Ruleflow

Ruleflow-group

Rules can be assigned to a ruleflow-group. Rules in a ruleflow-group can only fire when that group has been activated. This means that ruleflow-groups act as a kind of bucket: as long as the ruleflow-group is deactivated, activations are not put on the agenda directly but stored in the bucket. Once it becomes activated, all its activations are transferred to the agenda. Deactivating a ruleflow-group automatically removes any possible activations of that ruleflow-group back into the bucket.

Rules can define in which ruleflow-group they belong by using the ruleflow-group attribute:

```
rule "Sample Rule" ruleflow-group "Group1"
  when
    # conditions
  then
    # actions
end
```

Ruleflow-groups are deactivated by default, when creating a new working memory. You can activate or deactivate a ruleflow-group through the agenda:

```
workingMemory.getAgenda().activateRuleFlowGroup("Group1");
workingMemory.getAgenda().deactivateRuleFlowGroup("Group1");
```

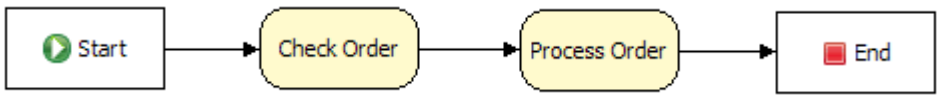
You could even activate or deactivate ruleflow-groups from inside other rules.

A ruleflow-group automatically deactivates once it contains no more activations. This behavior can however be configured:

```
workingMemory.getAgenda().getRuleFlowGroup("Group1").setAutoDeactivate(false);
```

Ruleflow

We go even further by allowing you to specify a flow chart describing when to activate and deactivate certain ruleflow-groups. These flow charts allow sequencing of ruleflow-groups (the ruleflow-group gets activated after the first one has been deactivated), parallelism (multiple ruleflow-groups are active simultaneously), conditional activation (ruleflow-groups are only activated if a certain condition evaluates to true), looping, etc. These ruleflows are specified using a graphical ruleflow chart:



For example, this ruleflow uses two ruleflow-groups, one containing rules that are responsible for validating an order, another capable of processing the order. This ruleflow then makes sure that orders are only processed after they have been checked.

Ruleflows have a name, a version and a unique id. They are internally represented as object graphs

of nodes and connections inbetween those nodes. They are internally stored as xml files. Whenever you create a new ruleflow using the Jboss Rules IDE, the ruleflow itself is stored as two separate files: one (with extension .rfm) containing the definition of the ruleflow and one (with extension .rf) containing all the graphical information (like the locations of the nodes) that is only used for visualizing the ruleflow.

You can add ruleflows to a RuleBase the same way as you add rules to a package:

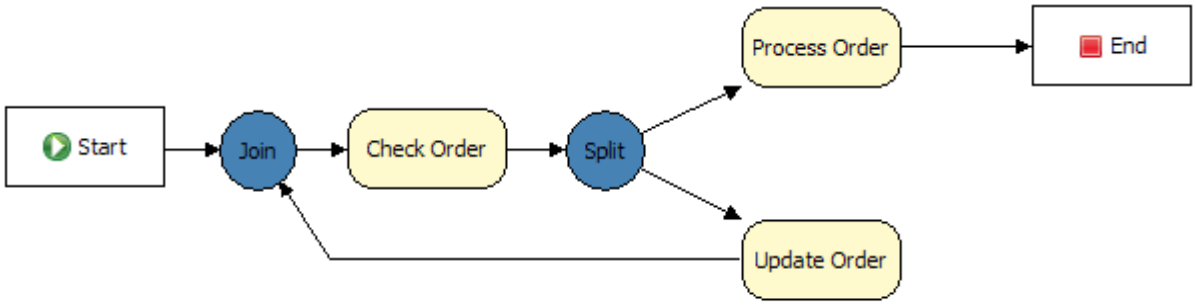
```
// create a package containing all the rules
final PackageBuilder builder = new PackageBuilder();
builder.addPackageFromDrl( new InputStreamReader(
    getClass().getResourceAsStream( "rules.drl" ) ) );
final Package pkg = builder.getPackage();

// create a new process containing the ruleflow
ProcessBuilder processBuilder = new ProcessBuilder();
processBuilder.addProcessFromFile(new InputStreamReader(
    getClass().getResourceAsStream( "ruleflow.rfm" ) ) );

// add the package and the processes to the RuleBase
final RuleBase ruleBase = RuleBaseFactory.newRuleBase();
ruleBase.addPackage( pkg );
ruleBase.addProcess( processBuilder.getProcesses()[0]);
// create a new working memory for this RuleBase
final WorkingMemory workingMemory = ruleBase.newWorkingMemory();
```

Ruleflows can be started from the working memory (referencing their unique id):

```
workingMemory.startProcess(1);
```

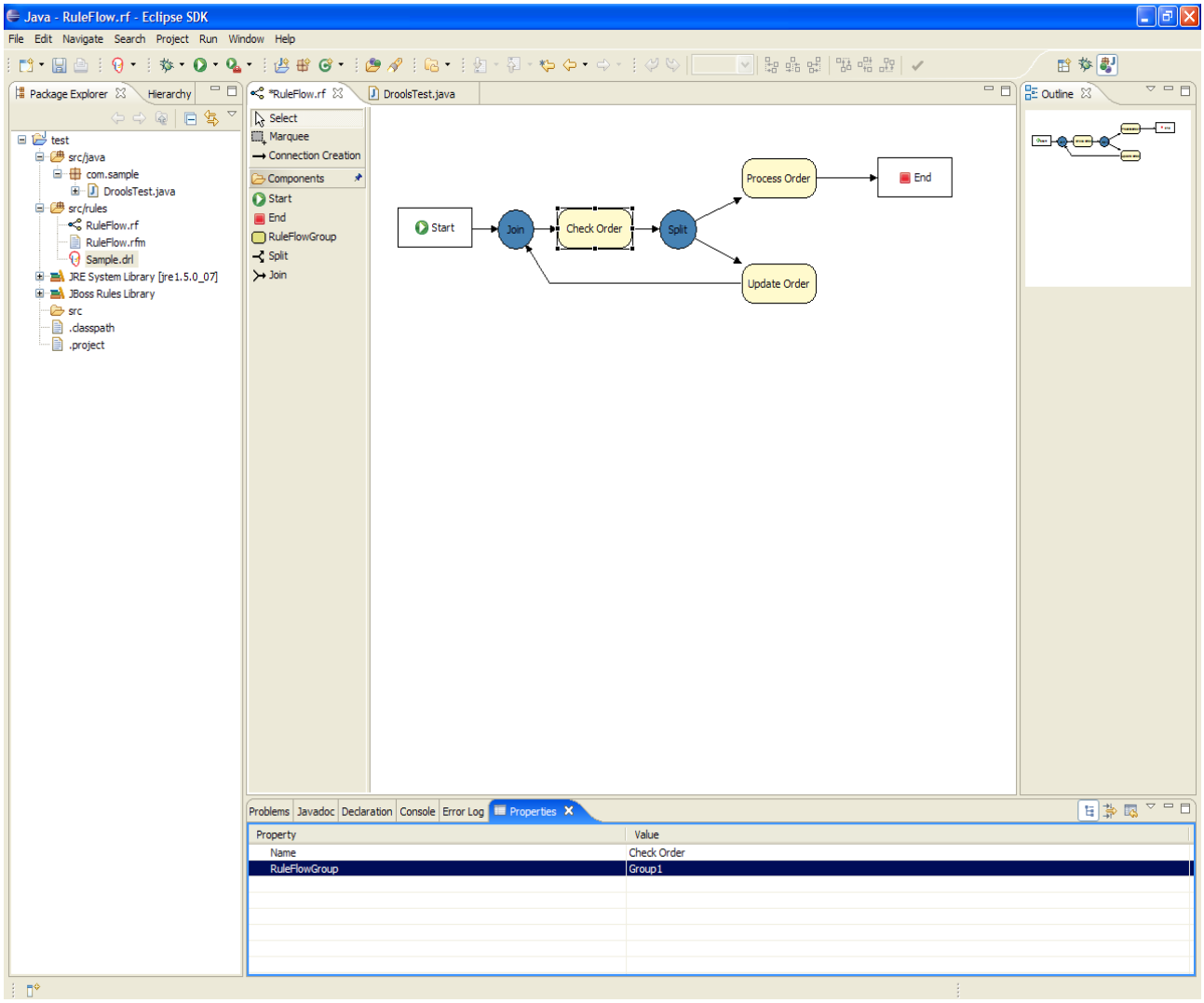
A ruleflow can be started like this from start-up code or even from inside rules. Ruleflows can be started more than once.

More advanced ruleflows can be created by using splits and joins. Different types of splits (AND, XOR and OR) and joins (AND, XOR) are possible.



The JBoss Rules IDE support creating ruleflows based on the Graphical Editing Framework. As shown in the screenshot, you can create ruleflows by selecting the appropriate elements you want to draw in the palette and using them on the canvas. The outline view shows a miniature representation of the entire flow. The properties view shows the properties of the selected element. For example, a RuleFlowGroup node has a property RuleFlowGroup, where the name of the ruleflow-group should be specified. Similarly, split and join nodes have a type (AND, XOR, OR),

and a split can possibly be associated with constraints for each of its outgoing connections (in the case of an XOR or an OR split).



At this point, ruleflow-groups should not be reused in more than one ruleflow, and you should not start a new instance of a process before the previous one has ended.