.3.4. Documentation expanded

The documentation now includes:

- detailed documentation on planning entity and planning variable
- an optimization algorithms overview
- guidelines on what's the easiest path to get started
- Info on common pitfalls and solutions

The `NQueens` examples has been refactored to feel more like a real-world example.

3.2.3.5. Logging revised

INFO and DEBUG logging is far less verbose. Use TRACE logging to see everything. The log is also now easier to read.

3.2.4. Drools Integration

3.2.4.1. Event Listeners

Drools supports adding 3 types of listeners to `KnowledgeSessions` - *AgendaListener*, *WorkingMemoryListener*, *ProcessEventListener*

The `drools-spring` module allows you to configure these listeners to `KnowledgeSessions` using XML tags. These tags have identical names as the actual listener interfaces i.e., `<drools:agendaEventListener....>`, `<drools:workingMemoryEventListener....>` and `<drools:processEventListener....>`.

`drools-spring` provides features to define the listeners as standalone (individual) listeners and also to define them as a group.

3.3. What is New and Noteworthy in Drools 5.2.0

3.3.1. Knowledge API (renamed from Drools API)

3.3.1.1. Core

3.3.1.1.1. MVEL

The MVEL dialect has been improved. We have moved all variable lookups to the new indexed factories, which should allow faster execution, as well as being simpler code. The build process for mvel has been reviewed to streamline it to avoid wasteless object creation so that the build time is faster. We still have some more improvements on this to share the ParserConfiguration which will make each MVEL compilation unit faster to initialise and use less memory, as they will share import information for each package.

It was always possible to execute with MVEL in both dynamic and strict mode, with strict mode for static type safety the default. This is configurable via the MVEL dialect configuration:

```
drools.dialect.mvel.strict = <true|false>
```

However there were some places in execution when strict mode was not enforced. Effort has now been done to ensure that type safety is enforced through out, unless "strict == false". This means that some bad code that compiled before may not compile now, because it is not type safe. For those cases where the type safety cannot be achieved at compile time we added the @typesafe annotation, discussed in it's own section.

3.3.1.1.2. Classloader

The Classloader has been improved to use a CompositeClassLoader instead of the previous hierarchical "parent" classloader. This was necessary for OSGi where each module needs it's own classpath, but reflection will not work if classloader cannot be found. Modules now add themselves to the composite classloader when first initialised. This is also exposed to the user where the previous Classloader argument on the kbase and kbuilder configuration now takes vararg of ClassLoaders, all of which are now searchable.

3.3.2. Drools Expert and Fusion

3.3.2.1. Lazy Truth Maintenance

You no longer need to enable or disable truth maintenance, via the kbase configuration. It is now handled automatically and turned on only when needed. This was done along with the code changes so that all entry points use the same code, previous to this the default entry point and named entry points used different code, to avoid TMS overhead for event processing.

3.3.2.2. Multi-function Accumulates

The accumulate CE now supports multiple functions. For instance, if one needs to find the minimum, maximum and average value for the same set of data, instead of having to repeat the accumulate statement 3 times, a single accumulate can be used.

```
rule "Max, min and average"
  when
    accumulate( Cheese( $price : price ),
                $max : max( $price ),
                $min : min( $price ),
                $avg : average( $price ) )
  then
    // do something
  end
```

3.3.2.3. Parameterized constructors for declared types

Generate constructors with parameters for declared types.

Example: for a declared type like the following:

```
declare Person
  firstName : String @key
  lastName : String @key
  age : int
end
```

The compiler will implicitly generate 3 constructors: one without parameters, one with the @key fields, and one with all fields.

```
Person() // parameterless constructor
Person( String firstName, String lastName )
Person( String firstName, String lastName, int age )
```

3.3.2.4. Type Declaration 'extends'

Type declarations now support 'extends' keyword for inheritance

In order to extend a type declared in Java by a DRL declared subtype, repeat the supertype in a declare statement without any fields.

```
import org.people.Person
```

```
declare Person
end

declare Student extends Person
    school : String
end

declare LongTermStudent extends Student
    years : int
    course : String
end
```

3.3.2.5. Free Form expressions in Constraints (New Parser)

The parser has been rewritten. We had reached the limitations of what we could achieve in pure ANTLR and moved to a hybrid parser, that adds flexibility to the language.

The main benefit with the new parser is that the language now support free form expressions for constraints and 'from' statements. So complex expressions on nested accessors, method calls etc should now all be possible as simple constraints without wrapping them with an `eval(.....)`. This was also important for us to start to move towards a single consistent grammar for both the "when" left hand side and "then" right hand side. As previously we had to document the restricted limitations of a field constraint on the LHS compared to expressions used inside of an 'eval' or used on the RHS. Complex expressions are still internally rewritten as evals, so it's just syntactic sugar.

Examples:

```
Person( age * 2 > $anotherPersonsAge + 2 ) // mathematical expressions
Person( addresses["home"].streetName.startsWith( "High Park" ) ) // method calls
    and collections simplified syntax
Person( isAdult() ) // boolean expression without relational operator
```

The new parser also support free form expressions on the "from" clause, allowing for instance, new syntaxes, like inline creation for lists:

```
Cheese( ) from [ $stilton, $brie, $provolone ] // inline list creation and
iteration
```

3.3.2.6. Rule API

A fluent API was created to allow programmatic creation of rules as an alternative to the previously suggested method of template creation.

```

PackageDescr pkg = DescrFactory.newPackage()
    .name("org.drools.example")
    .newRule().name("Xyz")
        .attribute("ruleflow-group", "bla")
    .lhs()
        .and()
            .pattern("Foo").id("$foo",
false ).constraint("bar==baz").constraint("x>y").end()
        .not().pattern("Bar").constraint("a+b==c").end().end()
    .end()
    .end()
    .rhs("System.out.println();").end()
    .getDescr();

```

3.3.2.7. Positional Arguments

Patterns now support positional arguments on type declarations.

Positional arguments are ones where you don't need to specify the field name, as the position maps to a known named field. i.e. `Person(name == "mark")` can be rewritten as `Person("mark";)`. The semicolon ';' is important so that the engine knows that everything before it is a positional argument. Otherwise we might assume it was a boolean expression, which is how it could be interpreted after the semicolon. You can mix positional and named arguments on a pattern by using the semicolon ';' to separate them. Any variables used in a positional that have not yet been bound will be bound to the field that maps to that position.

```

declare Cheese
    name : String
    shop : String
    price : int
end

```

The default order is the declared order, but this can be overridden using `@Position`

```

declare Cheese
    name : String @position(1)
    shop : String @position(2)
    price : int @position(0)
end

```

The `@Position` annotation, in the `org.drools.definition.type` package, can be used to annotate original pojos on the classpath. Currently only fields on classes can be annotated. Inheritance of classes is supported, but not interfaces or methods yet.

Example patterns, with two constraints and a binding. Remember semicolon ';' is used to differentiate the positional section from the named argument section. Variables and literals and expressions using just literals are supported in positional arguments, but not variables.

```
Cheese( "stilton", "Cheese Shop", p; )
Cheese( "stilton", "Cheese Shop"; p : price )
Cheese( "stilton"; shop == "Cheese Shop", p : price )
Cheese( name == "stilton"; shop == "Cheese Shop", p : price )
```

3.3.2.8. Backward Chaining

Drools now provides Prolog style derivation queries, as an experimental feature. What this means is that a query or the 'when' part of a rule may call a query, via a query element. This is also recursive so that a query may call itself. A query element may be prefixed with a question mark '?' which indicates that we have a pattern construct that will pull data, rather than the normal reactive push nature of patterns. If the ? is omitted the query will be executed as a live "open query" with reactivity, similar to how normal patterns work.

A key aspect of BC is unification. This is where a query parameter may be bound or unbound, when unbound it is considered an output variable and will bind to each found value.

In the example below x and y are parameters. Unification is done by subsequent bindings inside of patterns. If a value for x is passed in, it's as though the pattern says "thing == x". If a value for x is not passed in it's as though "x: thing" and x will be bound to each found thing.

Because Drools does not allow multiple bindings on the same variable we introduce ':=' unification symbol to support this.

```
declare Location
    thing : String
    location : String
end

query isContainedIn( String x, String y )
    Location( x := thing, y := location )
    or
    ( Location(z := thing, y := location) and ?isContainedIn( x := x, z := y ) )
end
```

Positional and mixed positional/named are supported.

```
declare Location
    thing : String
```



```
        location : String
    end

    query isContainedIn( String x, String y )
        Location(x, y;)
        or
        ( Location(z, y;) and ?isContainedIn(x, z;) )
    end
```

Here is an example of query element inside of a rule using mixed positional/named arguments.

```
package org.drools.test

import java.util.List
import java.util.ArrayList

dialect "mvel"

declare Here
    place : String
end

declare Door
    fromLocation : String
    toLocation : String
end

declare Location
    thing : String
    location : String
end

declare Edible
    thing : String
end

query connect( String x, String y )
    Door(x, y;)
    or
    Door(y, x;)
end

query whereFood( String x, String y )
    ( Location(x, y;) and
      Edible(x;) )
    or
    ( Location(z, y;) and
```

```
        whereFood(x, z;) )
end

query look(String place, List things, List food, List exits)
    Here(place;)
        things := List() from accumulate( Location(thing, place;),
        collectList( thing ) )
        food := List() from accumulate( ?whereFood(thing, place;),
        collectList( thing ) )
        exits := List() from accumulate( ?connect(place, exit;), collectList( exit ) )
end

rule reactiveLook
    when
        Here( $place : place)
        ?look($place, $things; $food := food, $exits := exits)
    then
        System.out.println( \"You are in the \" + $place);
        System.out.println( \" You can see \" + $things );
        System.out.println( \" You can eat \" + $food );
        System.out.println( \" You can go to \" + $exits );
    end
end
```

As previously mentioned you can use live "open" queries to reactively receive changes over time from the query results, as the underlying data it queries against changes. Notice the "look" rule calls the query without using '?'.

```
query isContainedIn( String x, String y )
    Location(x, y;)
    or
    ( Location(z, y;) and isContainedIn(x, z;) )
end

rule look when
    Person( $l : likes )
    isContainedIn( $l, 'office'; )
then
    insertLogical( $l + 'is in the office' );
end
```

Literal expressions can be passed as query arguments, but at this stage you cannot mix expressions with variables.

It is possible to call queries from Java leaving arguments unspecified using the static field `org.drools.runtime.rule.Variable.v` - note you must use 'v' and not an alternative instance of `Variable`. The following example will return all objects contained in the office.

```

results = ksession.getQueryResults( "isContainedIn", new Object[] { Variable.v,
    "office" } );
l = new ArrayList<List<String>>();
for ( QueryResultsRow r : results ) {
    l.add( Arrays.asList( new String[] { (String) r.get( "x" ), (String)
        r.get( "y" ) } ) );
}

```

The algorithm uses stacks to handle recursion, so the method stack will not blow up.

3.3.2.9. Non Typesafe Classes

@typesafe(<boolean>) has been added to type declarations. By default all type declarations are compiled with type safety enabled; @typesafe(false) provides a means to override this behaviour by permitting a fall-back, to type unsafe evaluation where all constraints are generated as MVEL constraints and executed dynamically. This can be important when dealing with collections that do not have any generics or mixed type collections.

3.3.2.10. Session Reports

Added experimental framework to inspect a session and generate a report, either based on a predefined template or with a user created template.

```

// Creates an inspector
SessionInspector inspector = new SessionInspector( ksession );
// Collects the session info
StatefulKnowledgeSessionInfo info = inspector.getSessionInfo();
// Generate a report using the "simple" report template
String report = SessionReporter.generateReport( "simple", info, null );

```

3.3.3. Drools and jBPM integration

3.3.3.1. Improved Camel integration

Camel integration using the Drools EndPoint was improved with the creation of both DroolsConsumer and DroolsProducer components. Configurations were added to support the insertion of either Camel's Exchange, Message or Body into the Drools session, allowing for the easy development of dynamic content based routing applications. Also, support to entry points was added.

Examples of routes:

```

from( "direct:test-no-ep" ).to( "drools://node/ksession1?action=insertBody" );

```

```
from(          "direct:test-with-ep"          ).to(          "drools://node/ksession1?  
action=insertBody&entryPoint=ep1" );  
from(          "direct:test-message"          ).to(          "drools://node/ksession1?  
action=insertMessage" );  
from(          "direct:test-exchange"         ).to(          "drools://node/ksession1?  
action=insertExchange" );
```

3.3.4. Merging Drools Flow into jBPM5

The Drools Flow project and the jBPM project have been merged into the the newest version of the jBPM project, called jBPM5. jBPM5 combines the best of both worlds: merging the experience that was built up with the jBPM project over several years in supporting stable, long-living business processes together with the improvements that were prototyped as part of Drools Flow to support more flexible and adaptive processes. Now that jBPM 5.0 has been released, the Drools project will be using jBPM5 as the engine to support process capabilities. Drools Flow as a subproject will no longer exist, but its vision will continue as part of the jBPM project, still allowing (optional but) advanced integration between business rules, business processes and complex event processing and a unified environment for all three paradigms.

The impact for the end user however should be minimal, as the existing (knowledge) API is still supported, the underlying implementation has just been replaced with a newer version. All existing features should still be supported, and many more.

For more information, visit <http://www.jboss.org/jbpm>

3.3.5. Guvnor

3.3.5.1. Guvnor Look & Feel Moving Closer To Native GWT Look

We have removed GWT-Ext from Guvnor and now only use GWT.

3.3.5.2. Embed Guvnor Editors

Embed Guvnor Editor's in external applications. You can create or edit assets like Business Rules, Technical Rules, DSL definitions and Decision Tables in your applications using Guvnor's specific editors. You can even edit multiple assets at once.

3.3.5.3. Annotations come to Declarative Models

The ability to add annotations in Guvnor to declarative models has been added. This allows Fact Types to be defined as events.

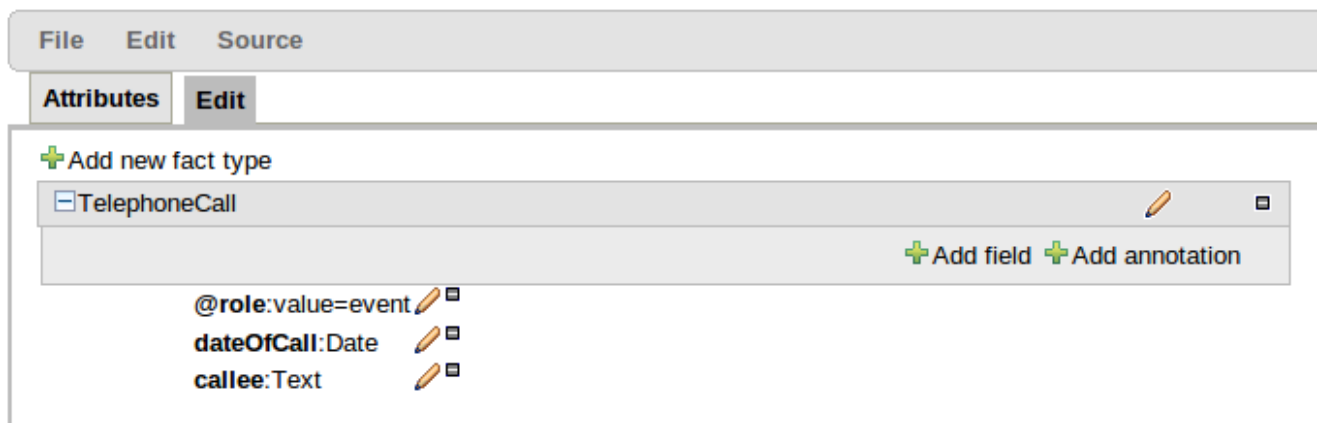


Figure 3.35. Annotations



Figure 3.36. Annotation editor

3.3.5.4. Support for Complex Event Processing in the guided editors

The guided editors have been enhanced to allow full use of Drools Fusion's Complex Event Processing operators, sliding windows and entry-points.

File Edit Source

Attributes Edit

WHEN

1.

There is a TelephoneCall [\$mtc] with:
dateOfCall[\$dc] greater than 23-May-2011
<no window>

2.

All TelephoneCall with:
this before 31-Dec-2011
this after 10s \$mtc
over window:length 10
From Entry Point telephone-calls

THEN
(show options...)

Figure 3.37. Complex Event Processing

3.3.5.5. Decision Tables

The existing Guided Decision Table has been replaced to provide a foundation on which to build our future guided Decision Table toolset. The initial release largely provides an equivalent feature-set to the obsolete Guided Decision Table with a few improvements, as explained in more detail below. A change from the legacy table was essential for us to begin to realise our desire to provide the number one web-based decision table available. With this foundation we will be able to expand the capabilities of our Guided Decision Table toolset to provide a feature rich, user-friendly environment.

Find Business rule assets [mortgages] Pricing loans

Decision table

Condition columns

- amount min
- amount max
- period
- income
- deposit max
- New column

Action columns

- Loan approved
- LMI
- rate
- New column

(options)

Add Attribute/Metadata:

#	Description	amount min	amount max	period	deposit max	income	Loan approved	LMI	rate
		LoanApplication [application]				IncomeSource			
		amount [>]	amount [<=]	lengthYears [==]	deposit [<]	type [==]			
+	1	131000	200000	30	20000	Asset	true	0	2
+	2	10000	100000	20	2000	Job	true	0	4
+	3	100001	130000	20	3000	Job	true	10	6

Add Row

rule: Pricing loans

Pricing loans

Other meta data ...

Subject:

Type:

External link:

Source:

Version history ...

Figure 3.38. Basic Decision Table

3.3.5.5.1. Cell Merging

Adjacent cells in the same column with the same value can be merged thus eliminating the need for otherwise cluttered views of multiple rows containing the same value.

Find Business rule assets [mortgages] Pricing loans

Decision table

Condition columns

- amount min
- amount max
- period
- deposit max
- income
- Application Date
- New column

Action columns

- Loan approved
- LMI
- rate
- New column

(options)

Add Attribute/Metadata:

#	Description	amount min	amount max	period	deposit max	income	Application Date	Loan approved	LMI
		LoanApplication [application]				IncomeSource	Applicant [\$a]		
		amount [>]	amount [<=]	lengthYears [==]	deposit [<]	type [==]	applicationDate [==]		
1		100001	130000	20	3000	Job	2011-02-17	true	10
2		10000	100000	20	2000	Job	2011-02-16		0
3		131000	200000	30	20000	Asset	2011-02-15		0

Figure 3.39. Merged Decision Table

3.3.5.5.2. Typed-columns

The major basic data-types (numeric, date, text and Boolean) are handled as such and respond as you'd expect to sorting.

Find Business rule assets [mortgages] Pricing loans

Decision table

Condition columns

- amount min
- amount max
- period
- deposit max
- income
- Application Date
- New column

Action columns

- Loan approved
- LMI
- rate
- New column

(options)

Add Attribute/Metadata:

#	Description	amount min	amount max	period	deposit max	income	Application Date	Loan approved	LMI
		LoanApplication [application]				IncomeSource	Applicant [\$a]		
		amount [>]	amount [<=]	lengthYears [==]	deposit [<]	type [==]	applicationDate [==]		
1		100001	130000	20	3000	Job	2011-02-17	true	10
2		10000	100000	20	2000	Job	2011-02-16	true	0
3		131000	200000	30	20000	Asset	2011-02-15	true	0

Figure 3.40. Datatype Sorting

3.3.5.5.3. Improved header

The table header section has been improved to show more verbose information to facilitate design-time understanding. The table has a fixed header that remains as you'd expect, at the top of the table, whilst scrolling larger decision tables.

Find

Business rule assets [mortgages]

Pricing loans

File

Edit

Source

Decision table

		amount min	amount max	period	deposit max	income	Application Date			
	tion	LoanApplication [application]				IncomeSource	Applicant [\$a]	Loan approved	LMI	rate
		amount [>]	amount [<=]	lengthYears [==]	deposit [<]	type [==]	applicationDate [==]			
+										
+										
+										
+										
+										
+										
+										
+										
+										
+										
+		10000	100000	20	2000	Job	2011-02-16			4
+		131000	200000	30	20000	Asset	2011-02-15	true	0	2

Add Row

Figure 3.41. Scrolling Decision Table

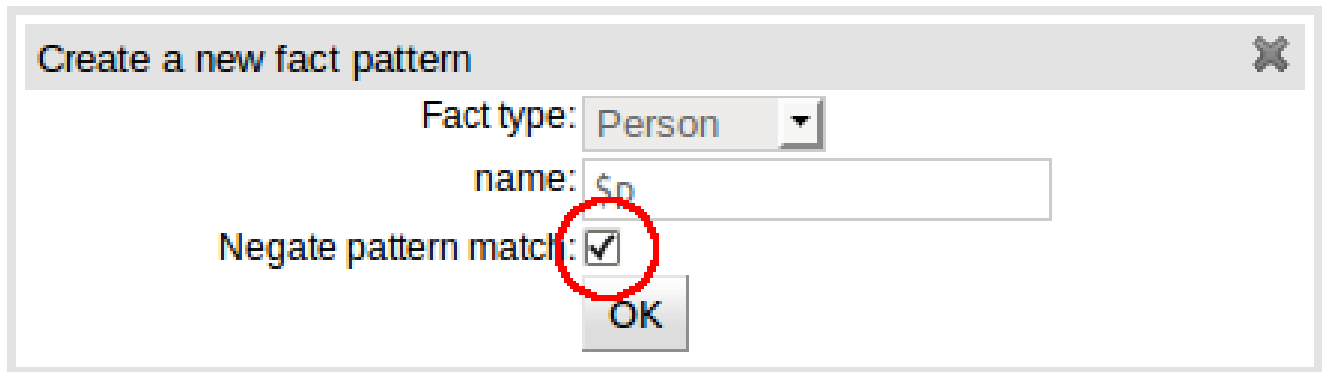
3.3.5.5.4. Tighter integration

The decision table responds better to changes made to column definitions; so changes to column Fact Type, Fact Type field, calculation type, value list etc are reflected in the table itself.

3.3.5.5.5. Negation of Fact patterns

Fact Patterns in condition columns can be negated to match when the defined pattern does not exist in WorkingMemory. In essence it is now possible to construct rules within the decision table equivalent to the following DRL fragment:-

```
not Cheese( name == "Cheddar" )
```



Create a new fact pattern

Fact type: Person

name: \$p

Negate pattern match: ☒

OK

Figure 3.42. Negation of Fact Patterns

3.3.5.5.6. Negation of rules

Entire rules can be negated, giving rise to DRL frgements such as:-

```
rule "no cheddar liked by Fred"
  when
    not (
      $c : Cheese( name == "Cheddar" )
      Person( name == "Fred", favouriteCheese == $c )
    )
  then
    ..
  end
```

Add an option to the rule

⚙

Add an option to the rule

Metadata:

Attribute:

Choose...

Choose...
salience
enabled
date-effective
date-expires
no-loop
agenda-group
activation-group
duration
auto-focus
lock-on-active
ruleflow-group
dialect
negate

Add an option to the rule

(options)

Add Attribute/Metadata: +

Attributes:

☒ negate

 Default value:

☐ Hide this column

	#	Description	negate	name	age	age
+ □	1		<input checked="" type="checkbox"/>	Bill	30	12345
+ □	2		<input type="checkbox"/>	Ben	<otherwise>	12345
+ □	3		<input type="checkbox"/>	Weed	40	12345
+ □	4		<input type="checkbox"/>	<otherwise>	50	12345

Figure 3.43. Negation of rules

3.3.5.5.7. Support for "otherwise"








Condition columns containing literal values that use the equality (==, equal to) or inequality (!=, not equal to) operators can now contain a meta-value for "otherwise" which represents all other values not explicitly defined in the column. This feature gives rise to DRL fragments such as the following:-

```
#from row number: 1
rule "Row 1 dtable"
    salience 1
    dialect "mvel"
when
    $p : Person( name == "Bill" , age != "30" )
then
    $p.setAge( 12345 );
end

#from row number: 2
rule "Row 2 dtable"
    salience 2
    dialect "mvel"
when
    $p : Person( name == "Ben" , age in ( "30", "40", "50" ) )
then
    $p.setAge( 12345 );
end

#from row number: 3
rule "Row 3 dtable"
    salience 3
    dialect "mvel"
when
    $p : Person( name == "Weed" , age != "40" )
then
    $p.setAge( 12345 );
end

#from row number: 4
rule "Row 4 dtable"
    salience 4
    dialect "mvel"
when
    $p : Person( name not in ( "Bill", "Ben", "Weed" ) , age != "50" )
then
    $p.setAge( 12345 );
end
```

		#	Description	name	age	
				Person [\$p]		age
				name [==]	age [!=]	
						
		1		Bill	30	12345
		2		Ben	<otherwise>	12345
		3		Weed	40	12345
		4		<otherwise>	50	12345

Add row...

Otherwise

Figure 3.44. Support for "otherwise"

3.3.5.5.8. Enhanced Package's Report

Templates Rules and Decision Tables rules are now included in the package's report.

3.3.5.5.9. Spring Context Editor

Now it is possible to create and mange Spring Context files inside Guvnor. These Context Files are exposed through URLs so external applications can use them.



Figure 3.45. Editing Spring Context

3.3.5.5.10. Configuring Multiple Guvnor Instances In a Jackrabbit Cluster

We added a new task in drools-ant which helps with configuring multiple Guvnor instances to be able to share their Jackrabbit content.

3.3.5.5.11. Configuring Guvnor to use an external RDBMS made easier

The default Guvnor repository configuration uses embedded Derby databases which writes the workspace and version information to the local file system. This is not always optimal for a production system where it makes sense to use an external RDBMS.

We added a new section under the "Administration" tab called "Repository Configuration" which helps generate the repository.xml configuration file for a number of databases (Microsoft SQL Server, MySQL, Oracle, PostgreSQL, Derby, H2)

3.3.5.5.12. Web-based BPMN2 authoring in Guvnor

We have completed big parts of the integration between Guvnor and the Oryx web-based business process editor. Our primary use cases supported by this integration are:

- Viewing existing jBPM5 processes in Guvnor
- Prototyping new jBPM5 processes in Guvnor.

Please note that we are still working on full round-tripping support between the web-based Oryx editor and our BPMN2 support provided through the eclipse plugin

3.3.6. Eclipse

3.3.6.1. Removal of BRL Guided Editor

The BRL Guided Editor has been removed due to lack of interest and it falling behind. Its removal allows more focus on the GWT based Guided Editor in Guvnor. The DRL, text-based, Guided Editor remains unaffected.

3.3.7. Maven artifactId changes

A couple of maven artifacts (jars, wars, ...) have been renamed so it is more clear what they do. Below is the list of the GAV changes, adjust your `pom.xml` files accordingly when upgrading to the new version.

Table 3.1. Maven GAV changes overview

Old groupId	Old artifactId	New groupId	New artifactId
org.drools	drools (the parent pom)	org.drools	droolsjbpm-parent
org.drools	drools-api	org.drools	knowledge-api
org.drools	drools-docs-introduction	org.drools	droolsjbpm-introduction-docs
org.drools	drools-examples-drl	org.drools	drools-examples
org.drools	drools-examples-fusion / drools-examples-drl (jBPM using parts)	org.drools	droolsjbpm-integration-examples
org.drools	drools-docs-expert	org.drools	droolsjbpm-expert-docs
org.drools	drools-docs-fusion	org.drools	droolsjbpm-fusion-docs
org.drools	drools-repository	org.drools	guvnor-repository
org.drools	drools-ide-common	org.drools	droolsjbpm-ide-common
org.drools	drools-guvnor	org.drools	guvnor-webapp
org.jboss.drools.guvnor.tools	guvnor-importer	org.drools	guvnor-bulk-importer
org.drools	drools-docs-guvnor	org.drools	guvnor-docs
org.drools	drools-server	org.drools	drools-camel-server
org.drools	drools-docs-integration	org.drools	droolsjbpm-integration-docs
org.drools	drools-flow-core	org.jbpm	jbpm-flow

Old groupId	Old artifactId	New groupId	New artifactId
org.drools	drools-flow-compiler	org.jbpm	jbpm-flow-builder
org.drools	drools-bpmn2	org.jbpm	jbpm-bpmn2
org.drools	drools-flow-persistence-jpa	org.jbpm	jbpm-persistence-jpa
org.drools	drools-bam	org.jbpm	jbpm-bam
org.drools	drools-process-task	org.jbpm	jbpm-human-task
org.drools	drools-gwt-console	org.jbpm	jbpm-gwt-console
org.drools	drools-gwt-form	org.jbpm	jbpm-gwt-form
org.drools	drools-gwt-graph	org.jbpm	jbpm-gwt-graph
org.drools	drools-gwt-war	org.jbpm	jbpm-gwt-war
org.drools	drools-gwt-server-war	org.jbpm	jbpm-gwt-server-war
org.drools	drools-workitems	org.jbpm	jbpm-workitems
org.drools	drools-docs-flow	org.jbpm	jbpm-docs

For example: before, in your `pom.xml` files, you declared `drools-api` like this:

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-api</artifactId>
  ...
</dependency>
```

And now, afterwards, in your `pom.xml` files, you declare `knowledge-api` like this instead:

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>knowledge-api</artifactId>
  ...
</dependency>
```

3.4. What is New and Noteworthy in Drools 5.1.0

3.4.1. Drools API

As in Drools 5.0 it is still possible to configure a `KnowledgeBase` using configuration, via a xml change set, instead of programmatically. However the change-set namespace is now versioned. This means that for Drools 5.1, the 1.0.0 xsd should be referenced.

Example 3.6. Here is a simple version 1.0.0 change set

```
<change-set xmlns='http://drools.org/drools-5.0/change-set'
xmlns:xs='http://www.w3.org/2001/XMLSchema-instance'
xs:schemaLocation='http://drools.org/drools-5.0/change-set change-set-5.0.xsd
main/resources/change-set-1.0.0.xsd' >
  <add>
    <resource source='classpath:org/domain/someRules.drl' type='DRL' />
    <resource source='classpath:org/domain/aFlow.drf' type='DRF' />
  </add>
</change-set>
```

3.4.2. Core

3.4.2.1. JMX Monitoring

JMX monitoring was added to support KnowledgeBase monitoring. This is specially important for long running processes like the ones usually required for event processing. Initial integration with JOPR was also added. JMX can be enabled with using the properties setting the knowledge base:

drools.mbeans = <enabled|disabled>

or this option at runtime

kbaseConf.setOption(MBeansOption.ENABLED)

3.4.2.2. Spring

Drools now has extensive Spring support, the XSD can be found in the the drools-spring jar. The namespace is "http://drools.org/schema/drools-spring"

Example 3.7. KnowledgeBuilder example

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:drools="http://drools.org/schema/drools-spring"
xmlns:camel="http://camel.apache.org/schema/spring"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://
www.springframework.org/schema/beans/spring-beans-2.0.xsd
http://drools.org/schema/
drools-spring http://anonsvn.jboss.org/repos/labs/labs/jbossrules/trunk/drools-
container/drools-spring/src/main/resources/org/drools/container/spring/drools-
spring-1.0.0.xsd
http://camel.apache.org/schema/spring http://
camel.apache.org/schema/spring/camel-spring.xsd">
```

```
<drools:resource id="resource1" type="DRL" source="classpath:org/drools/
container/spring/testSpring.drl"/>

<drools:kbase id="kbase1">
  <drools:resources>
    <drools:resource type="DRL" source="classpath:org/drools/container/spring/
testSpring.drl"/>
    <drools:resource ref="resource1"/>
    <drools:resource source="classpath:org/drools/container/spring/
IntegrationExampleTest.xls" type="DTABLE">
      <drools:decisiontable-conf input-type="XLS" worksheet-name="Tables_2" />
    </drools:resource>
  </drools:resources>

  <drools:configuration>
    <drools:mbeans enabled="true" />
    <drools:event-processing-mode mode="STREAM" />
  </drools:configuration>
</drools:kbase>
</beans>
```

KnowledgeBase takes the following configurations: "advanced-process-rule-integration, multithread, mbeans, event-processing-mode, accumulate-functions, evaluators and assert-behavior".

From the the kbase reference ksessions can be created

Example 3.8. Knowledge Sessions

```
<drools:ksession id="ksession1" type="stateless" name="stateless1" kbase="kbase1" /
>

<drools:ksession id="ksession2" type="stateful" kbase="kbase1" />
```

Like KnowledgeBases Knowledge sessions can take a number of configurations, including "work-item-handlers, "keep-references", "clock-type", "jpa-persistence".

Example 3.9. Knowledge Sessions Configurations

```
<drools:ksession id="ksession1" type="stateful" kbase="kbase1" >
  <drools:configuration>
    <drools:work-item-handlers>
      <drools:work-item-handler name="handlername" ref="handlerid" />
    </drools:work-item-handlers>
    <drools:keep-reference enabled="true" />
    <drools:clock-type type="REALTIME" />
  </drools:configuration>
</drools:ksession>
```

```

    </drools:configuration>
</drools:ksession>

```

StatefulKnowledgeSessions can be configured for JPA persistence

Example 3.10. JPA configuration for StatefulKnowledgeSessions

```

<bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="org.h2.Driver" />
  <property name="url" value="jdbc:h2:tcp://localhost/DroolsFlow" />
  <property name="username" value="sa" />
  <property name="password" value="" />
</bean>

<bean id="myEmf" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="dataSource" ref="ds" />
  <property name="persistenceUnitName" value="org.drools.persistence.jpa.local" />
</bean>

<bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">
  <property name="entityManagerFactory" ref="myEmf" />
</bean>

<drools:ksession id="jpaSingleSessionCommandService" type="stateful" kbase="kbase1">
  <drools:configuration>
    <drools:jpa-persistence>
      <drools:transaction-manager ref="txManager" />
      <drools:entity-manager-factory ref="myEmf" />
      <drools:variable-persisters>
        <drools:persister for-
implementation="org.drools.persistence.instance.processinstance.persisters.JPAVariablePersister"/
        >
        <drools:persister for-
implementation="org.drools.persistence.instance.processinstance.persisters.JPAVariablePersister"/
        >
        <drools:persister for-
implementation="org.drools.persistence.instance.processinstance.persisters.JPAVariablePersister"/
        >
      </drools:variable-persisters>
    </drools:jpa-persistence>
  </drools:configuration>
</drools:ksession>

```

Knowledge Sessions can support startup batch scripts, previous versions used the "script" element name, this will be updated to "batch". The following commands are supported: "insert-

object", "set-global", "fire-all-rules", "fire-until-halt", "start-process", "signal-event". Anonymous beans or named "ref" attributes may be used.

Example 3.11. Startup Batch Commands

```
<drools:ksession id="jpaSingleSessionCommandService" type="stateful" kbase="kbase1">
  <drools:script>
    <drools:insert-object ref="person1" />
    <drools:start-process process-id="proc name">
      <drools:parameter identifier="varName" ref="varRef" />
    </drools:start-process>
    <drools:fire-all-rules />
  </drools:script>
</drools:ksession>
```

ExecutionNodes are supported in Spring , these provide a Context of registered ksessions; this can be used with Camel to provide ksession routing.

Example 3.12. Execution Nodes

```
<execution-node id="node1" />

<drools:ksession id="ksession1" type="stateless" name="stateless" kbase="kbase1" node="node1" />
>

<drools:ksession id="ksession2" type="stateful" kbase="kbase1" node="node1"/>
```

3.4.2.3. Camel

Spring can be combined with Camel to provide declarative rule services. a Camel Policy is added from Drools which provides magic for injecting the ClassLoader used by the ksession for any data formatters, it also augments the Jaxb and XStream data formatters. In the case If Jaxb it adds additional Drools related path info and with XStream it registers Drools related converters and aliases.

You can create as many endpoints as you require, using different addresses. The CommandMessageBodyReader is needed to allow the payload to be handled by Camel.

Example 3.13. Rest Endpoint Configuration

```
<cxfrsServer id="rsServer"
  address="/kservice/rest"
  serviceClass="org.drools.jax.rs.CommandExecutorImpl">
```

```

<cxf:providers>
  <bean class="org.drools.jax.rs.CommandMessageBodyReader" />
</cxf:providers>
</cxf:rsServer>

```

Camel routes can then be attached to CXF endpoints, allowing you control over the payload for things like data formatting and executing against Drools ksessions. The DroolsPolicy adds some smarts to the route. If JAXB or XStream are used, it would inject custom paths and converters, it can also set the classloader too on the server side, based on the target ksession. On the client side it automatically unwraps the Response object.

This example unmarshalls the payload using an augmented XStream DataFormat and executes it against the ksession1 instance. The "node" there refers to the ExecutionContext, which is a context of registered ksessions.

Example 3.14. Camel Route

```

<bean id="droolsPolicy" class="org.drools.camel.component.DroolsPolicy" />

<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="cxfrs://bean://rsServer"/>
    <policy ref="droolsPolicy">
      <unmarshal ref="xstream" />
      <to uri="drools://node/ksession1" />
      <marshal ref="xstream" />
    </policy>
  </route>
</camelContext>

```

The Drools endpoint "drools:node/ksession1" consists of the execution node name followed by a separator and optional knowledge session name. If the knowledge session is not specified the route will look at the "lookup" attribute on the incoming payload instance or in the head attribute "DroolsLookup" to find it.

3.4.2.4. Drools Server

Spring, Camel and CXF can be combined for declarative services, drools-server is a .war that combines these with some sample xml that works out of the box to get you started, acting like a sort of template. If you are using the war in JBoss container you'll need to add this component, <http://camel.apache.org/camel-jboss.html>. The war includes a test.jsp showing an echo like example to get you started. This example just executes a simple "echo" type application. It sends a message to the rule server that pre-appends the word "echo" to the front and sends it back. By default the message is "Hello World", different messages can be passed using the url parameter msg - test.jsp?msg="My Custom Message".

3.4.2.5. Knowledge Agent Incremental Change Support

The new version of the Knowledge Agent supports `newInstance = false` in its configuration (incremental change-set build).

When setting this property to false, the KnowledgeAgent uses the existing KnowledgeBase references apply the incremental changes. Now KnowledgeAgent's can process monitored resource modifications in an incremental way. Modified definitions are compiled and compared against the original version. According to definition's type, the behaves in different ways:

- Rules: For rules, the Agent searches for modifications in its attributes, LHS and RHS.
- Queries: queries are always replaced on kbase whether they are modified or not.
- Other definitions: All other definitions are always replaced in kbase (like if they were modified). We expect to add better support for definition's modification detection in further versions.

The current implementation only supports the deletion of rules, queries and functions definitions. Type declarations cannot be deleted.

3.4.2.6. Session Inspection and Reporting framework

A new API based framework for runtime session inspection and reporting was introduced, allowing for better data gathering during debugging or profiling of the application. This inspection framework will become the basis of the tooling features to help providing more detailed information about the contents of each session. This api is experimental and not in drools-api for now, but feel free to play and help us improve it.

To inspect a session, one can use the following API calls:

Example 3.15. Creating a SessionInspector

```
StatefulKnowledgeSession ksession = ...

// ... insert facts, fire rules, etc

SessionInspector inspector = new SessionInspector( ksession );
StatefulKnowledgeSessionInfo info = inspector.getSessionInfo();
```

The `StatefulKnowledgeSessionInfo` instance will contain a lot of relevant data gathered during the analysis of the session. A simple example report template is provided and can be generated with the following API call:

Example 3.16. Generating a Report

```
String report = SessionReporter.generateReport( "simple", info, null );
```

3.4.3. Expert

3.4.3.1. Differential Update

Rete traditional does an update as a retract + assert, for a given fact this causes all partial matches to be destroyed, however during the assert some of which will be recreated again; because they were true before the update and still true after the update. This causes a lot of unnecessary object destruction and creation which puts more load on the Garbage Collector. Now an update is a single pass and inspects the partial matches in place avoiding the unnecessary destruction of partial matches. It also removes the need to undergo a normalisation process for events and truth maintenance; the normalisation process was where we would look at the activations retracted and activations inserted to figure out what was truly added and what was truly inserted to determine the "diff".

3.4.3.2. Channels

Exit Points have been replaced by the more aptly named channels, we felt this was more appropriate as they may be used by more than just the rule engine and not an exact opposite if Entry Points. Where entry points are explicitly related to entering a partition in the Rete network.

3.4.3.3. Live Queries

Drools has always had query support, but the result was returned as an iterable set; this makes it hard to monitor changes over time.

We have now complimented this with Live Queries, which has a listener attached instead of returning an iterable result set. These live queries stay open creating a view and publish change events for the contents of this view. So now you can execute your query, with parameters and listen to changes in the resulting view.

Example 3.17. Implementing ViewChangeListener

```
final List updated = new ArrayList();
final List removed = new ArrayList();
final List added = new ArrayList();

ViewChangeListener listener = new ViewChangeListener() {
    public void rowUpdated(Row row) {
        updated.add( row.get( "$price" ) );
    }

    public void rowRemoved(Row row) {
        removed.add( row.get( "$price" ) );
    }

    public void rowAdded(Row row) {
```

```
added.add( row.get( "$price" ) );
}
};

// Open the LiveQuery
LiveQuery query = ksession.openLiveQuery( "cheeses",
                                           new Object[] { "cheddar", "stilton" },
                                           listener );

...
...
query.dispose() // make sure you call dispose when you want the query to close
```

A Drools blog article contains an example of Glazed Lists integration for live queries,

<http://blog.athico.com/2010/07/glazed-lists-examples-for-drools-live.html>

3.4.3.4. Timers and Calendars

Rule's now support both interval and cron based timers, which replace the now deprecated duration attribute.

Example 3.18. Sample timer attribute uses

```
timer ( int: <initial delay> <repeat interval>? )
timer ( int: 30s )
timer ( int: 30s 5m )

timer ( cron: <cron expression> )
timer ( cron:* 0/15 * * * ? )
```

Interval "int:" timers follow the JDK semantics for initial delay optionally followed by a repeat interval. Cron "cron:" timers follow standard cron expressions:

Example 3.19. A Cron Example

```
rule "Send SMS every 15 minutes"
    timer (cron:* 0/15 * * * ?)
when
    $a : Alarm( on == true )
then
    channels[ "sms" ].insert( new Sms( $a.mobileNumber, "The alarm is still on" );
end
```

Calendars can now control when rules can fire. The Calendar api is modelled on [Quartz](http://www.quartz-scheduler.org/) <http://www.quartz-scheduler.org/> :

Example 3.20. Adapting a Quartz Calendar

```
Calendar weekDayCal = QuartzHelper.quartzCalendarAdapter(org.quartz.Calendar quartzCal)
```

Calendars are registered with the StatefulKnowledgeSession:

Example 3.21. Registering a Calendar

```
ksession.getCalendars().set( "week day", weekDayCal );
```

They can be used in conjunction with normal rules and rules including timers. The rule calendar attribute can have one or more comma calendar names.

Example 3.22. Using Calendars and Timers together

```
rule "weekdays are high priority"
  calendars "weekday"
  timer (int:0 1h)
when
  Alarm()
then
  send( "priority high - we have an alarm# ");
end

rule "weekend are low priority"
  calendars "weekend"
  timer (int:0 4h)
when
  Alarm()
then
  send( "priority low - we have an alarm# ");
end
```

3.4.3.5. Decision Tables (Excel)

3.4.3.5.1. Simple templating for variable length comma separated lists within cells

It is now possible to have a comma separated list of values in a cell and render those with a forall template

Example 3.23. DTable forall syntax

```
forall(<separator>?){<codesnippt>}
```

Example 3.24. DTable forall examples

```
forall(,) {propertyName == $}  
forall(&&) {propertyName == $}  
forall(||) {propertyName == $}  
forall(||) {propertyNameA == $} && forall(||){propertyNameB == $}  
etc
```

3.4.4. Flow

3.4.4.1. BPMN2

As we already announced earlier, the Drools team has decided to support the use of the upcoming BPMN 2.0 specification for specifying business processes using XML. This milestone includes a significant extension of the BPMN2 parser to support more of the BPMN2 features using Drools Flow. More specifically:

- more extensive event support: much more combinations of event types (start, intermediate and end) and event triggers (including for example error, escalation, timer, conditional and signal events), have been included, as well as (interrupting and non-interrupting) boundary events
- sub-process parameters
- diverging inclusive gateway
- etc.

BPMN2 processes have also been integrated in the entire Drools tool chain, to support the entire life cycle of the business process. This includes

- The ability to use BPMN2 processes in combination with our Eclipse tooling
- Guvnor as process repository
- web-based management using the BPM console
- auditing and debugging
- domain-specific processes
- etc.

As a result, Drools Flow is not only the first open-source process engine that supports such a significant set of BPMN2 constructs natively, our knowledge-oriented approach also allows you to easily combine your BPMN2 processes with business rules and complex event processing, all using the same APIs and tools.

3.4.4.2. Web-based Management console

Drools Flow processes can now also be managed through a web console. This includes features like managing your process instances (starting/stopping/inspecting), inspecting your (human) task list and executing those tasks, and generating reports.

This console is actually the (excellent!) work of Heiko Braun, who has created a generic BPM console that can be used to support multiple process languages. We have therefore implemented the necessary components to allow this console to communicate with the Drools Flow engine.

3.4.4.3. Pluggable Variable Persistence

Drools Flow can persist the runtime state of the running processes to a database (so they don't all need to be in memory and can be restored in case of failure). Our default persistence mechanism stores all the runtime information related to one process instance as a binary object (with associated metadata). The data associated with this process instance (aka process instance variables) were also stored as part of that binary object. This however could generate problem (1) when the data was not Serializable, (2) when the objects were too large to persist as part of the process instance state or (3) when they were already persisted elsewhere. We have therefore implemented pluggable variable persisters where the user can define how variable values are stored. This for example allows you to store variable values separately, and does support JPA entities to be stored separately and referenced (avoiding duplication of state).

3.4.4.4. Improved Process Instance Migration

Over time, processes may evolve. Whenever a process is updated, it is important to determine what should happen to the already running process instances. We have improved our support for migrating running process instances to a newer version of the process definition. Check out the Drools Flow documentation for more information.

3.4.4.5. Installation Script

The Drools build now exports an installer that simplifies installing the Eclipse plugin, Guvnor and the gwt-console. It creates and copies the necessary jars and wars and deploys them to the JBoss AS. It also includes a simple evaluation process example you can use to test your setup. For more info, download the drools installer and take a look at the readme within.

3.4.5. Guvnor

Appearance has been cleaned, for example less pop ups. Reminders for save after changes in assets and information about actions that were taken, also better error reporting if something goes wrong.

3.4.5.1. Discussions

The comments are below the "documentation" section (and of course optional) (and there is an Atom feed to them).



Figure 3.46. Realtime Discussions

A "backchannel" type connection that is kept open from the browser to allow messages to push back - this means (when enabled) that messages show up in real time (and other handy things like if something is added to a list - the list is updated).

3.4.5.2. Inbox

The inbox feature provides the ability to track what you have opened, or edited - this shows up under an "Inbox" item in the main navigator. <http://blog.athico.com/2009/09/inbox-feature-to-track-recent-changes.html>

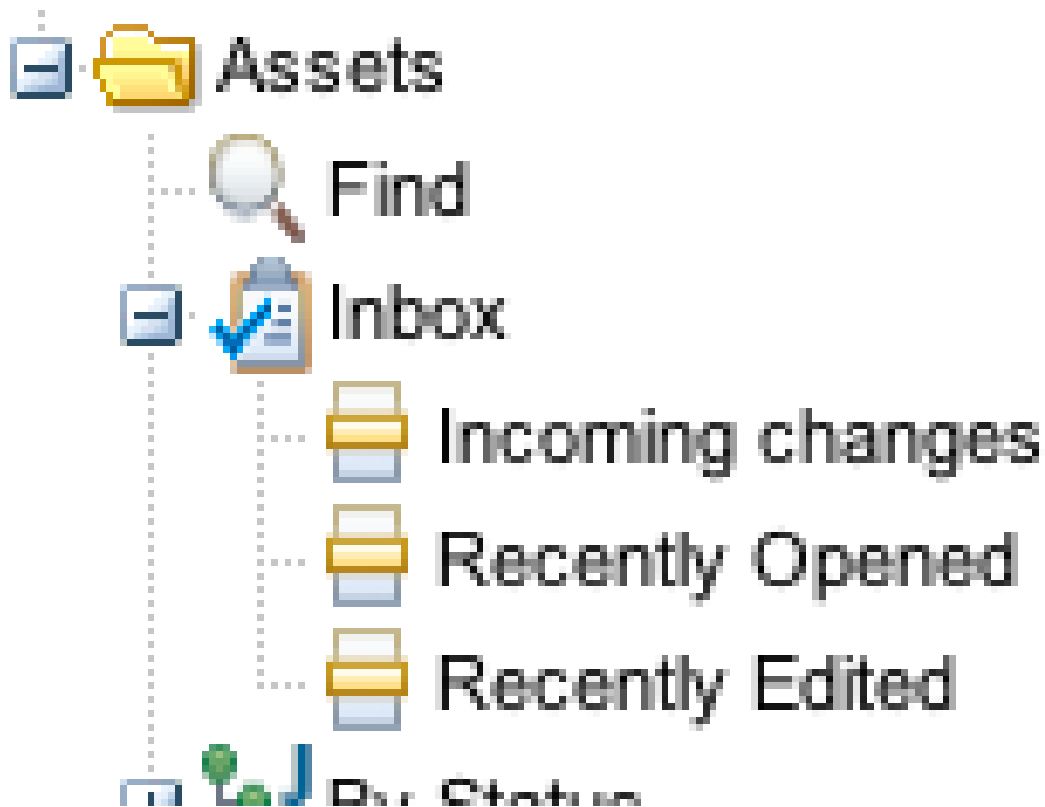


Figure 3.47. Inbox Categories

- **Recently Opened**
 - Clicking on the recently opened item will open a listing of all items you have "recently" opened (it tracks a few hundred items that you were last to look at).
- **Recently Edited**
 - Any items that you save changes to, or comment on will show up here, once again.
- **Incoming changes**
 - This tracks changes made by *other people* to items that are in *your* "Recently Edited" list. When you open these items they then are removed from this list (but remain in your Recently Edited list).

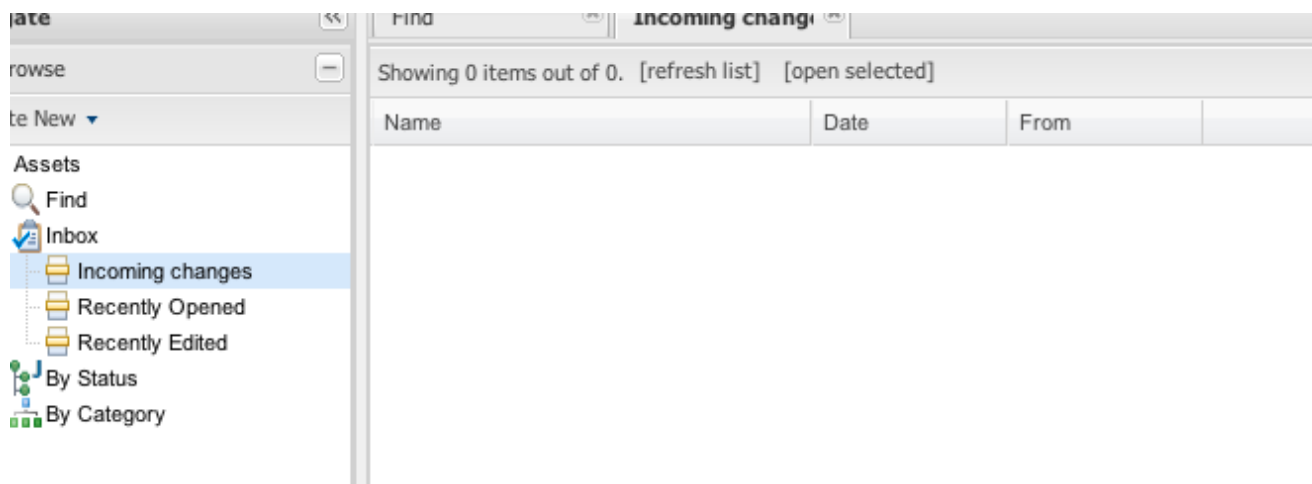


Figure 3.48. Inbox Item Lists

3.4.5.3. Bulk Importer

The Guvnor-Importer is a maven build tool that recurses your rules directory structure and constructs an xml import file that can be manually imported into the Drools-Guvnor web interface via the import/export administration feature.

3.4.5.4. DroolsDoc

PDF document containing information about the package and each DRL asset. DroolsDoc for knowledge package can be downloaded from package view under "Information and important URLs"

3.4.5.5. Update to GWT 2.0

GWT was updated, making Guvnor faster.

3.4.5.6. Build in Selector

The built in selector allows user to choose what assets to build according to:

- Status (eg, Dev, QA etc)
- Category
- Metadata

3.4.5.7. Single Asset Verification

It is possible to verify just the asset you are working on (ruleflow, rule, decision table). Verification finds issues like conflicting restrictions in a rule or redundant rows in decision tables.

3.4.5.8. Global Area

Assets stored in Global area can be shared to all packages.

3.4.5.9. Diff Check between Snapshots

Lists the changes between two snapshots.

3.4.5.10. View Multiple Assets in One Tab

Makes it possible to open more than one asset into one view. All the assets can be saved and edited as a group.

3.4.5.11. From/Collect/Accumulate support

Guided Editor has basic support for From, Accumulate and Collect Patterns. You can add any of these structures as regular Patterns. New expression builder component was created to add support for nested method calls of a variable. Using the “plus” button at the top of rule’s WHEN section or using the new “Add after” button present in every Pattern will open the popup to add new conditional elements to your rule. In the list of possible elements you will find three new entries: “From”, “From Accumulate” and “From Collect”.

When you add a new “From” element, you will see something like the image below in the guided editor. The left pattern of the “From” conditional element is a regular Pattern. You can add there any type of conditional element you want. The right section of the “From” pattern is an expression builder.

WHEN

1. There is a Hospital [\$h]
2. There is a Bed with:
 - status equal to Break Not Set
 - From \$h.beds. Choose...

THEN

(show options...)

Figure 3.49. From CE Builder

When using 'from collect' In the left pattern you can choose from “java.util.Collection”, “java.util.List” or “java.util.Set” Fact Types. This Fact Types will be automatically included in the package’s Fact Types list.

The right pattern of the collect conditional element could be one of this patterns:

- Fact Type Pattern
- Free Form Expression
- From Pattern
- From Collect Pattern
- From Accumulate Pattern

The screenshot displays the 'WHEN' section of the Drools CE Builder. It contains two conditional elements:

- 1. There is a Hospital [\$h]**
 - There is a java.util.Set with:
 - size **greater than** 0
- 2. From Collect**
 - All Bed with:
 - status **equal to** Break Not Set

Below the 'WHEN' section is the 'THEN' section, which is currently empty and includes a '(show options...)' link. Green plus icons are visible on the right side of the interface for adding new elements.

Figure 3.50. From Collect CE Builder

When using 'from accumulate' The left pattern could be any Fact Type Pattern. The right section of this conditional element is splitted in two:

WHEN

1. There is a Hospital [\$h]
2. All Bed [\$b] with:
 - doubleValue greater than 0
 - status equal to Break Not Set

Custom Code Function

Function: sum (\$b)

THEN
(show options...)

Figure 3.51. From Accumulate CE Builder

The left pattern could be any Fact Type Pattern. The right section of this conditional element is splitted in two:

- Source Pattern: (Bed \$n, in the screenshot) could be any Fact Type, From, Collect or Accumulate pattern.
- Accumulate function: Here you will find a tabbed panel where you can enter an accumulate function (sum()) in the screenshot) or you can create an online custom function using the “Custom Code” tab.

3.4.5.12. Rule Templates

Rule Templates allow the Guided editor to be used to build complex rules that can then be authored easily through a spreadsheet's tabular data metaphor. Instead of a field's value, simply mark it as a named "Template Key" and that key is available as a column in the grid. Each row will be applied to the rule template to generate a rule.

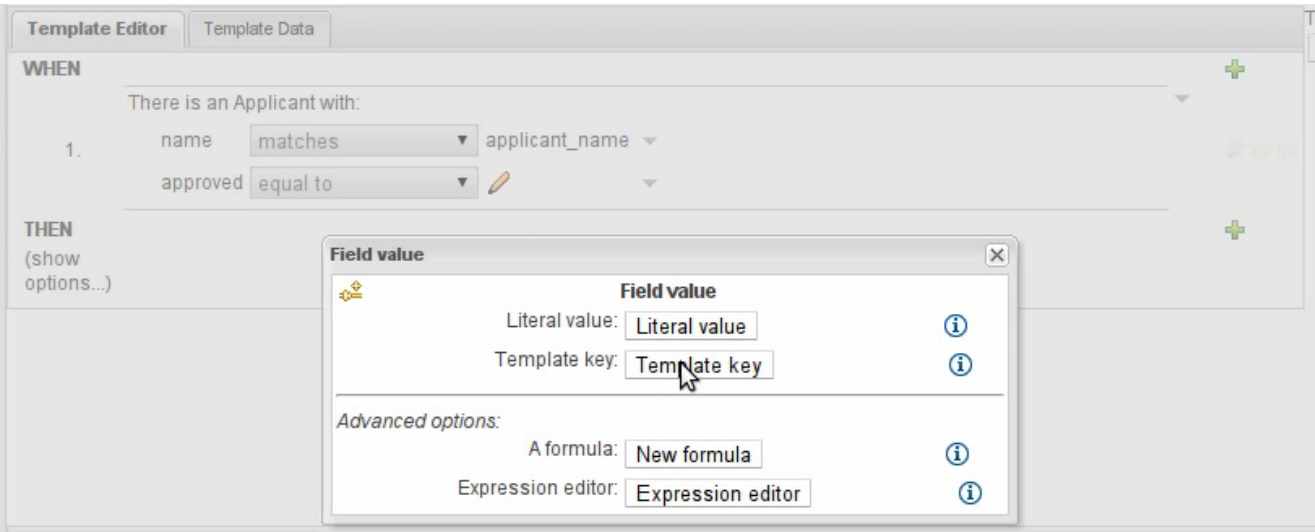


Figure 3.52. Adding a Template Key in the 'WHEN' Section

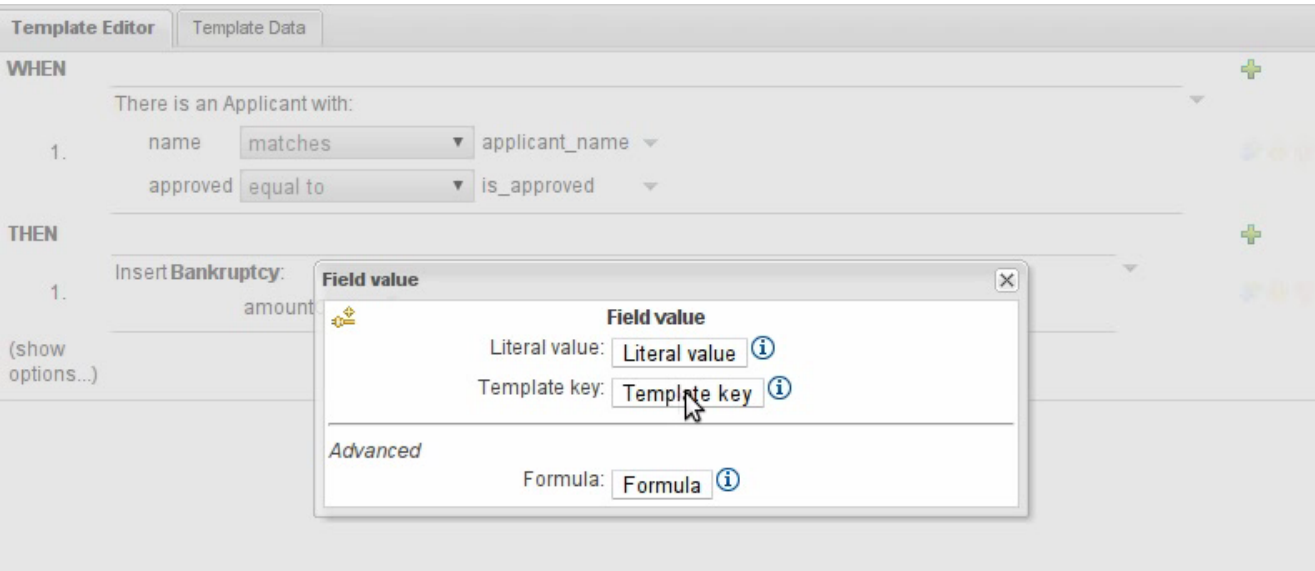


Figure 3.53. Adding a Template Key in the "THEN" Section

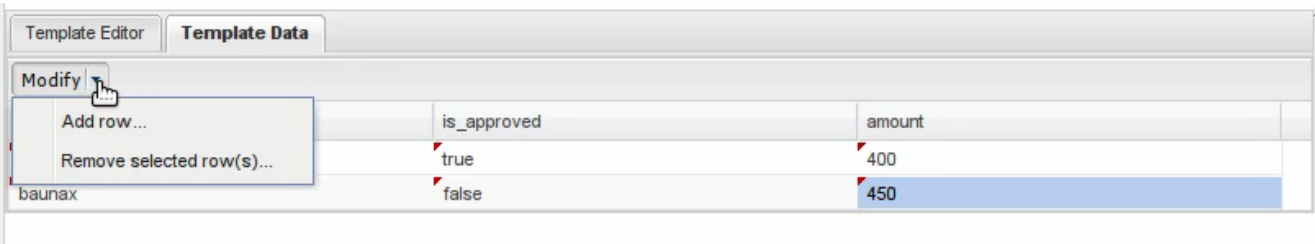


Figure 3.54. Populating Rows against those Template Keys

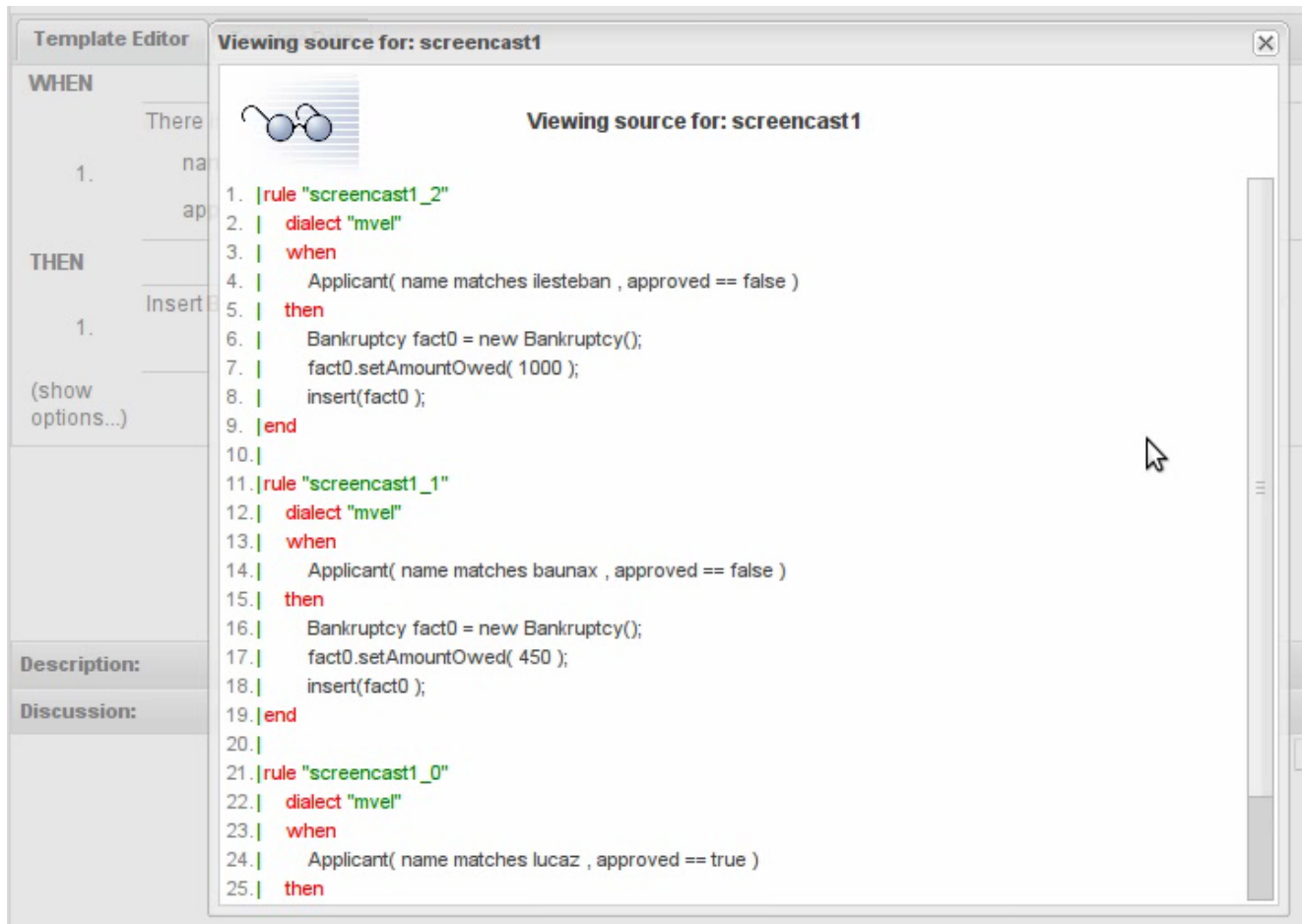


Figure 3.55. Each Row generates a Rule

3.4.5.13. Working Sets

When modelling rules the user gets exposed to all the fact types which can be a bit over whelming. Working Sets allow related fact types can be grouped together, provided a more managable view of selecting fact types, when authoring rules

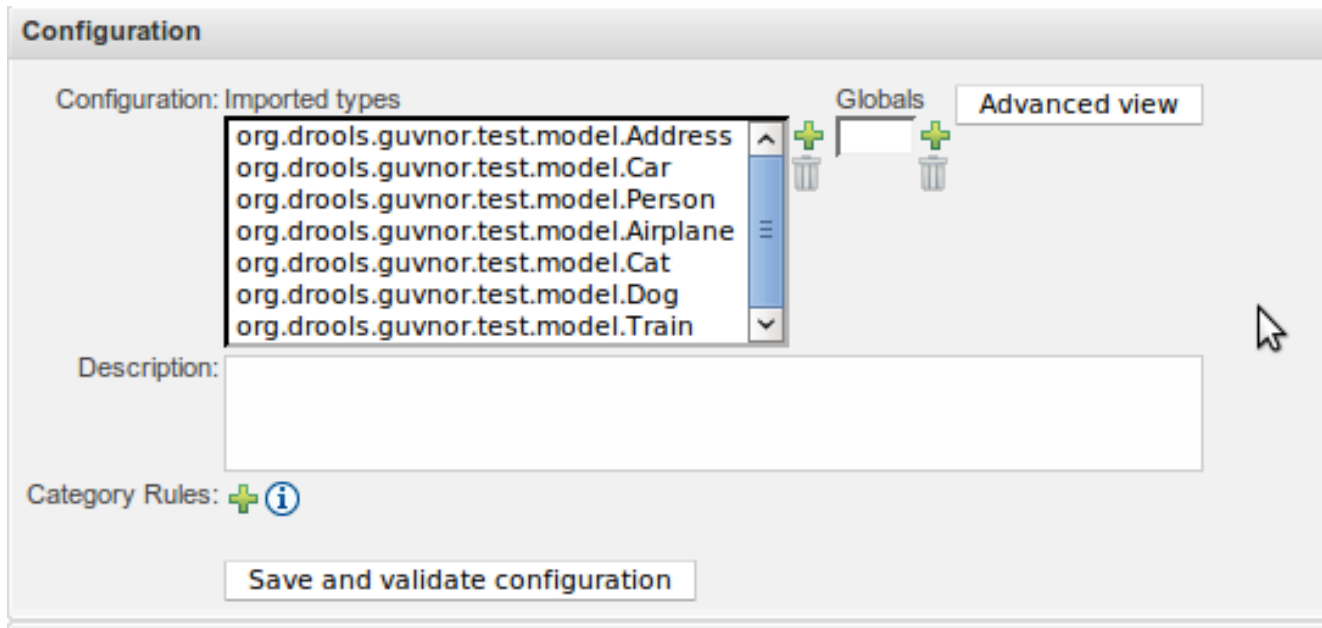


Figure 3.56. Show the imported types

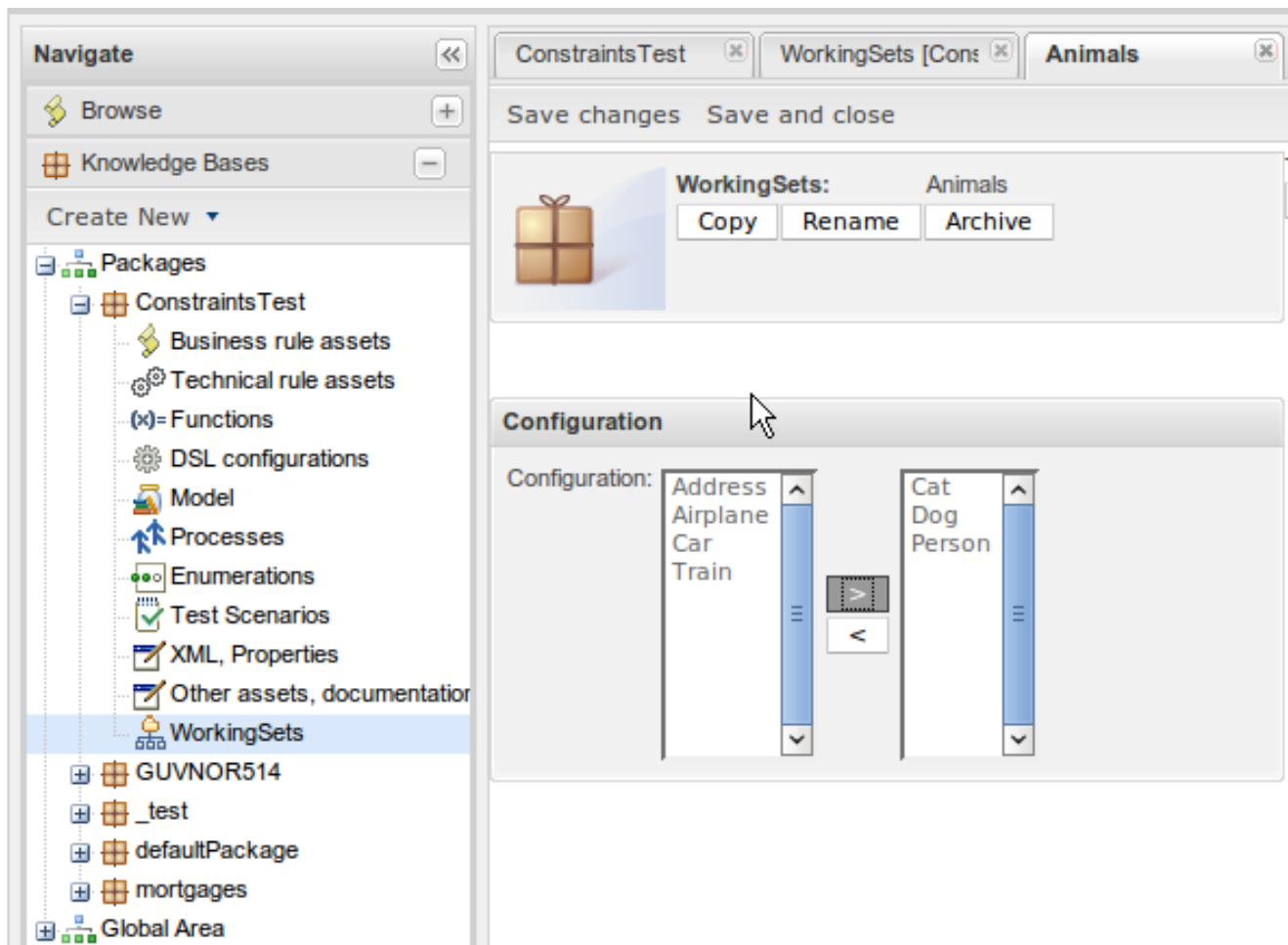


Figure 3.57. Create Animals Working Set

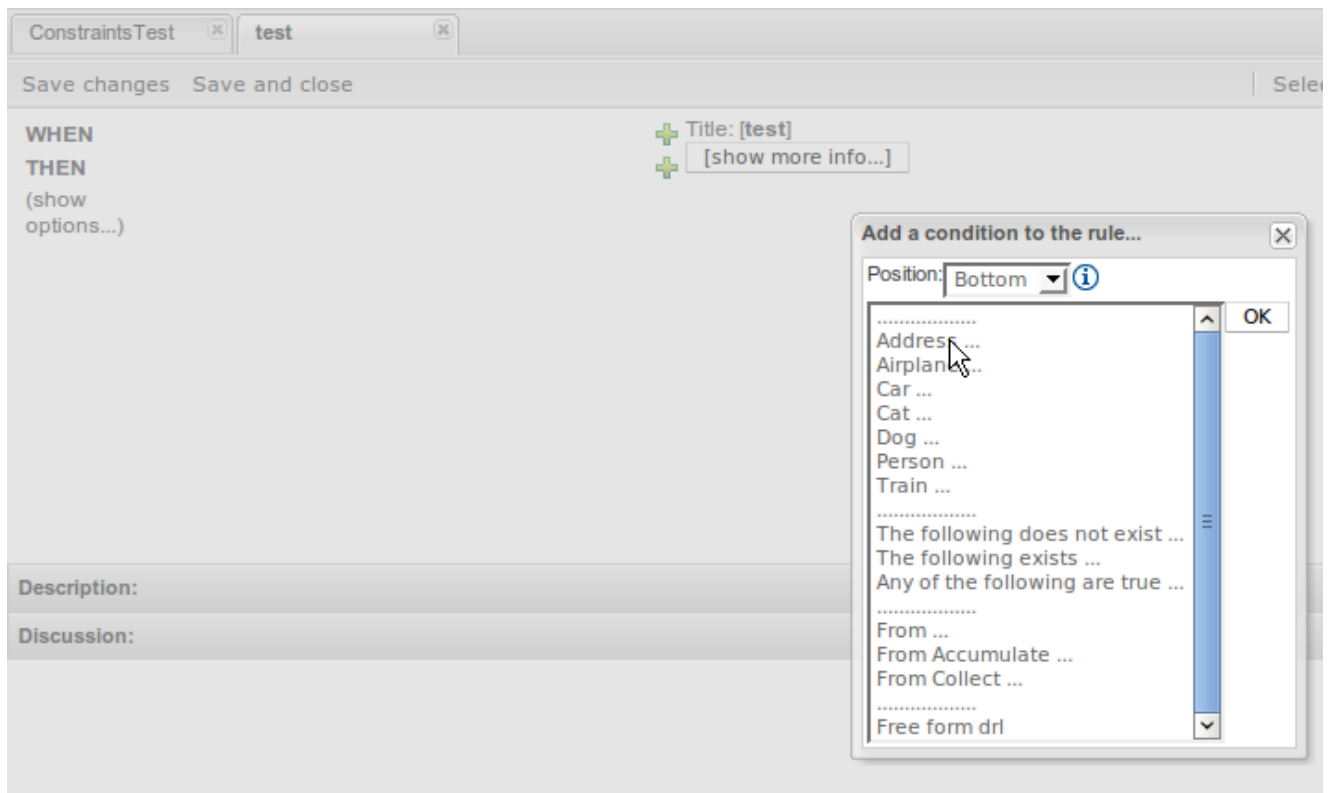


Figure 3.58. Without the Working Set all types are visible

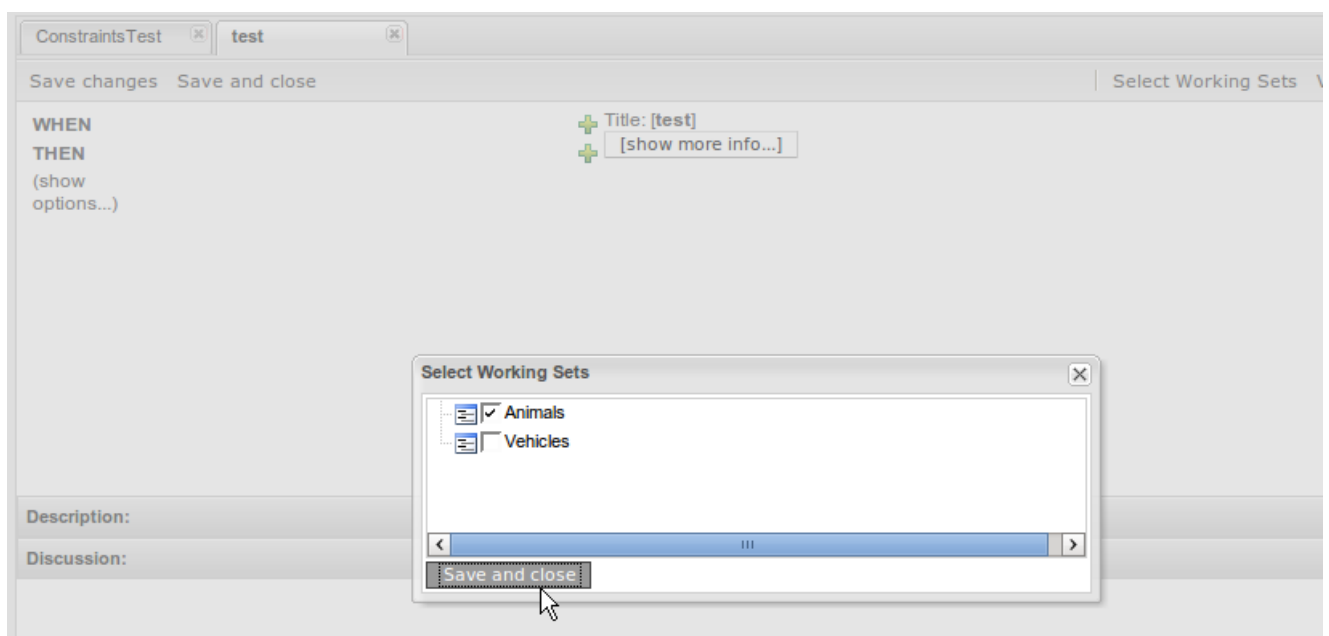


Figure 3.59. Select the Animals Working Set

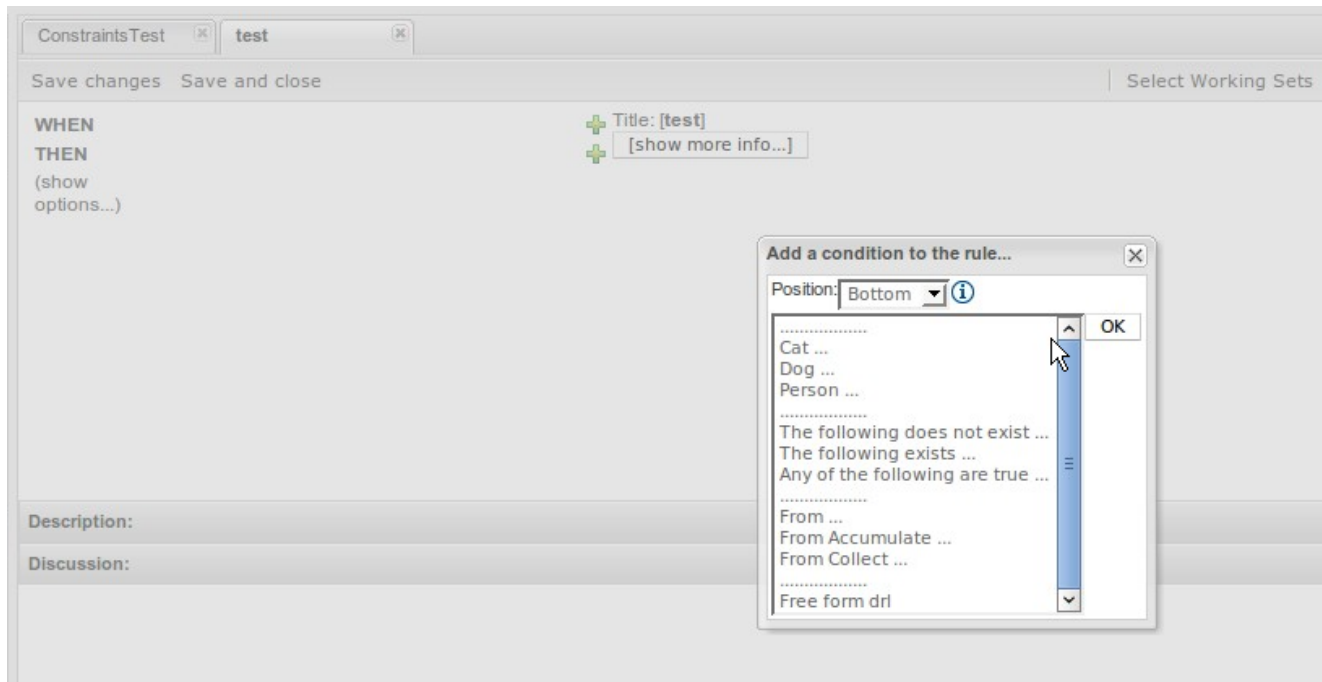


Figure 3.60. The available types are reduced

3.4.5.14. Fact Constraints

Working Sets can be combined with fact constraints to provide additional design time validation. For instance if you are authoring a rule on someone's age, we can know the valid ranges at design time and use this to constrain the author. The fact constraints are part of the workingset and when authoring a rule you must select the work set constraints that you wish to be applied as part of the validation process.

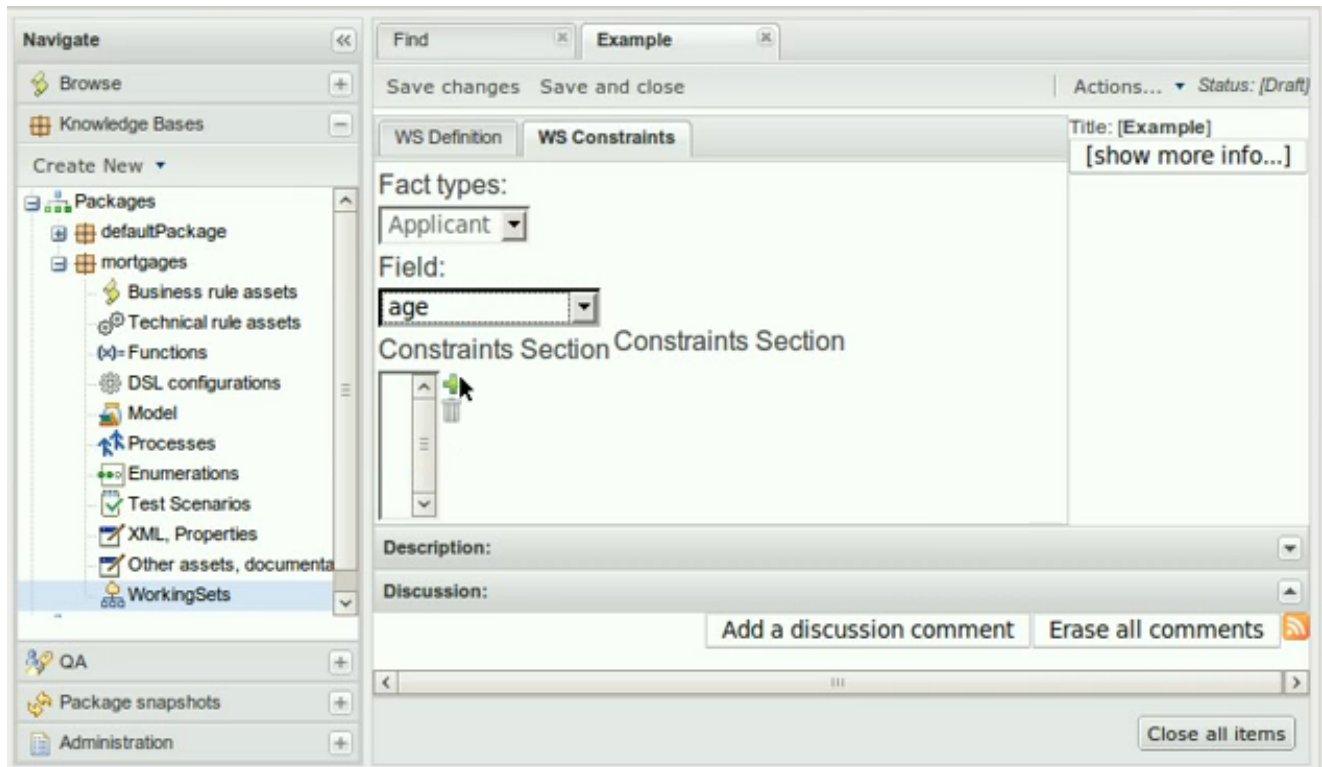


Figure 3.61. Selecting the field

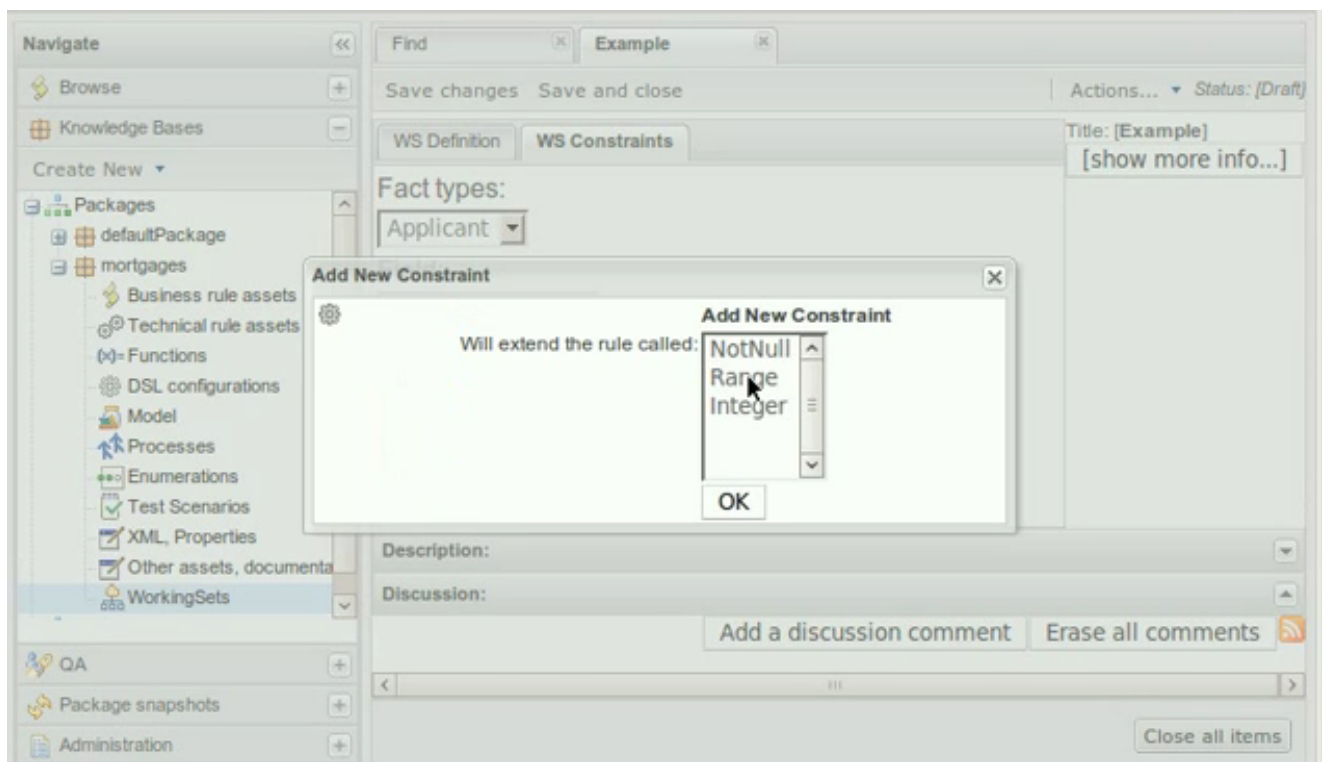


Figure 3.62. Selecting the type of field constraint

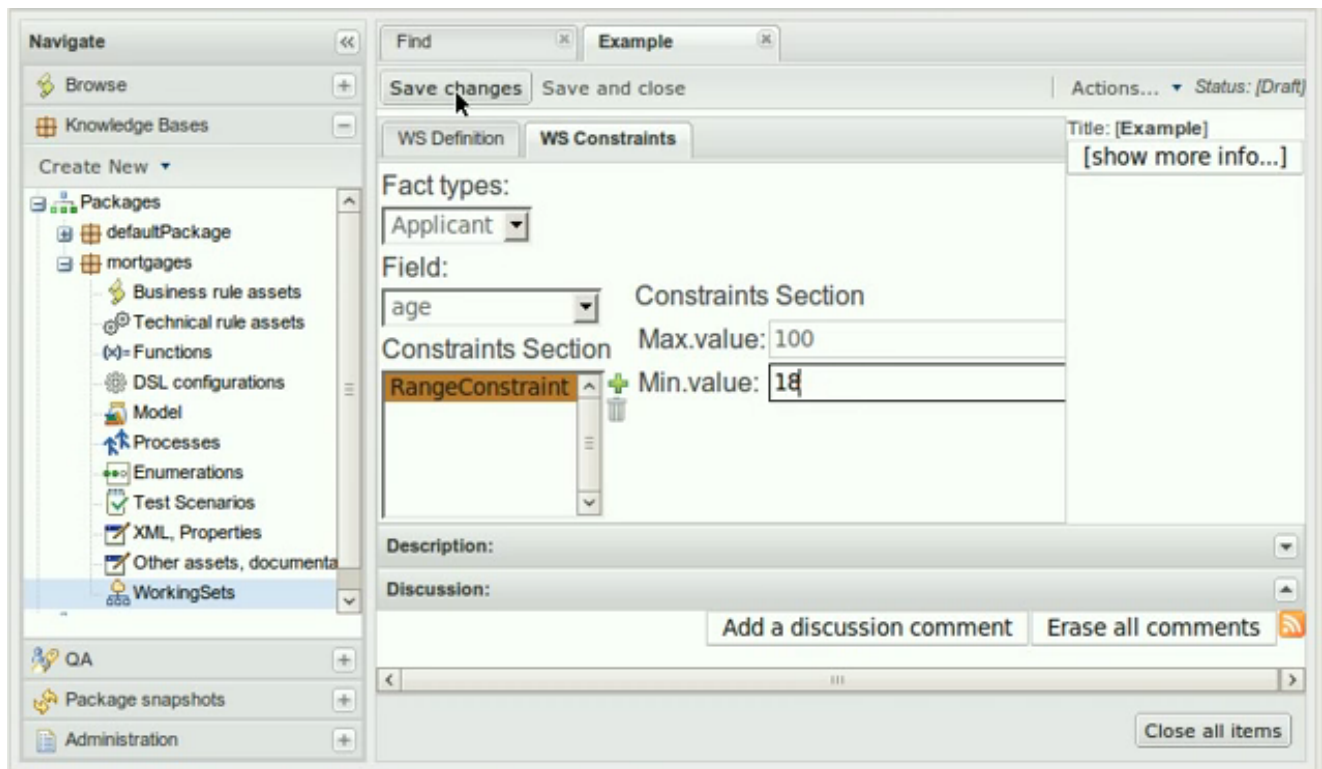


Figure 3.63. A example range constraint

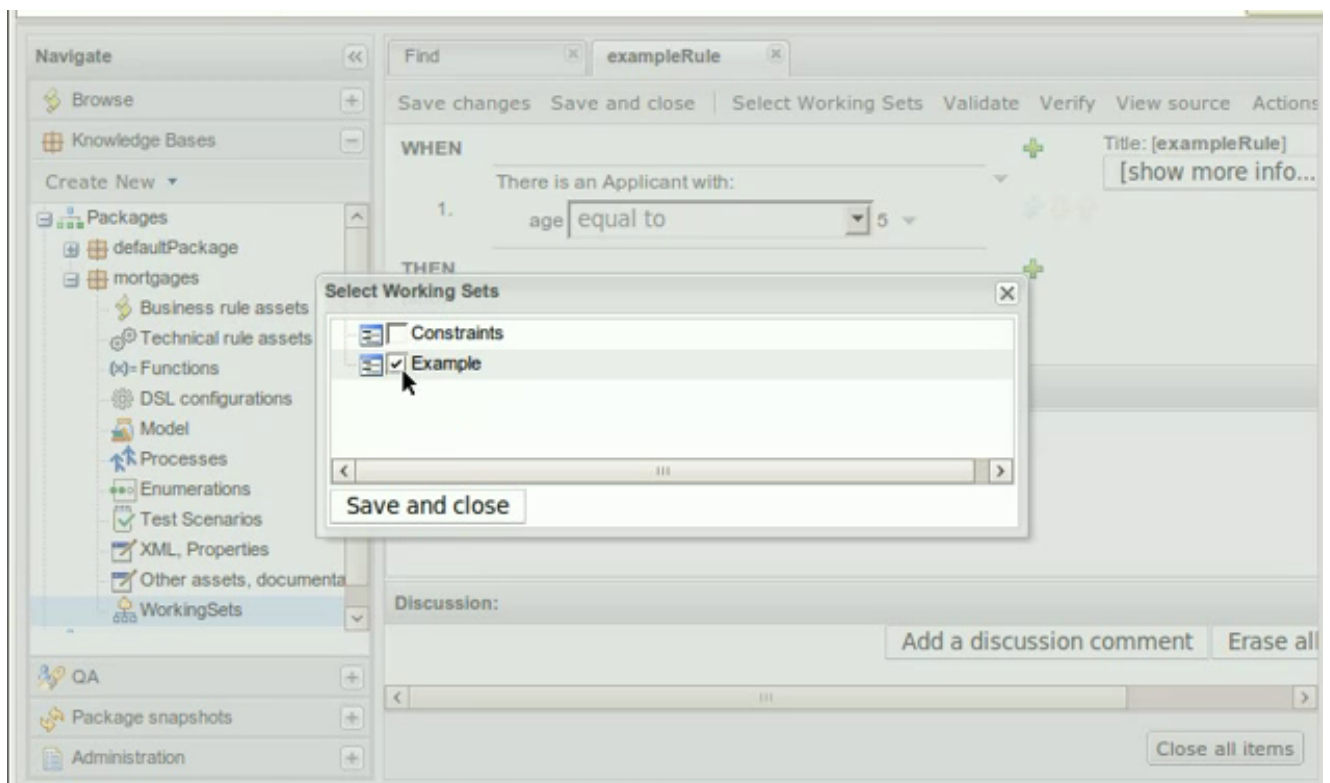


Figure 3.64. Select the working set to validate the age field

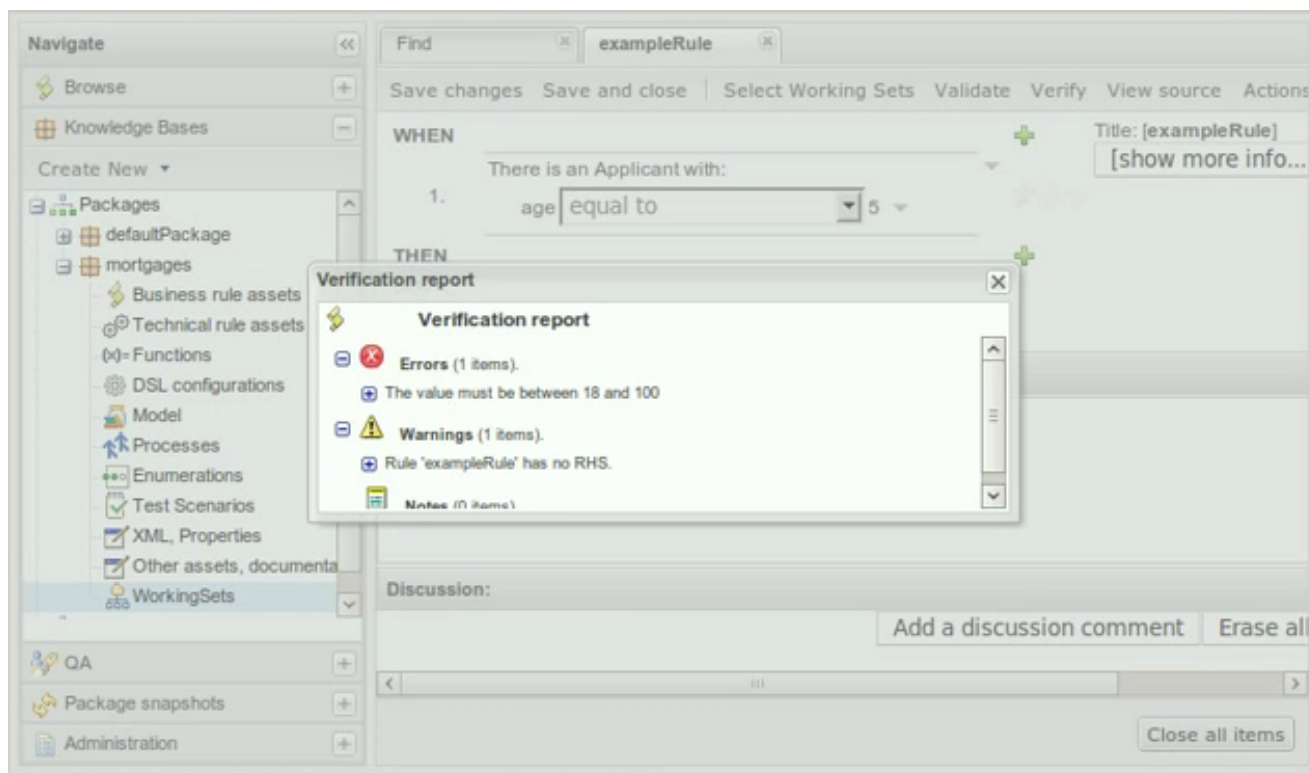


Figure 3.65. A age of 5 is invalid

3.4.5.15. Guided Rule Editor

Editing rules is made more explicit. Editor is less "boxy" and rules are written more as a normal text. "contains" keyword was added and binding variables to restrictions is now easier.

3.4.5.15.1. Pattern Order

Guided Editor supports Pattern reordering in LHS and RHS sections, as well as positional inserting, new Patterns can be inserted in any order (and not always at the end).

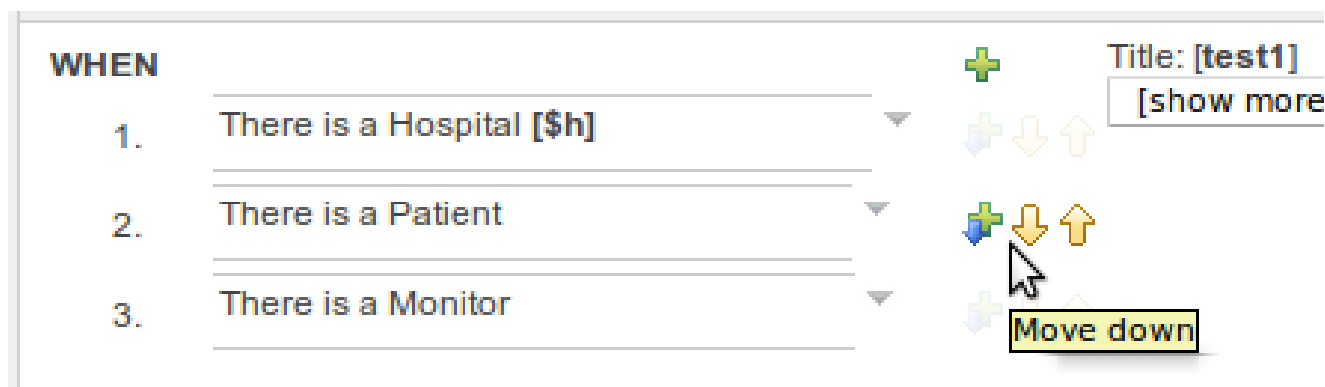


Figure 3.66. Move elements up or down

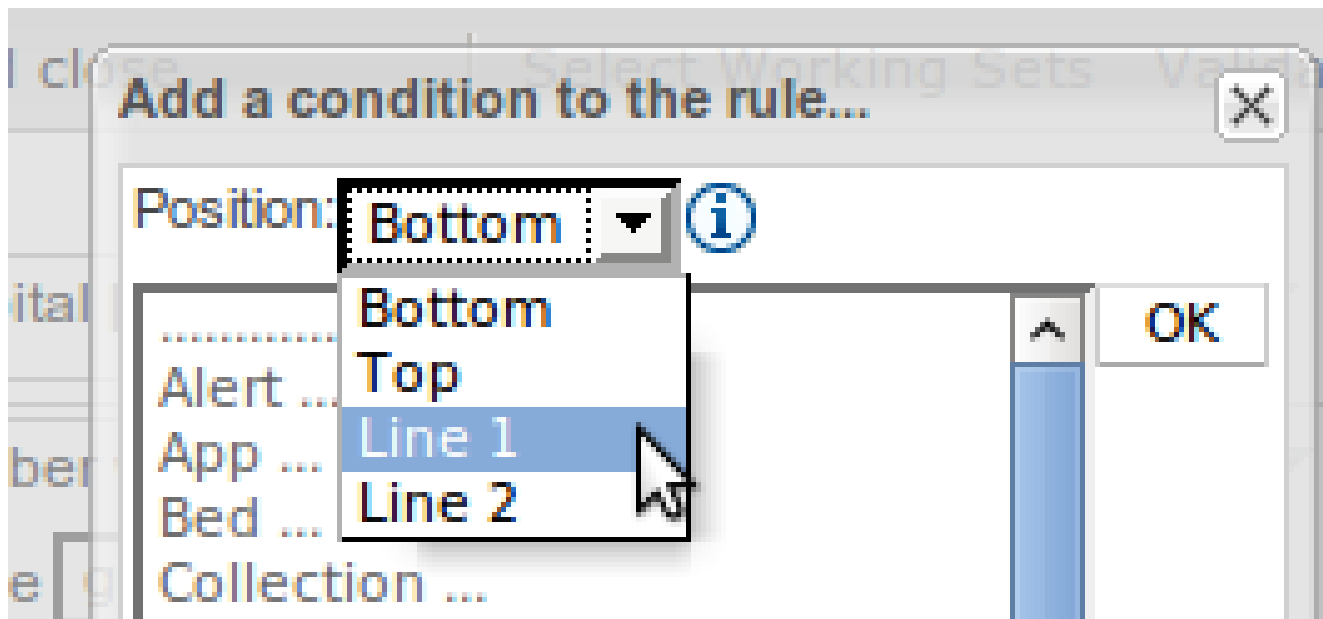


Figure 3.67. Insert Element at any position

3.4.5.16. Decision Table (Guvnor)

Keyword "in" was added.

Columns can be moved and location of a new row can be selected freely.

3.4.6. Eclipse

3.4.6.1. Group Rules in outline view

You can now use sorting and grouping for Agenda Groups.

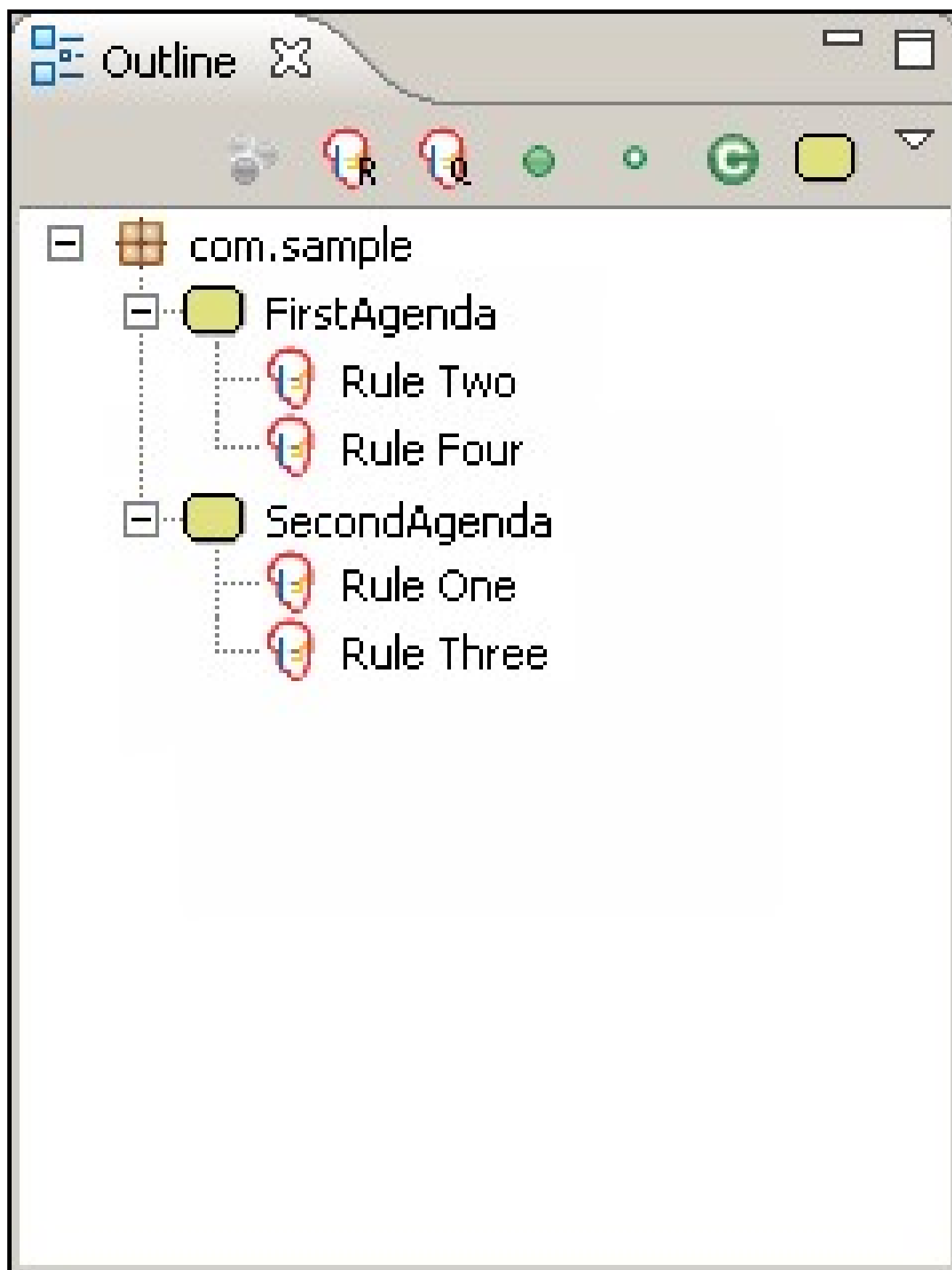


Figure 3.68. Group by Agenda Group

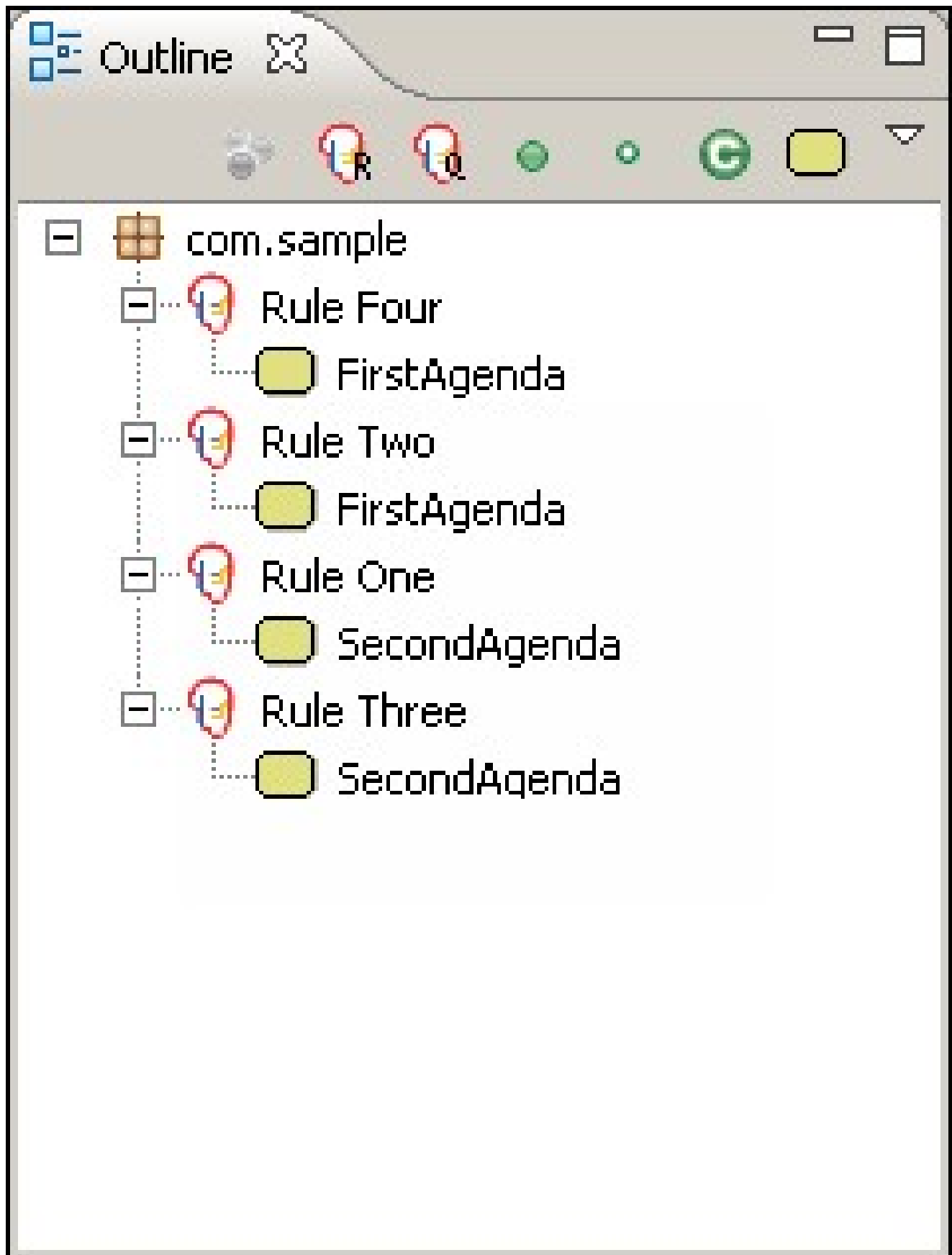


Figure 3.69. Sort by Agenda Group

3.4.6.2. Drag/Drop File support in audit View

It is now possible to drag and drop log files into the audit view.

3.4.7. Known Issues

3.4.7.1. Multithread mode

There is a known issue with the experimental multi-thread execution mode as described in the following JIRA.

<https://jira.jboss.org/browse/JBRULES-2125> [???

3.5. What is New and Noteworthy in Drools 5.0.0

3.5.1. Drools API

Drools now has complete api/implementation separation that is no longer rules oriented. This is an important strategy as we move to support other forms of logic, such as workflow and event processing. The main change is that we are now knowledge oriented, instead of rule oriented. The module drools-api provide the interfaces and factories and we have made pains to provide much better javadocs, with lots of code snippets, than we did before. Drools-api also helps clearly show what is intended as a user api and what is just an engine api, drools-core and drools-compiler did not make this clear enough. The most common interfaces you will use are:

- `org.drools.builder.KnowledgeBuilder`
- `org.drools.KnowledgeBase`
- `org.drools.agent.KnowledgeAgent`
- `org.drools.runtime.StatefulKnowledgeSession`
- `org.drools.runtime.StatelessKnowledgeSession`

Factory classes, with static methods, provide instances of the above interfaces. A pluggable provider approach is used to allow provider implementations to be wired up to the factories at runtime. The Factories you will most commonly used are:

- `org.drools.builder.KnowledgeBuilderFactory`
- `org.drools.io.ResourceFactory`
- `org.drools.KnowledgeBaseFactory`

- `org.drools.agent.KnowledgeAgentFactory`

Example 3.25. A Typical example to load a rule resource

```
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kbuilder.add( ResourceFactory.newUrlResource( url ),
             ResourceType.DRL );
if ( kbuilder.hasErrors() ) {
    System.err.println( builder.getErrors().toString() );
}

KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
kbase.addKnowledgePackages( builder.getKnowledgePackages() );

StatefulKnowledgeSession ksession = knowledgeBase.newStatefulKnowledgeSession();
ksession.insert( new Fibonacci( 10 ) );
ksession.fireAllRules();

ksession.dispose();
```

A Typical example to load a process resource. Notice the `ResourceType` is changed, in accordance with the `Resource` type:

Example 3.26. A Typical example to load a process resource. Notice the `ResourceType` is changed, in accordance with the `Resource` type

```
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kbuilder.add( ResourceFactory.newUrlResource( url ),
             ResourceType.DRF );
if ( kbuilder.hasErrors() ) {
    System.err.println( builder.getErrors().toString() );
}

KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
kbase.addKnowledgePackages( builder.getKnowledgePackages() );

StatefulKnowledgeSession ksession = knowledgeBase.newStatefulKnowledgeSession();
ksession.startProcess( "Buy Order Process" );

ksession.dispose();
```

'kbuilder', 'kbase', 'ksession' are the variable identifiers often used, the k prefix is for 'knowledge'.

Example 3.27. We have uniformed how decision trees are loaded, and they are now consistent with no need to pre generate the DRL with the spreadsheet compiler

```
DecisionTableConfiguration dtconf = KnowledgeBuilderFactory.newDecisionTableConfiguration();
dtconf.setInputType( DecisionTableInputType.XLS );
dtconf.setWorksheetName( "Tables_2" );
kbuilder.add(
    ResourceFactory.newUrlResource( "file://
IntegrationExampleTest.xls" ),
    ResourceType.DTABLE,
    dtconf );
```

It is also possible to configure a `KnowledgeBase` using configuration, via a xml change set, instead of programmatically.

Example 3.28. Here is a simple change set

```
<change-set xmlns='http://drools.org/drools-5.0/change-set'
xmlns:xs='http://www.w3.org/2001/XMLSchema-instance'
xs:schemaLocation='http://drools.org/drools-5.0/change-set change-
set-5.0.xsd' >
  <add>
    <resource source='classpath:org/domain/someRules.drl' type='DRL' />
    <resource source='classpath:org/domain/aFlow.drf' type='DRF' />
  </add>
</change-set>
```

Example 3.29. And it is added just like any other `ResourceType`

```
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kbuilder.add( ResourceFactory.newUrlResource( url ),
    ResourceType.ChangeSet );
```

The other big change for the `KnowledgeAgent`, compared to the `RuleAgent`, is that polling scanner is now a service. further to this there is an abstraction between the agent notification and the resource monitoring, to allow other mechanisms to be used other than polling.

Example 3.30. These services currently are not started by default, to start them do the following

```
ResourceFactory.getResourceChangeNotifierService().start();
ResourceFactory.getResourceChangeScannerService().start();
```

There are two new interfaces added, `ResourceChangeNotifier` and `ResourceChangeMonitor`. KnowledgeAgents subscribe for resource change notifications using the `ResourceChangeNotifier` implementation. The `ResourceChangeNotifier` is informed of resource changes by the added `ResourceChangeMonitors`. We currently only provide one out of the box monitor, `ResourceChangeScannerService`, which polls resources for changes. However the api is there for users to add their own monitors, and thus use a push based monitor such as JMS.

```
ResourceFactory.getResourceChangeNotifierService().addResourceChangeMonitor( myJmsMonitor );
```

3.5.2. Drools Guvnor

- New look web tooling

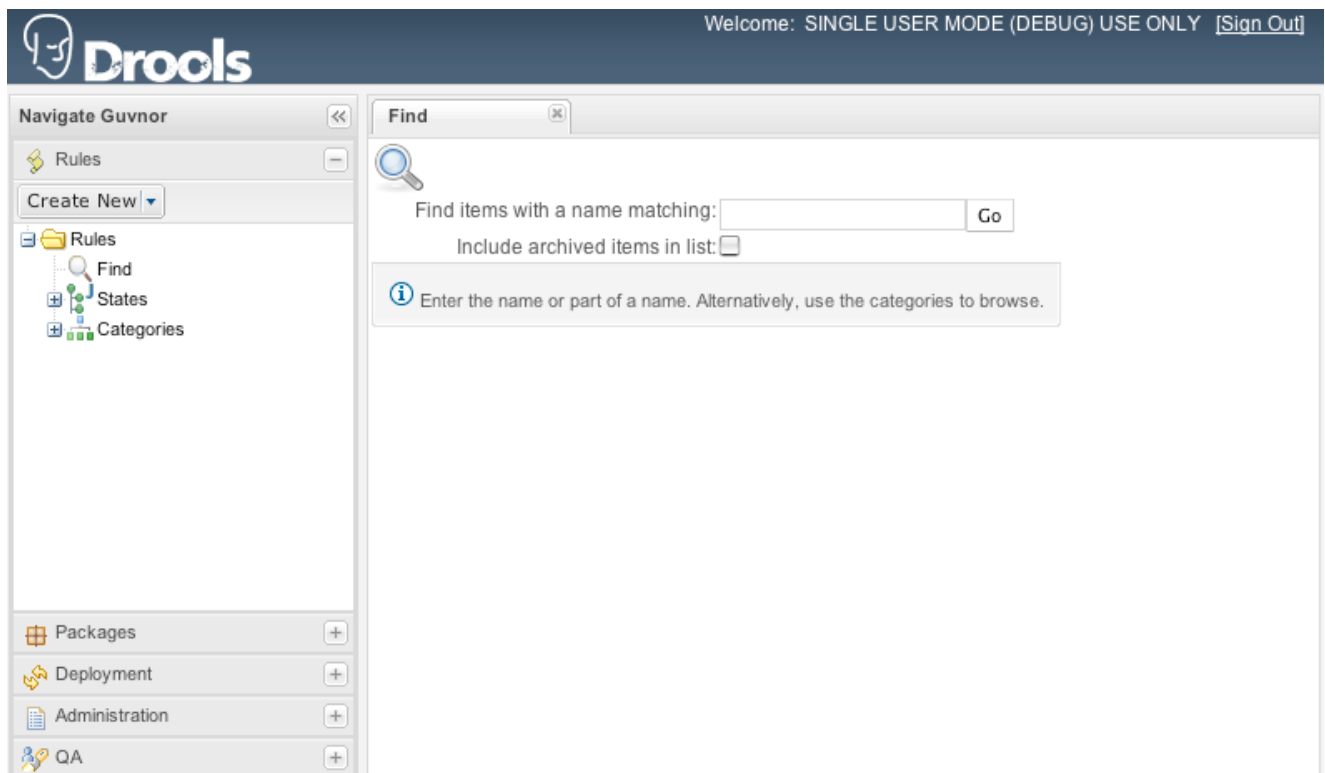


Figure 3.70. New Look

- Web based decision table editor

Decision table							
Modify...							
	Description	Advertiser type	age is at least	Postcode gre...	Postcode les...	Set the value...	Set the rea...
Advertiser type: Agency (3 Items)							
1	Good suburbs	Agency	10	4000	4100	→ 42	Loyal
2	Good suburbs	Agency		2000	2100	→ 43	Good region
4	Good suburbs	Agency		2200	2300	→ 43	Good region
Advertiser type: Partner (1 Item)							
3	Partners	Partner	1			→ 49	Other

Figure 3.71. Web based decision table editor

- Integrated scenario testing

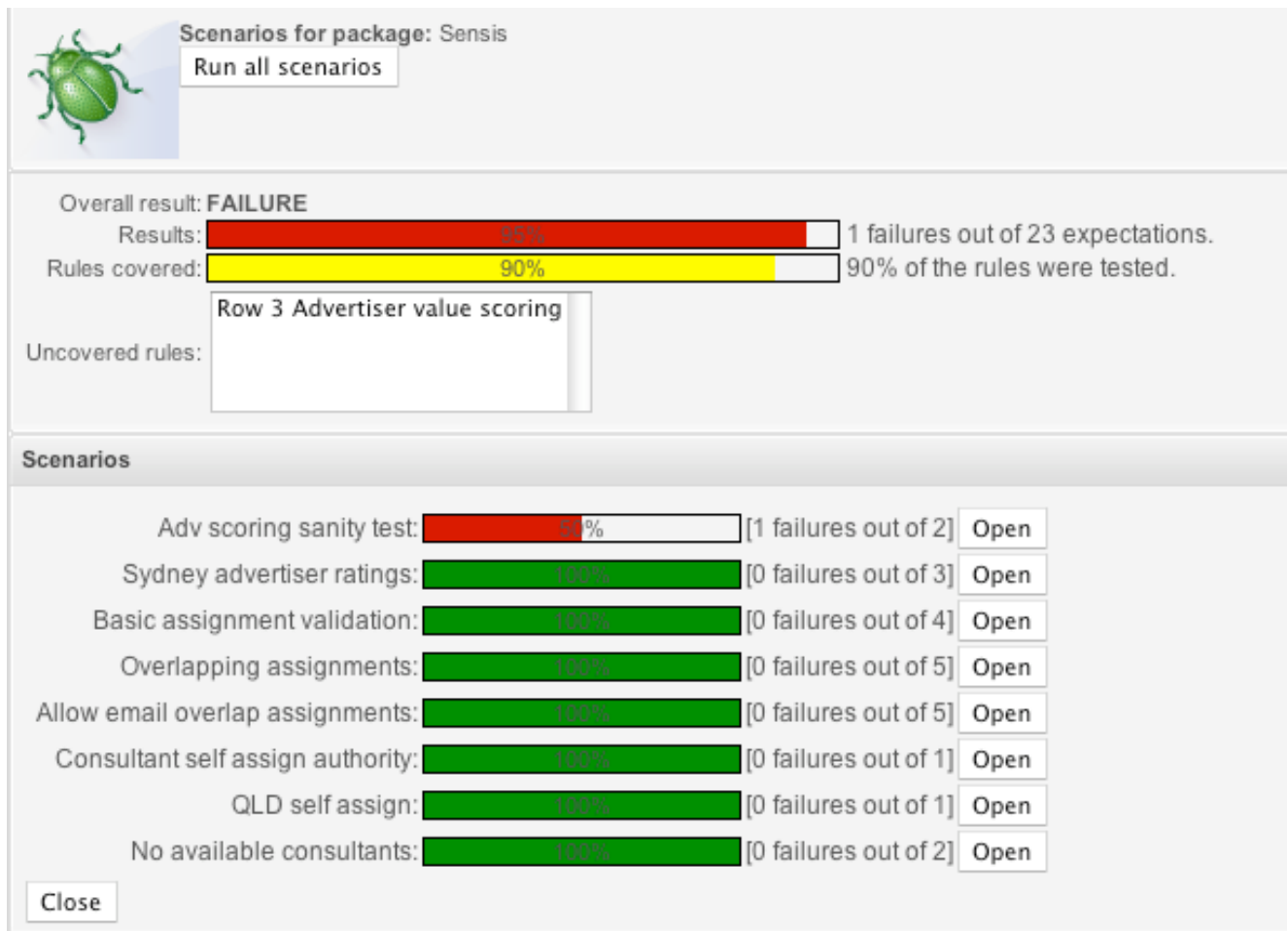


Figure 3.72. Running all scenarios

Save changes Copy Archive Change state Status: [C]

Run scenario

Results: 50%

Summary: [adv] field [valueScore] was [42].
 [adv] field [valueReason] was [Loyal] expected [Other].

+ GIVEN

insert [Advertiser]

[adv]

advertiserType: Agency

ageOfCustomer: 12

postcode: 4064

+ EXPECT

2 rules fired in 0ms. Show rules fired

Use real date and time

Advertiser [adv] has values.

valueScore: equals 42

valueReason: equals Other (Actual: Loyal)

More...

(configuration) All rules may fire

+ (globals)

Adv scoring sanity test

Categories: +

Modified on: Mar 28, 2008 4:47:25
 by: alan_parsons
 Note:
 Version: 7
 Created Mar 28, 2008 12:08:2
 on: PM
 Created by: alan_parsons
 Format: scenario

Package: Sensis

Subject:

Type:

External link:

Source:

Version history

Figure 3.73. Running single scenario

- WebDAV file based interface to repository

Folders	Name	Internet Address
Desktop	com.billasurf.finance	http://172.16.190.2:8888/org....
My Documents	com.billasurf.hrman	http://172.16.190.2:8888/org....
My Computer	com.billasurf.manufacturing	http://172.16.190.2:8888/org....
3 1/2 Floppy (A:)	com.billasurf.manufacturing.plant	http://172.16.190.2:8888/org....
Local Disk (C:)	com.billasurf.sales	http://172.16.190.2:8888/org....
DVD-RW Drive (D:)	defaultPackage	http://172.16.190.2:8888/org....
Shared Folders on '.host' (Z:)		
Control Panel		
Shared Documents		
Administrator's Documents		
Web Folders		
Guvnor		
packages		
com.billasurf.finance		
com.billasurf.hrman		
com.billasurf.manufacturing		
com.billasurf.manufacturing.plant		
com.billasurf.sales		
defaultPackage		

Figure 3.74. WebDAV

- Declarative modelling of types (types that are not in pojors)

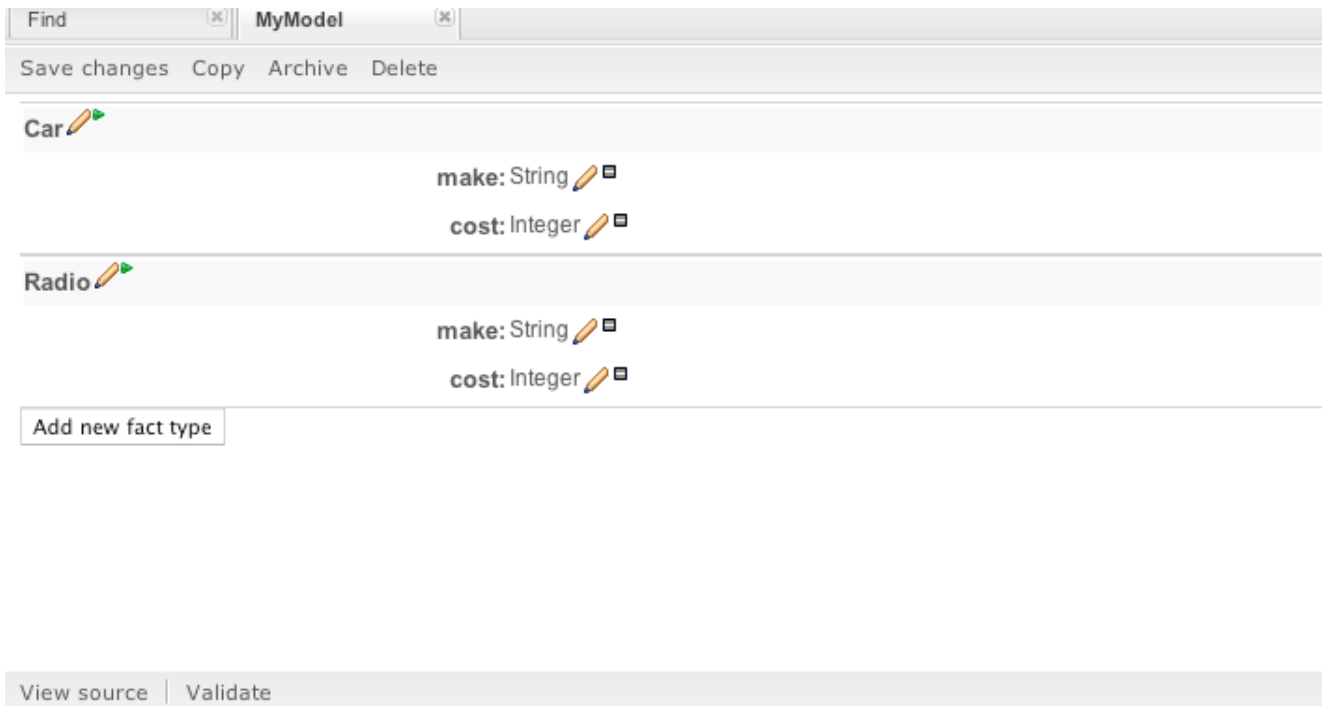


Figure 3.75. >Declarative modelling

This works with the new "declare" statement - you can now declare types in drl itself. You can then populate these without using a pojo (if you like). These types are then available in the rulebase.

- Fine grained security (lock down access to the app per package or per category). Users who only have category permissions have limited UI capability (ideal for business users)
- Execution server - access rules via XML or JSON for execution
- Category rules allows you to set 'parent rules' for a category. Any rules appearing in the given category will 'extend' the rule specified - ie inherit the conditions/LHS. The base rule for the category can be set on package configuration tab. RHS is not inherited, only the LHS
- Scenario runner detects infinite loops
- Scenario runner can show event trace that was recorded by audit logger
- DSL sentences in guided editor can now be set to show enums as a dropdown, dates as a date picker, booleans as a checkbox and use regular expressions to validate the inputs (DSL Widgets in Guvnor)
- Functions can be edited with text editor
- It is possible to add objects to global collections.
- Translations to English, Spanish, Chinese and Japanese

3.5.3. Drools Expert

3.5.3.1. Asymmetrical Rete algorithm implementation

Shadow proxies are no longer needed. Shadow proxies protected the engine from information change on facts, which if occurred outside of the engine's control it could not be modified or retracted.

3.5.3.2. `PackageBuilder` can now build multiple namespaces

You no longer need to confine one `PackageBuilder` to one package namespace. Just keeping adding your DRLs for any namespace and `getPackages()` returns an array of `Packages` for each of the used namespaces.

Example 3.31. Getting multiple packages

```
Package[] packages = pkgBuilder.getPackages();
```

3.5.3.3. `RuleBase` attachment to `PackageBuilder`

It is now possible to attach a `RuleBase` to a `PackageBuilder`, this means that rules are built and added to the rulebase at the same time. `PackageBuilder` uses the `Package` instances of the actual `RuleBase` as it's source, removing the need for additional `Package` creation and merging that happens in the existing approach.

Example 3.32. Attaching `RuleBase` to `PackageBuilder`

```
RuleBase ruleBase = RuleBaseFactory.newRuleBase();  
PackageBuilder pkgBuilder = new PackageBuilder( ruleBase, null );
```

3.5.3.4. Binary marshalling of stateful sessions

Stateful sessions can now saved and resumed at a later date. Pre-loaded data sessions can now be created. Pluggable strategies can be used for user object persistence, i.e. hibernate or identity maps.

3.5.3.5. Type Declaration

Drools now supports a new base construct called Type Declaration. This construct fulfils two purposes: the ability to declare fact metadata, and the ability to dynamically generate new fact types local to the rule engine. The Guvnor modelling tool uses this underneath. One example of the construct is:

Example 3.33. Declaring `StockTick`

```
declare StockTick
  @role( event )
  @timestamp( timestampAttr )

  companySymbol : String
  stockPrice : double
  timestampAttr : long
end
```

3.5.3.6. Declaring Fact Metadata

To declare and associate fact metadata, just use the @ symbol for each metadata ID you want to declare. Example:

Example 3.34. Declaring metadata

```
declare StockTick
  @role( event )
end
```

3.5.3.7. Triggering Bean Generation

To activate the dynamic bean generation, just add fields and types to your type declaration:

Example 3.35. Declaring `Person`

```
declare Person
  name : String
  age : int
end
```

3.5.3.8. DSL improvements

A series of DSL improvements were implemented, including a completely new parser and the ability to declare matching masks for matching variables. For instance, one can constrain a phone number field to a 2-digit country code + 3-digit area code + 8-digit phone number, all connected by a "-" (dash), by declaring the DSL map like: The phone number is {number:\d{2}-\d{3}-\d{8}} Any valid java regexp may be used in the variable mask.

3.5.3.9. `fireUntilHalt()`

Drools now supports "fireUntilHalt()" feature, that starts the engine in a reactive mode, where rules will be continually fired, until a halt() call is made. This is specially useful for CEP scenarios that require what is commonly known as "active queries".

3.5.3.10. Rule Base partitioning and multi-thread propagation

Drools ReteOO algorithm now supports an option to start the rule base in a multi-thread mode, where Drools ReteOO network is split into multiple partitions and rules are then evaluated concurrently by multiple threads. This is also a requirement for CEP where there usually are several independent rules running concurrently, with near realtime performance/throughput requirements and the evaluation of one can not interfere with the evaluation of others.

3.5.3.11. XSD Model Support

Drools now supports XSD models. Remember though the XSD model is generated as pojos local to the Drools classloader. A helper class is there to assist in the creation of the model in the packagebuilder. Once the data model is generated you'll typically use the JAXB dataloader to insert data.

3.5.3.12. Data Loaders

Drools now supports two data loaders, Smooks and JAXB. Smooks is an open source data transformation tool for ETL and JAXB a standard sun data mapping tool. Unit tests showing Smooks can be found [here](#) and JAXB [here](#).

3.5.3.13. Type safe configuration

In addition to the ability of configuring options in drools through configuration files, system properties and by setting properties through the API `setProperty()` method, Drools-API now supports type safe configuration. We didn't want to add specific methods for each possible configuration methods for two reasons: it polutes the API and every time a new option is added to Drools, the API would have to change. This way, we followed a modular, class based configuration, where a new Option class is added to the API for each possible configuration, keeping the API stable, but flexible at the same time. So, in order to set configuration options now, you just need to use the enumerations or factories provided for each option. For instance, if you want to configure the knowledge base for assert behavior "equality" and to automatically remove identities from pattern matchings, you would just use the enums:

Example 3.36. Configuring

```
KnowledgeBaseConfiguration config = KnowledgeBaseFactory.newKnowledgeBaseConfiguration();
config.setOption( AssertBehaviorOption.EQUALITY );
config.setOption( RemoveIdentitiesOption.YES );
```


For options that don't have a predefined constant or can assume multiple values, a factory method is provided. For instance, to configure the alpha threshold to 5, just use the "get" factory method:

Example 3.37. Configuring alpha threshold

```
config.setOption( AlphaThresholdOption.get(5) );
```

As you can see, the same `setOption()` method is used for the different possible configurations, but they are still type safe.

3.5.3.14. New accumulate functions: `collectSet` and `collectList`

There are times when it is necessary to collect sets or lists of values that are derived from the facts attributes, but are not facts themselves. In such cases, it was not possible to use the `collect` CE. So, Drools now has two accumulate functions for such cases: `collectSet` for collecting sets of values (i.e., with no duplicate values) and `collectList` for collecting lists of values (i.e., allowing duplicate values):

Example 3.38. New accumulate functions

```
# collect the set of unique names in the working memory
$names : Set() from accumulate( Person( $n : name, $s : surname ),
                               collectSet( $n + " " + $s ) )

# collect the list of alarm codes from the alarms in the working memory
$codes : List() from accumulate( Alarm( $c : code, $s : severity ),
                                collectList( $c + $s ) )
```

3.5.3.15. New metadata for type declarations:

`@propertyChangeSupport`

Facts that implement support for property changes as defined in the Javabeans(tm) spec, now can be annotated so that the engine register itself to listen for changes on fact properties. The boolean parameter that was used in the `insert()` method in the Drools 4 API is deprecated and does not exist in the `drools-api` module.

Example 3.39. `@propertyChangeSupport`

```
declare Person
    @propertyChangeSupport
```

```
end
```

3.5.3.16. Batch Executor

Batch Executor allows for the scripting of a Knowledge session using Commands, which can also re, both the `StatelessKnowledgeSession` and `StatefulKnowledgeSession` implement this interface. Commands are created using the `CommandFactory` and executed using the "execute" method, such as the following insert Command:

Example 3.40. Using `CommandFactory`

```
ksession.execute( CommandFactory.newInsert( person ) );
```

Typically though you will want to execute a batch of commands, this can be achieved via the composite Command `BatchExecution`. `BatchExecutionResults` is now used to handle the results, some commands can specify "out" identifiers which it used to add the result to the `BatchExecutionResult`. Handily queries can now be executed and results added to the `BatchExecutionResult`. Further to this results are scoped to this execute call and return via the `BatchExecutionResults`:

Example 3.41. Using `BatchExecutionResult`

```
List<Command> cmds = new ArrayList<Command>();
cmds.add( CommandFactory.newSetGlobal( "list1", new ArrayList(), true ) );
cmds.add( CommandFactory.newInsert( new Person( "jon", 102 ), "person" ) );
cmds.add( CommandFactory.newQuery( "Get People", "getPeople" ) );

BatchExecutionResults results = ksession.execute( CommandFactory.newBatchExecution( cmds ) );
results.getValue( "list1" ); // returns the ArrayList
results.getValue( "person" ); // returns the inserted fact Person
results.getValue( "Get People" ); // returns the query as a QueryResults instance.
end
```

The `CommandFactory` details the supported commands, all of which can be marshalled using `XStream` and the `BatchExecutionHelper`. This can be combined with the pipeline to automate the scripting of a session.

Example 3.42. Using `PipelineFactory`

```
Action executeResultHandler = PipelineFactory.newExecuteResultHandler();
```

```

Action assignResult = PipelineFactory.newAssignObjectAsResult();
assignResult.setReceiver( executeResultHandler );
Transformer outTransformer = PipelineFactory.newXStreamToXmlTransformer( BatchExecutionHelper.m
outTransformer.setReceiver( assignResult );
KnowledgeRuntimeCommand batchExecution = PipelineFactory.newBatchExecutor();
batchExecution.setReceiver( outTransformer );
Transformer inTransformer = PipelineFactory.newXStreamFromXmlTransformer( BatchExecutionHelper.m
inTransformer.setReceiver( batchExecution );
Pipeline pipeline = PipelineFactory.newStatelessKnowledgeSessionPipeline( ksession );
pipeline.setReceiver( inTransformer );

```

Using the above for a rulset that updates the price of a Cheese fact, given the following xml to insert a Cheese instance using an out-identifier:

Example 3.43. Updating Cheese fact

```

<batch-execution>
<insert out-identifier='outStilton'>
  <org.drools.Cheese>
    <type>stilton</type>
    <price>25</price>
    <oldPrice>0</oldPrice>
  </org.drools.Cheese>
</insert>
</batch-execution>

```

We then get the following BatchExecutionResults:

Example 3.44. Updating Cheese fact

```

<batch-execution-results>
<result identifier='outStilton'>
  <org.drools.Cheese>
    <type>stilton</type>
    <oldPrice>0</oldPrice>
    <price>30</price>
  </org.drools.Cheese>
</result>
</batch-execution-results>

```

3.5.3.17. Marshalling

The MarshallerFactory is used to marshal and unmarshal StatefulKnowledgeSessions. At the simplest it can be used as follows:

Example 3.45. Using `MarshallerFactory`

```
// ksession is the StatefulKnowledgeSession
// kbase is the KnowledgeBase
ByteArrayOutputStream baos = new ByteArrayOutputStream();
Marshaller marshaller = MarshallerFactory.newMarshaller( kbase );
marshaller.marshall( baos, ksession );
baos.close();
```

However with marshalling you need more flexibility when dealing with referenced user data. To achieve this we have the `ObjectMarshallingStrategy` interface. Two implementations are provided, but the user can implement their own. The two supplied are `IdentityMarshallingStrategy` and `SerializeMarshallingStrategy`. `SerializeMarshallingStrategy` is the default, as used in the example above and it just calls the `Serializable` or `Externalizable` methods on a user instance. `IdentityMarshallingStrategy` instead creates an int id for each user object and stores them in a `Map` the id is written to the stream. When unmarshalling it simply looks to the `IdentityMarshallingStrategy` map to retrieve the instance. This means that if you use the `IdentityMarshallingStrategy` it's stateful for the life of the `Marshaller` instance and will create ids and keep references to all objects that it attempts to marshal.

Example 3.46. Code to use a `IdentityMarshallingStrategy`

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
Marshaller marshaller = MarshallerFactory.newMarshaller( kbase, new ObjectMarshallingStrategy[ ] );
marshaller.marshall( baos, ksession );
baos.close();
```

For added flexibility we can't assume that a single strategy is suitable for this we have added the `ObjectMarshallingStrategyAcceptor` interface that each `ObjectMarshallingStrategy` has. The `Marshaller` has a chain of strategies and when it attempts to read or write a user object it iterates the strategies asking if they accept responsibility for marshalling the user object. One one implementation is provided the `ClassFilterAcceptor`. This allows strings and wild cards to be used to match class names. The default is `"*.*"`, so in the above the `IdentityMarshallingStrategy` is used which has a default `"*.*"` acceptor. But lets say we want to serialise all classes except for one given package, where we will use identity lookup, we could do the following:

Example 3.47. Using identity lookup

```

ByteArrayOutputStream baos = new ByteArrayOutputStream();
ObjectMarshallingStrategyAcceptor identityAcceptor = MarshallerFactory.newClassFilterAcceptor(
ObjectMarshallingStrategy identityStrategy = MarshallerFactory.newIdentityMarshallingStrategy(
Marshaller marshaller = MarshallerFactory.newMarshaller( kbase, new ObjectMarshallingStrategy(
marshaller.marshall( baos, ksession );
baos.close();

```

3.5.3.18. Knowledge Agent

The `KnowledgeAgent` is created by the `KnowledgeAgentFactory`. The `KnowledgeAgent` provides automatic loading, caching and re-loading, of resources and is configured from a properties files. The `KnowledgeAgent` can update or rebuild this `KnowledgeBase` as the resources it uses are changed. The strategy for this is determined by the configuration given to the factory, but it is typically pull based using regular polling. We hope to add push based updates and rebuilds in future versions. The Following example constructs an agent that will build a new `KnowledgeBase` from the files specified in the path String. It will poll those files every 30 seconds to see if they are updated. If new files are found it will construct a new `KnowledgeBase`, instead of updating the existing one, due to the "newInstance" set to "true" (however currently only the value of "true" is supported and is hard coded into the engine):

Example 3.48. Constructing an agent

```

// Set the interval on the ResourceChangeScannerService if you are to use it
// and default of 60s is not desirable.
ResourceChangeScannerConfiguration sconf = ResourceFactory.getResourceChangeScannerService().newConfiguration();
sconf.setProperty( "drools.resource.scanner.interval",
    "30" ); // set the disk scanning interval to 30s, default is 60s
ResourceFactory.getResourceChangeScannerService().configure( sconf );
KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
KnowledgeAgentConfiguration aconf = KnowledgeAgentFactory.newKnowledgeAgentConfiguration();
aconf.setProperty( "drools.agent.scanDirectories",
    "true" ); // we want to scan directories, not just files,
    turning this on turns on file scanning
aconf.setProperty( "drools.agent.newInstance",
    "true" ); // resource changes results in a new instance
    of the KnowledgeBase being built,
    // this cannot currently be set to false
    for incremental building

KnowledgeAgent kagent = KnowledgeAgentFactory.newKnowledgeAgent( "test
    agent", // the name of the agent

```

```
        kbase, // the KnowledgeBase
        to use, the Agent will also monitor any exist knowledge definitions
        aconf );
kagent.applyChangeSet( ResourceFactory.newUrlResource( url ) ); // resource to
the change-set xml for the resources to add
```

KnowledgeAgents can take a empty KnowledgeBase or a populated one. If a populated KnowledgeBase is provided, the KnowledgeAgent will iterate KnowledgeBase and subscribe to the Resource that it finds. While it is possible for the KnowledgeBuilder to build all resources found in a directory, that information is lost by the KnowledgeBuilder so those directories will not be continuously scanned. Only directories specified as part of the `applyChangeSet(Resource)` method are monitored.

3.5.4. Drools Flow

Drools 4.0 had simple "RuleFlow" which was for orchestrating rules. Drools 5.0 introduces a powerful (extensible) workflow engine. It allows users to specify their business logic using both rules and processes (where powerful interaction between processes and rules is possible) and offers a unified enviroment.

3.5.4.1. Process Instance view at a specific breakpoint

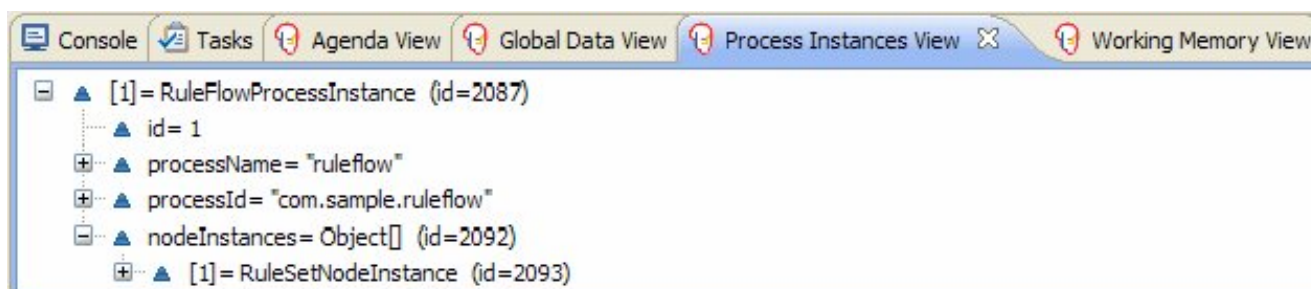


Figure 3.76. Rule Flow properties

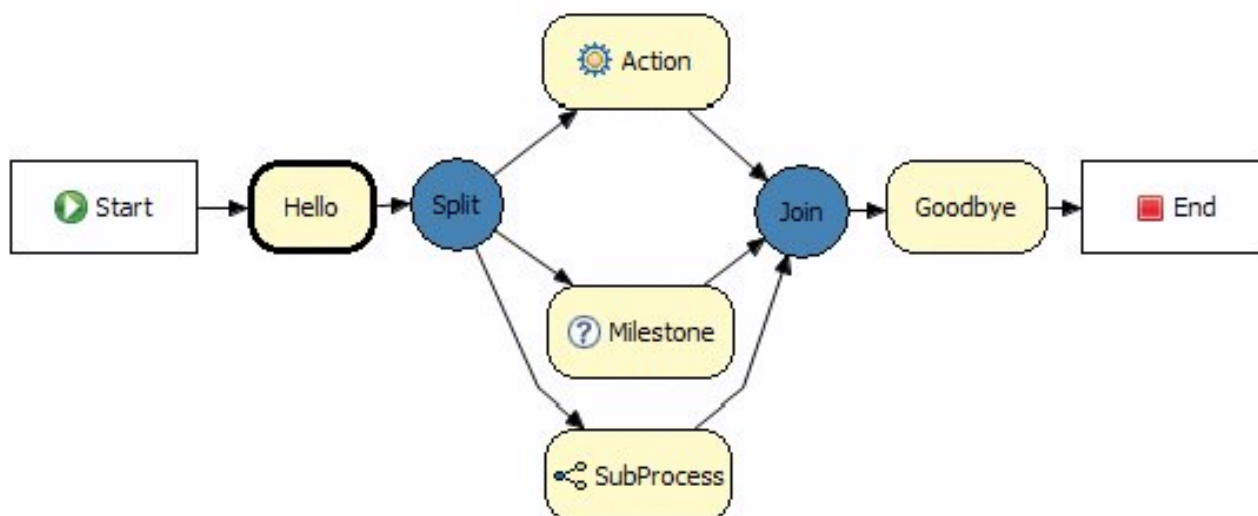


Figure 3.77. Current active nodes in a workflow in a specific breakpoint

3.5.4.2. New Nodes

Timers:

A timer node can be added which causes the execution of the node to wait for a specific period. Currently just uses JDK defaults of initial delay and repeat delay, more complex timers will be available in further milestones.

Human Task:

Processes can include tasks that need to be executed by human actors. Human tasks include parameters like taskname, priority, description, actorId, etc. The process engine can easily be integrated with existing human task component (like for example a WS-HumanTask implementation) using our pluggable work items (see below). Swimlanes and assignment rules are also supported.

The palette in the screenshot shows the two new components, and the workflow itself shows the human task in use. It also shows two "work items" which is explained in the next section:

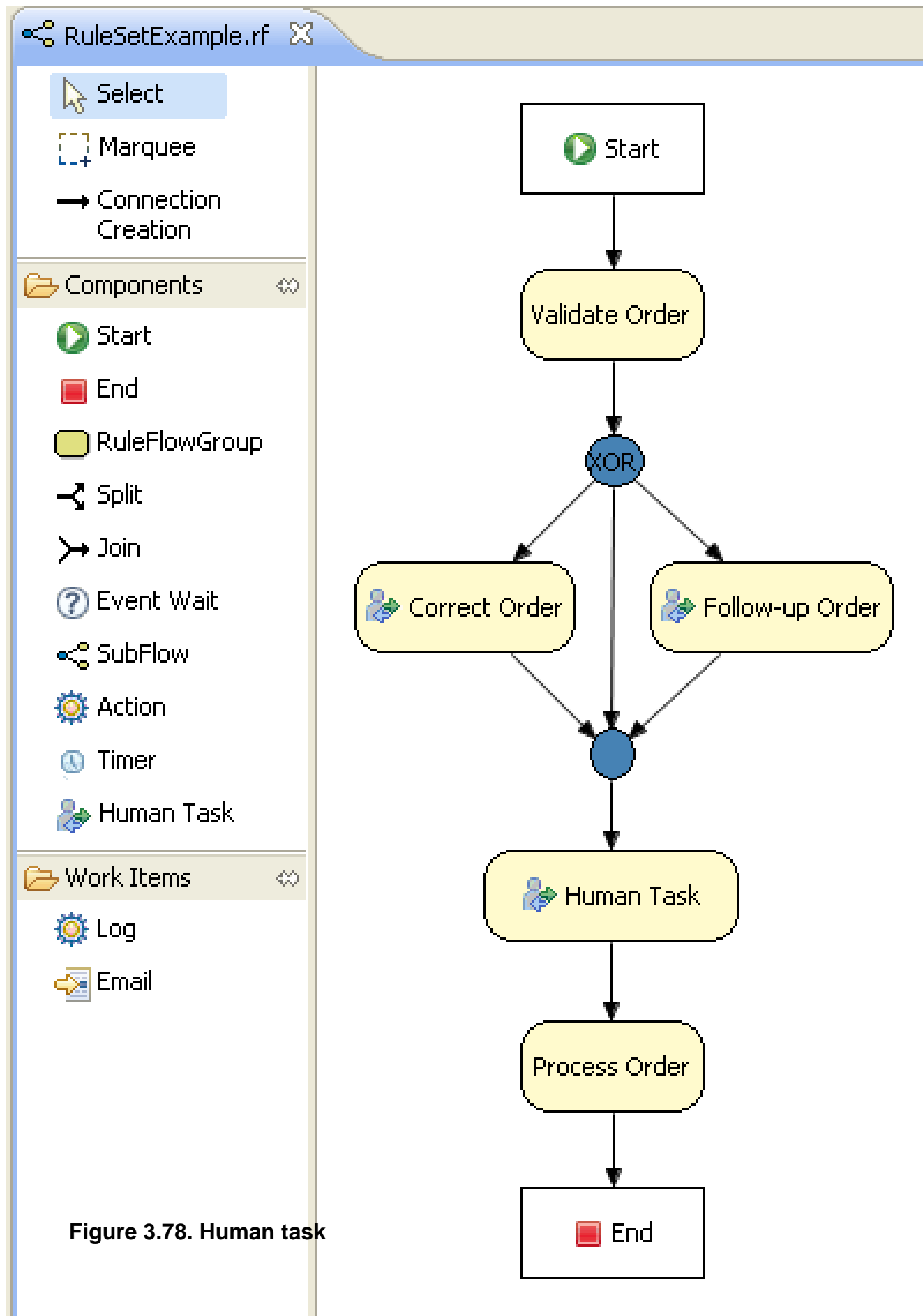


Figure 3.78. Human task

3.5.4.3. Domain Specific Work Items

Domain Specific Work Items are pluggable nodes that users create to facilitate custom task execution. They provide an api to specify a new icon in the palette and gui editor for the tasks properties, if no editor gui is supplied then it defaults to a text based key value pair form. The api then allows execution behaviour for these work items to be specified. By default the Email and Log work items are provided. The Drools flow Manual has been updated on how to implement these.

The below image shows three different work items in use in a workflow, "Blood Pressure", "BP Medication", "Notify GP":

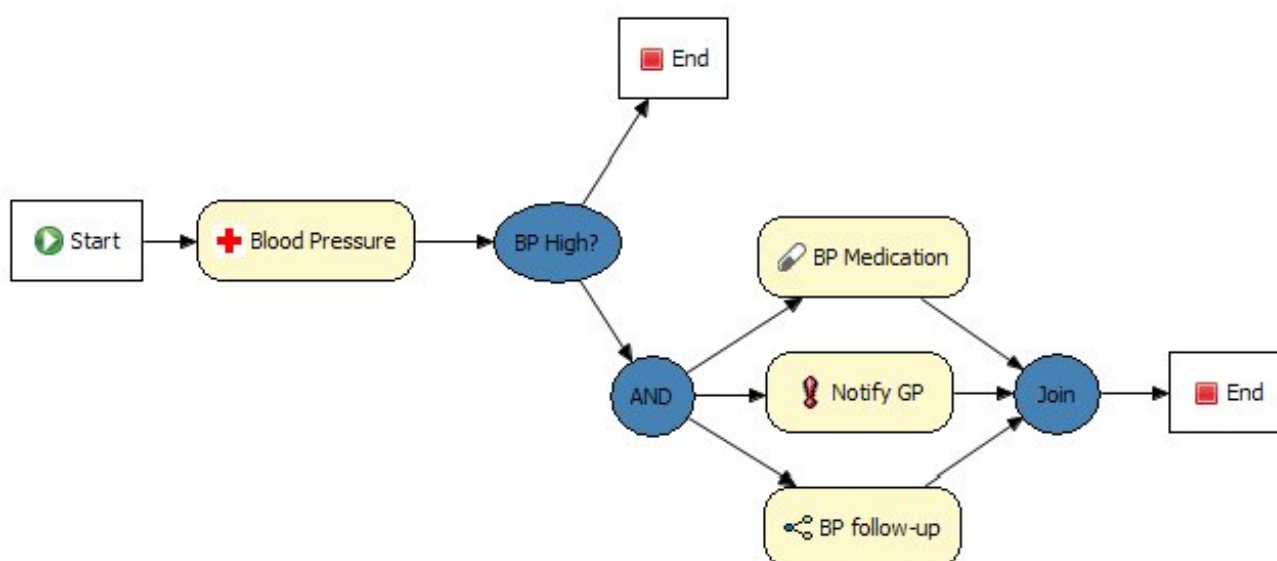


Figure 3.79. Work items

This one owns a new "Notification" work item:

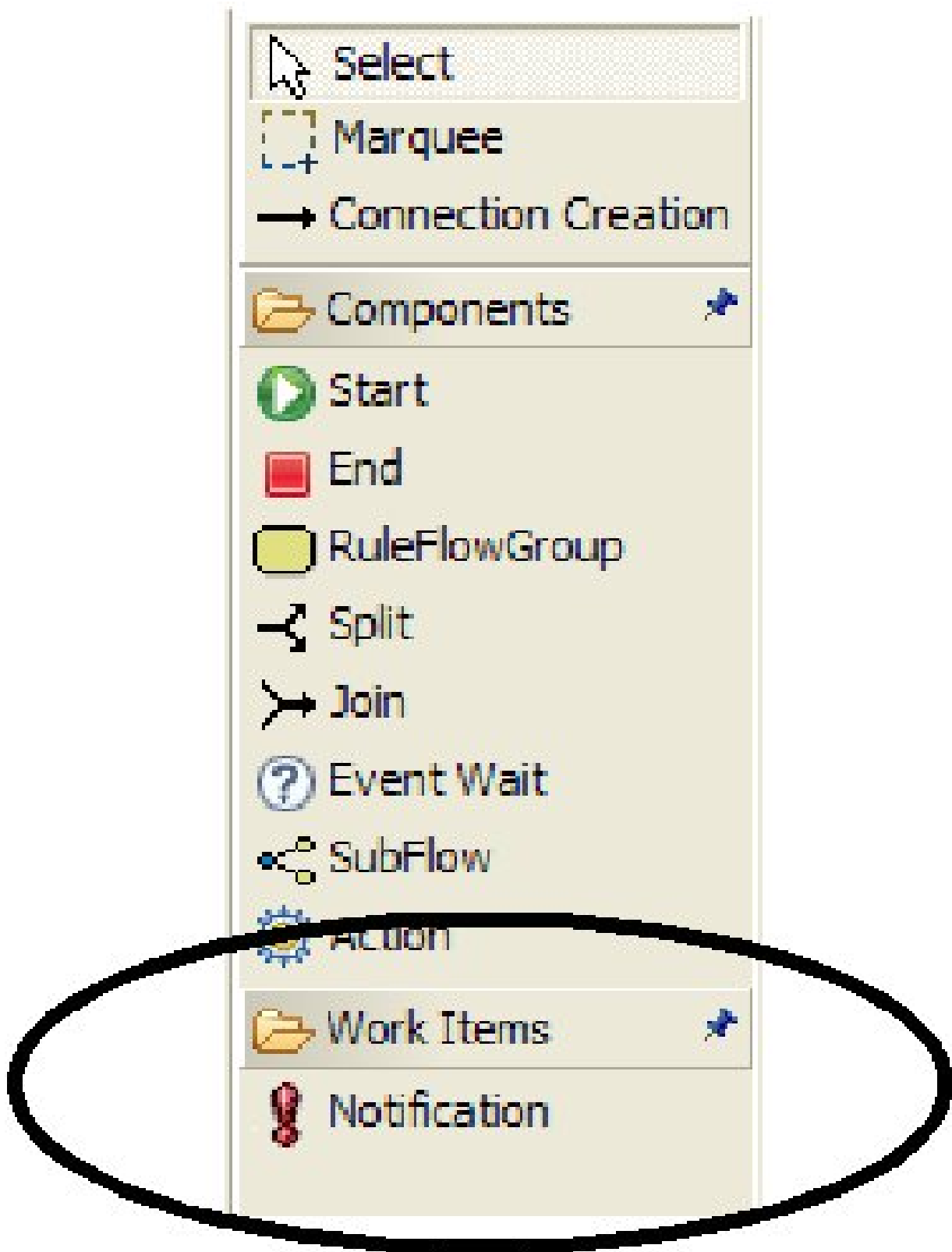


Figure 3.80. Notification

3.5.4.4. Extensible Process Definition Language (ePDL)

Drools 4.0 used Xstream to store its content, which was not easily human writeable. Drools 5.0 introduced the ePDL which is a XML specific to our process language, it also allows for domain specific extensions which has been talked about in detail in this blog posting "Drools Extensible Process Definition Language (ePDL) and the Semantic Module Framework (SMF)". An example of the XML language, with a DSL extension in red, is shown below.

Example 3.49. Example of the XML language

```
<process name="process name" id="process name" package-name="org.domain"
xmlns="http://drools.org/drools-4.0/process"
xmlns:mysdsl="http://domain/org/mydsl"
xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xs:schemaLocation="http://drools.org/drools-4.0/process                drools-
processes-4.0.xsd" >
<nodes>
  <start id="0" />

  <action id="1" dialect="java">
    list.add( "action node was here" );
  </action>

  <mysdsl:logger id="2" type="warn">
    This is my message
  </mysdsl:logger>

  <end id="3" />
</nodes>

<connections>
  <connection from="0" to="1" />
  <connection from="1" to="2" />
  <connection from="2" to="3" />
</connections>

</process>
```

3.5.4.5. Pluggable Nodes

The underlying nodes for the framework are completely pluggable making it simple to extend and to implement other execution models. We already have a partial implementation for OSWorkflow and are working with Deigo to complete this to provide a migration path for OSWorkflow users. Other enhancements include exception scopes, the ability to include on-entry and on-exit actions on various node types, integration with our binary persistence mechanism to persist the state of long running processes, etc. Check out the Drools Flow documentation to learn more.

3.5.4.6. Human tasks

Human task management is very important in the context of processes. While we allow users to plug in any task component they prefer, we have developed a human task management component that supports the entire life cycle of human tasks based on the WS-HumanTask specification.

3.5.4.7. New functions to the Drools Flow language

- Event nodes that allow a process to respond to external events
- Exception handlers and exception handler scopes to handle exceptions that could be thrown
- A ForEach node allows instantiating a section of your flow multiple times, for each element in a collection
- Data type support has been extended
- Timers are integrated with common node types

As a result, new node types and properties have been added to the Drools Flow editor in Eclipse. You can also find examples of these new features in the integration tests (e.g. ProcessExceptionHandlerTest, ProcessTimerTest, etc.).

3.5.4.8. Work items

Our pluggable work item approach allows you to plug in domain-specific work in your process in a declarative manner. We plan to build a library of common work items and already provide an implementation for sending emails, finding files, archiving, executing system commands, logging and human tasks.

3.5.4.9. JPA

Improved support for persistence (JPA) and transactions (JTA).

Example 3.50. An example on how to use persistence and transactions in combination with processes

```
// create a new JPA-based session and specify the JPA entity manager factory
Environment env = KnowledgeBaseFactory.newEnvironment();
env.set( EnvironmentName.ENTITY_MANAGER_FACTORY, persistence.createEntityManagerFactory(
    "my-name" ) );
env.set( EnvironmentName.TRANSACTION_MANAGER, TransactionManagerServices.getTransactionManager() );

StatefulKnowledgeSession session = KnowledgeService.newStatefulKnowledgeSession( env,
    KnowledgeSessionConfiguration may be null, and a default will be used
```

```

int sessionId = ksession.getId();

// if no transaction boundary is specified, the method invocation is executed
// in a new transaction automatically
ProcessInstance processInstance = ksession.startProcess( "org.drools.test.TestProcess" );

// the users can also specify the transaction boundary themselves
UserTransaction ut = (UserTransaction) new InitialContext().lookup( "java:comp/
UserTransaction" );
ut.begin();
ksession.insert( new Person( "John Doe" ) );
ksession.startProcess( "org.drools.test.TestProcess" );
ksession.fireAllRules();
ut.commit();

```

3.5.4.10. Variable Injection

Support direct access to process variables in both MVEL and Java in code constraints and actions, so if you have a variable called "person" in your process, you can now describe constraints like:

Example 3.51. Variable injection example

```

* [Java code constraint] return person.getAge() > 20;
* [MVEL action] System.out.println(person.name);

```

3.5.4.11. Miscellaneous Enhancements

- Process instances can now listen for external events by marking the event node property "external" as true. External events are signaled to the engine using `session.signalEvent(type, eventData)` More information on how to use events inside your processes can be found in the Drools Flow documentation here: <https://hudson.jboss.org/hudson/job/drools/lastSuccessfulBuild/artifact/trunk/target/docs/drools-flow/html/ch03.html#d0e917>
- Process instances are now safe for multi-threading (as multiple thread are blocked from working on the same process instance)
- Process persistence / transaction support has been further improved. Check out the `drools-process/drools-process-enterprise` project for more information.
- The human task component has been extended to support all kinds of data for input / output / exceptions during task execution.

Example 3.52. As a result, the life cycle methods of the task client have been extended to allow content data

```
taskClient.addTask(task, contentData, responseHandler)
taskClient.complete(taskId, userId, outputData, responseHandler)
taskFail.complete(taskId, userId, outputData, responseHandler)

long contentId = task.getTaskData().getDocumentContentId();
taskClient.getContent(contentId, responseHandler);
ContentData content = responseHandler.getContent();
```

- It is now possible to migrate old Drools4 RuleFlows (using the xstream format) to Drools5 processes (using readable xml) during compilation. Migration will automatically be performed when adding the RuleFlow to the KnowledgeBase when the following system property is set:
`drools.ruleflow.port = true`
- The "Transform" work item allows you to easily transform data from one format to another inside processes. The code and an example can be found in the `drools-process/drools-workitems` directory.
- Function imports are now also supported inside processes.
- The history log - that keeps the history of all executed process instances in a database - has been extended so it is now capable of storing more detailed information for one specific process instance. It is now possible to find out exactly which nodes were triggered during the execution of the process instance.
- A new type of join has been added, one that will wait until *n* of its *m* incoming connections have been completed. This *n* could either be hardcoded in the process or based on the value of a variable in the process.
- Improvements have been made to make persistence easier to configure. The persistence approach is based on a command service that makes sure that all the client invocations are executed inside a transaction and that the state is stored in the database after successful execution of the command. While this was already possible in M4 using the commands directly, we have extended this so that people can simply use the normal `StatefulKnowledgeSession` interface but simply can configure the persistence using configuration files. For more details, check out the chapter on persistence in the Drools Flow documentation.

3.5.5. Drools Fusion

Drools 5.0 brings to the rules world the full power of events processing by supporting a number of CEP features as well as supporting events as first class citizens in the rules engine.

3.5.5.1. Event Semantics

Events are (from a rules engine perspective) a special type of fact that has a few special characteristics:

- they are immutable
- they have strong time-related relationships
- they may have clear lifecycle windows
- they may be transparently garbage collected after it's lifecycle window expires
- they may be time-constrained
- they may be included in sliding windows for reasoning

3.5.5.2. Event Declaration

Any fact type can assume an event role, and its corresponding event semantics, by simply declaring the metadata for it.

Example 3.53. Both existing and generated beans support event semantics:

```
# existing bean assuming an event role
import org.drools.test.StockTick
declare StockTick
    @role( event )
end

# generated bean assuming an event role
declare Alarm
    @role( event )
    type : String
    timestamp : long
end
```

3.5.5.3. Entry-Point Stream Listeners

A new key "from entry-point" has been added to allow a pattern in a rule to listen on a stream, which avoids the overhead of having to insert the object into the working memory where it is potentially reasoned over by all rules.

```
$st : StockTick( company == "ACME", price > 10 ) from entry-point "stock stream"
```

Example 3.54. To insert facts into an entry point

```
WorkingMemoryEntryPoint entry = wm.getWorkingMemoryEntryPoint( "stock stream" );  
entry.insert( ticker );
```

StreamTest shows a unit for this.

3.5.5.4. Event Correlation and New Operators

Event correlation and time based constraint support are requirements of event processing, and are completely supported by Drools 5.0. The new, out of the box, time constraint operators can be seen in these test case rules: test_CEP_TimeRelationalOperators.drl

As seen in the test above, Drools supports both: primitive events, that are point in time occurrences with no duration, and compound events, that are events with distinct start and end timestamps.

The complete list of operators are:

- coincides
- before
- after
- meets
- metby
- overlaps
- overlappedby
- during
- includes
- starts
- startedby
- finishes
- finishedby

3.5.5.5. Sliding Time Windows

Drools 5.0 adds support for reasoning over sliding windows of events. For instance:


```
StockTick( symbol == "RHAT" ) over window:time( 60 )
```

The above example will only pattern match the RHAT stock ticks that happened in the last 60 clock ticks, discarding any event older than that.

3.5.5.6. Session Clock

Enabling full event processing capabilities requires the ability to configure and interact with a session clock. Drools adds support for time reasoning and session clock configuration, allowing it to not only run real time event processing but also simulations, what-if scenarios and post-processing audit by replaying a scenario.

Example 3.55. The Clock is specified as part of the SessionConfiguration, a new class that is optionally specified at session creation time

```
SessionConfiguration conf = new SessionConfiguration();
conf.setClockType( ClockType.PSEUDO_CLOCK );
StatefulSession session = ruleBase.newStatefulSession( conf );
```

3.5.5.7. Event Garbage Collection

Since events usually have strong temporal relationships, it is possible to infer a logical time window when events can possibly match. The engine uses that capability to calculate when an event is no longer capable of matching any rule anymore and automatically retracts that event.

3.5.5.8. Time Units Support

Drools adopted a simplified syntax for time units, based on the ISO 8601 syntax for durations. This allows users to easily add temporal constraints to the rules writing time in well known units. Example:

```
SomeEvent( this after[1m,1h30m] $anotherEvent )
```

The above pattern will match if SomeEvent happens between 1 minute (1m) and 1 hour and 30 minutes after \$anotherEvent.

3.5.5.9. Support to event expiration policy

added the ability to define a per-type event expiration policy. In the example below, the StockTick events will expire 10 minutes after they enter the system:

```
declare StockTick @role( event ) @expires( 10m ) end
```

3.5.5.10. Support to temporal operations over arbitrary dates.

Example 3.56. added the ability for point-in-time operators (before, after and coincides) to be used with any arbitrary date field

```
rule "Allow access"
when
  WorkHours( $s : start, $e : end )
  Login( time after $s, time before $e )
then
  // allow access
end
```

3.5.6. Eclipse IDE

- Support multiple runtimes: The IDE now supports multiple runtimes. A Drools runtime is a collection of jars on your file system that represent one specific release of the Drools project jars. To create a runtime, you must either point the IDE to the release of your choice, or you can simply create a new runtime on your file system from the jars included in the Drools Eclipse plugin. Drools runtimes can be configured by opening up the Eclipse preferences and selecting the Drools -> Installed Drools Runtimes category, as shown below.

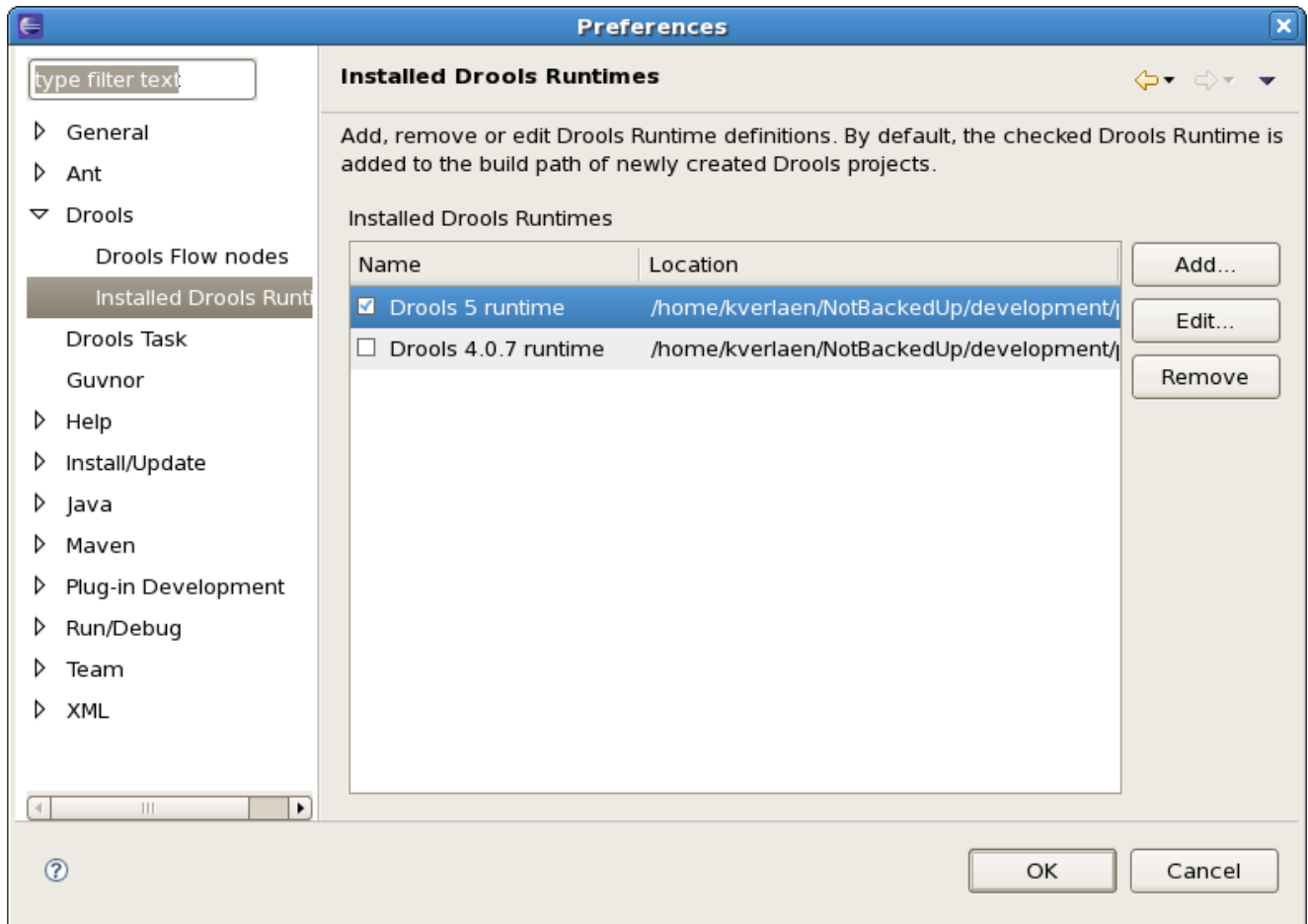


Figure 3.81. Run times

- Debugging of rules using the MVEL dialect has been fixed
- Drools Flow Editor
 - Process Skins allow you to define how the different RuleFlow nodes are visualized. We now support two skins: the default one which existed before and a BPMN skin that visualizes the nodes using a BPMN-like representation: <http://blog.athico.com/2008/10/drools-flow-and-bpmn.html>
 - An (X)OR split now shows the name of the constraint as the connection label
 - Custom work item editors now signal the process correctly that it has been changed

3.6. What is new in Drools 4.0

Drools 4.0 is a major update over the previous Drools 3.0.x series. A whole new set of features were developed which special focus on language expressiveness, engine performance and tools availability. The following is a list of the most interesting changes.

3.6.1. Language Expressiveness Enhancements

- New Conditional Elements: from, collect, accumulate and forall
- New Field Constraint operators: not matches, not contains, in, not in, memberOf, not memberOf
- New Implicit Self Reference field: this
- Full support for Conditional Elements nesting, for First Order Logic completeness.
- Support for multi-restrictions and constraint connectives && and ||
- Parser improvements to remove previous language limitations, like character escaping and keyword conflicts
- Support for pluggable dialects and full support for MVEL scripting language
- Complete rewrite of DSL engine, allowing for full l10n
- Fact attributes auto-vivification for return value restrictions and inline-eval constraints
- Support for nested accessors, property navigation and simplified collection, arrays and maps syntax
- Improved support for XML rules

3.6.2. Core Engine Enhancements

- Native support for primitive types, avoiding constant autoboxing
- Support for transparent optional Shadow Facts
- Rete Network performance improvements for complex rules
- Support for Rule-Flows
- Support for Stateful and Stateless working memories (rule engine sessions)
- Support for Asynchronous Working Memory actions
- Rules Engine Agent for hot deployment and BRMS integration
- Dynamic salience for rules conflict resolution
- Support for Parameterized Queries
- Support for halt command
- Support for sequential execution mode

- Support for pluggable global variable resolver

3.6.3. IDE Enhancements

- Support for rule break-points on debugging
- WYSIWYG support for rule-flows
- New guided editor for rules authoring
- Upgrade to support all new engine features

3.6.4. Business Rules Management System - BRMS

- New BRMS tool
- User friendly web interface with nice WEB 2.0 ajax features
- Package configuration
- Rule Authoring easy to edit rules both with guided editor (drop-down menus) and text editor
- Package compilation and deployment
- Easy deployment with Rule Agent
- Easy to organize with categories and search assets
- Versioning enabled, you can easily replace yours assets with previously saved
- JCR compliant rule assets repository

3.6.5. Miscellaneous Enhancements

- Slimmed down dependencies and smaller memory footprint

3.7. Upgrade tips from Drools 3.0.x to Drools 4.0.x

As mentioned before Drools 4.0 is a major update over the previous Drools 3.0.x series. Unfortunately, in order to achieve the goals set for this release, some backward compatibility issues were introduced, as discussed in the mail list and blogs.

This section of the manual is a work in progress and will document a simple how-to on upgrading from Drools 3.0.x to Drools 4.0.x.

3.7.1. API changes

There are a few API changes that are visible to regular users and need to be fixed.

3.7.1.1. Working Memory creation

Drools 3.0.x had only one working memory type that worked like a stateful working memory. Drools 4.0.x introduces separate APIs for Stateful and Stateless working memories that are called now Rule Sessions. In Drools 3.0.x, the code to create a working memory was:

Example 3.57. Drools 3.0.x: Working Memory Creation

```
WorkingMemory wm = rulebase.newWorkingMemory();
```

In Drools 4.0.x it must be changed to:

Example 3.58. Drools 4.0.x: Stateful Rule Session Creation

```
StatefulSession wm = rulebase.newStatefulSession();
```

The StatefulSession object has the same behavior as the Drools 3.0.x WorkingMemory (it even extends the WorkingMemory interface), so there should be no other problems with this fix.

3.7.1.2. Working Memory Actions

Drools 4.0.x now supports pluggable dialects and has built-in support for Java and MVEL scripting language. In order to avoid keyword conflicts, the working memory actions were renamed as showed below:

Table 3.2. Working Memory Actions equivalent API methods

Drools 3.0.x	Drools 4.0.x
WorkingMemory.assertObject()	WorkingMemory.insert()
WorkingMemory.assertLogicalObject()	WorkingMemory.insertLogical()
WorkingMemory.modifyObject()	WorkingMemory.update()

3.7.2. Rule Language Changes

The DRL Rule Language also has some backward incompatible changes as detailed below.

3.7.2.1. Working Memory Actions

The Working Memory actions in rule consequences were also changed in a similar way to the change made in the API. The following table summarizes the change:

Table 3.3. Working Memory Actions equivalent DRL commands

Drools 3.0.x	Drools 4.0.x
--------------	--------------

assert()	insert()
assertLogical()	insertLogical()
modify()	update()

3.7.2.2. Primitive support and unboxing

Drools 3.0.x did not have native support for primitive types and consequently, it auto-boxed all primitives in its respective wrapper classes. That way, any use of a boxed variable binding required a manual unbox.

Drools 4.0.x has full support for primitive types and does not wrap values anymore. So, all previous unwrap method calls must be removed from the DRL.

Example 3.59. Drools 3.0.x manual unwrap

```
rule "Primitive int manual unbox"
when
    $c : Cheese( $price : price )
then
    $c.setPrice( $price.intValue() * 2 )
end
```

The above rule in 4.0.x would be:

Example 3.60. Drools 4.0.x primitive support

```
rule "Primitive support"
when
    $c : Cheese( $price : price )
then
    $c.setPrice( $price * 2 )
end
```

3.7.3. Drools Update Tool

The Drools Update tool is a simple program to help with the upgrade of DRL files from Drools 3.0.x to Drools 4.0.x.

At this point, its main objective is to upgrade the memory action calls from 3.0.x to 4.0.x, but expect it to grow over the next few weeks covering additional scenarios. It is important to note that it does not make a dumb text search and replace in rules file, but it actually parses the rules file and try to make sure it is not doing anything unexpected, and as so, it is a safe tool to use for upgrade large sets of rule files.

The Drools update tool can be found as a maven project in the following source repository <http://anonsvn.labs.jboss.com/labs/jbossrules/trunk/experimental/drools-update/> you just need to check it out, and execute the maven clean install action with the project's pom.xml file. After resolve all the class path dependencies you are able to run the toll with the following command:

```
java -cp $CLASSPATH org.drools.tools.update.UpdateTool -f <filemask> [-d  
  <basedir>] [-s <sufix>]
```

The program parameters are very easy to understand as following.

- -h,--help, Shows a very simple list the usage help
- -d your source base directory
- -f pattern for the files to be updated. The format is the same as used by ANT: * = single file, directory ** = any level of subdirectories EXAMPLE: src/main/resources/**/*.drl = matches all DRL files inside any subdirectory of /src/main/resources
- -s,--sufix the sufix to be added to all updated files

3.7.4. DSL Grammars in Drools 4.0

It is important to note that the DSL template engine was rewritten from scratch to improve flexibility. One of the new features of DSL grammars is the support to Regular Expressions. This way, now you can write your mappings using regexp to have additional flexibility, as explained in the DSL chapter. Although, now you have to escape characters with regexp meaning. Example: if previously you had a matching like:

Example 3.61. Drools 3.0.x mapping

```
[when]{}- the {attr} is in [ {values} ]={attr} in ( {values} )
```

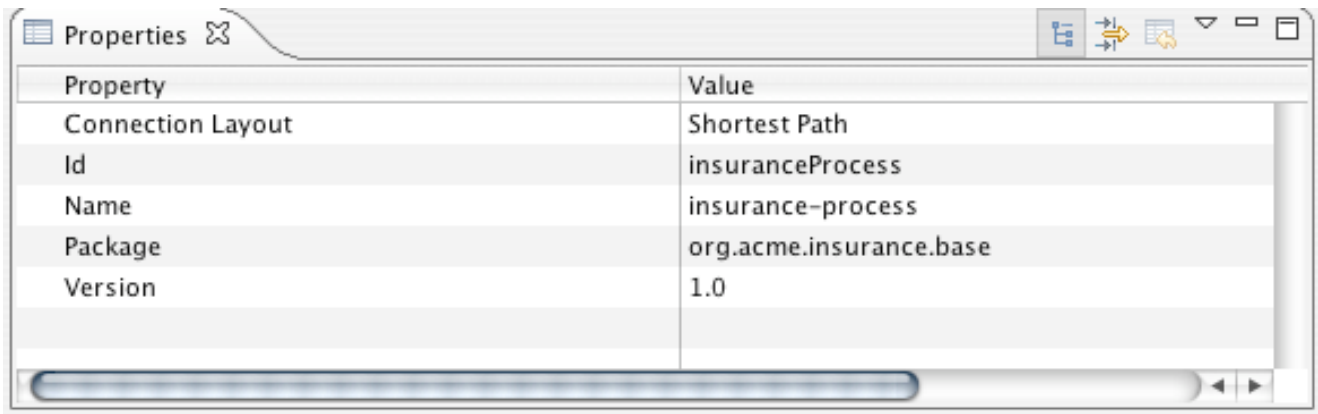
Now, you need to escape '[' and ']' characters, as they have special meaning in regexps. So, the same mapping in Drools 4.0 would be:

Example 3.62. Drools 4.0.x mapping with escaped characters

```
[when]{}- the {attr} is in \[ {values} \]={attr} in ( {values} )
```

3.7.5. Rule flow Update for 4.0.2

The Rule flow feature was updated for 4.0.2, and now all your ruleflows must declare a package name.



Property	Value
Connection Layout	Shortest Path
Id	insuranceProcess
Name	insurance-process
Package	org.acme.insurance.base
Version	1.0

Figure 3.82. Rule Flow properties

Chapter 4. Drools compatibility matrix

The following table shows the compatibility between different versions of Drools, jBPM and Guvnor.

Table 4.1. Compatibility matrix

Drools	jBPM	Guvnor
5.4.0.Beta1	5.2.0.Final	5.4.0.Beta1
5.3.0.Final	5.1.2.Final	5.3.0.Final
5.3.0.CR1	5.1.1.Final	5.3.0.CR1
5.2.1.Final	5.1.1.Final	5.2.1.Final
5.2.0.Final	5.1.0.Final	5.2.0.Final
5.2.0.CR1	5.1.0.CR1	5.2.0.CR1
5.2.0.M2	5.1.0.M1	5.2.0.M2

If you combine the versions specified in any *single* row of this table, they are compatible.

Index

E

Eclipse, 15

