Book: Drools JBoss Rules 5.x Developer’s Guide

Chapter 1

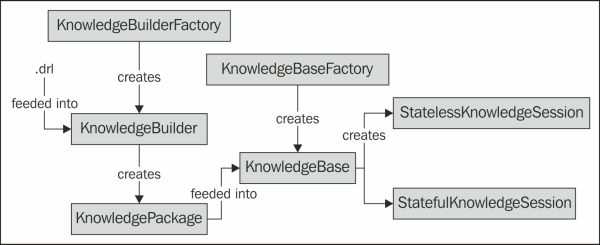
* Drools is a business logic integration platform(Blip)
* Since version 2.0 Drools has been based on the Rete algorithm
* The Drools platform consist of four main modules:
  + Drools Expert: this is a rule engine itself
  + Drools Fusion: This is a complex event processing (CEP) module.
  + jBPM: Workflow combines rules and processing together.
  + Drools Guvnor: This is a BRMS (Not covered in this book).

Chapter 2: Writing Basic Rules

**Structure of Rule:**

1. Package eg. “package droolsbook;”
2. Rule name eg. “rule “basic rule” “
3. Condition eg.
   1. When Account (balance < 100)
4. Action/Consequence eg.
   1. Then <do something>
5. End eg end.

**Executing rules**

* KnowledgeBase:
  + Org.drools.KnowledgeBAse
  + This is an interface that manages a collection of rules, processes, and internal types.
  + In Drools these are commonly referred to as Knowledge definitions or knowledge.
  + The knowledge definitions are grouped into knowledge packages.
  + The knowledge definitions can be added or removed
  + The main purpose of the KnowledgeBase object is to store and reuse them, because their creation is expensive.
  + The knowledgebase object provides methods for creating knowledge sessions. Their creation is very lightweight.
* By default, the KnowledgeBase object maintains a reference to all created knowledge sessions. This is to accommodate updates to KnowledgeBase at runtime.
  + Drools has one implementation of this interface
  + This implementation can be serialized.
  + We can reuse the serialized KnowlegeBase object instead of creating a new one every time.
  + This implementation is based on the **Rete algorithm.**
* The knowledgebase object is then in turn used to create a stateful knowledge session
* **StatefulKnowledgeSession**
  + **Org.drools.runtime.StatefulKnowlegeSession**
  + This is an interface that acts as an entry/exit point to the Drools engine.
  + It has methods for inserting, updating, or retracting facts.
  + The StatefulKnowledgeSession object is also used to set a session’s global variables.
  + The **fireAllRules** method is used to execute all rules.
  + Event handlers can be registered on the KnowledgeBase object for auditing, debugging, or other purposes.
  + When you finish working with the StatefulKnowledgeSession object, do not forget to call the **dispose method**, otherwise this object can’t be garbage collected, as it also has a transient reference to parent KnowledgeBase. (ie why you can do session.getKnowledgebase())
* If we want to reason over an object, we have to insert it into the session. The object is also sometimes referred to as **a fact.**
* **The KnowledgeBuilder**
  + Org.drools.builder.KnowledgeBuilder
  + This interface is responsible for building KnowledgePackages from knowledge definitions (rules, processes, and types).
  + The knowledge definitions can be in various formats.
  + If there were any problems with building, the KnowledgeBuilder object will report errors through these two methods: **hasErrors** and **getError**.
  + KnowledgePackages form KnowlegeBases.
* After the package is built it is added to the newly created KnowledgeBase. It can then be used to create knowledge sessions.
* 

**Variables in Rules**

* **Rules** can declare variables as follows:
  + **$account : Account ( $type : type) //** this is a condition that matches every Account and creates two variables: $account and $type.
  + Got it: this is a variable called $account of type ACCOUNT.
  + $type is a variable representing the account property called Account.Type.
  + The $ symbol is a common naming practice.
* Variables can be declared upfront for later use eg
  + $account : Account() – create a variable of type Account called $account.
  + Customer ( account == $account ) – if customer.account is equal to this account… objects or facts, mapping it to the account property/field in the customer object.
  + Official explanation: These are conditions with a join and match every Customer with his/her Account.

**Types**

**Rule consequence**

* When all conditions in a rule are met, the rule becomes activated. After all rules are evaluated and some of them become activated, the rule engine will pick one activated rule and will execute its consequence. We say that a rule has fired.
* The activated rule is chosen based on a conflict resolution strategy. The conflict resolution strategy uses multiple criteria to decide which rule to fire.
* After the rule has fired, the engine re-evaluates any changes that have been made by the previous rule’s consequence execution.
* This may activate or deactive other rules. Thisprocess repeats again until there is no activated rule.
* Consequence represents the actions that a rule will do once it fires. It can contain any valid Java or Mvel code.
* NOTE: It is considered a very bad practice to have conditional logic (if statements) within a rule consequence. Most of the time, a new rule should be created.
* The rule condition should contain simple actions; the facts can be modified, which may cause other rules to fire. Drools comes with these convenient methods for working with the current KnowledgeSession:
  + Modify: This is for updating existing facts in the session. For example, a rule that adds interest for deposited money:
    - Rule “interest calculation”
    - No-loop
    - When
      * $account : Account()
    - Then
      * Modify($account){
        + setBalance((long) ($account.getBalanc e() \* 1.03));
    - end
  + Note that Drools has another way to update facts in the session, the update method. However, the modify method should always be preffered, as it provides more information to the engine as to what has been updated.
  + Insert: This is used for inserting new facts into the session
  + Retract: this is used for removing existing facts from the session.
* When a rule is inserted/updated/retracted, the rule engine works with the new set of facts; rules may be activated/deactivated.

**Rule Attributes**

* Rule attributes are used to modify/enhance the standard rule behavior. All attributes are defined between the “rule” and “when” keywords. For example
  + Rule “rule attributes”
  + Salience 100
  + Dialect “mvel”
  + No-loop
    - When
      * //conditions
    - Then
      * //consequence
    - End
* The salience (priority) attribute:
  + The salience strategy is used by the conflict resolution strategy to decide which rule to fire.
  + By default it is the main criterion
  + We can use salience to define the order of firing rules.
  + It has one attribute which takes any expression that returns a number of type int (positive as well as negative numbers are valid).
  + The higher the value, the more likely a rule will be picked up by the conflict resolution strategy to fire.
    - Eg salience ($account.balance \* 5)
  + The default salience value if 0.
* The no-loop attribute
  + This attribute informs the rule engine that a rule should be activated only once per matched facts. For example, the rule in Code listing above will be activated only once per each Account instance. If it didn’t have the no-loop attribute, it will cause an infinite loop, because the consequence is updating the $account fact.
  + Note however that, if other rules modify the $account fact, our rule will reactivate.
  + The no-loop attribute has quite a limited use as it prevents only the rule to activate itself.
  + A more stronger alternative is the loc-on-active attribute
* Dialect – see above

Chapeter 3: Validating