JBoss Drools
Open-Source Business Logic Platform
Kris Verlaenen
Drools Developer
Overview

- Drools Expert
  - What is a rule?
  - Why rules?
  - How does a rule engine work?
- Drools Eclipse IDE
- Drools Guvnor
- Drools Flow
- Drools Fusion
What is a rule?

A principle or regulation governing conduct, action, procedure, arrangement, etc.

rule “RuleName”
  when
    // conditions
  then
    // actions
What is a rule?

- Rule conditions = Pattern matching
  
  p: Person ( age > 18, $ssn: ssn )
  DriversLicense ( type == "Category2", ssn == $ssn )

- Rule consequence = Action
  
  System.out.println(p.getName());
  insert(new Person());
Drools language features

• Rule conditional elements
  - And, Or, Exists, Not
  - Accumulate, Collect
  - From
  - Forall
  - Temporal rules

• Rule consequence
  - Pluggable dialects (Java, MVEL)

• Functions, globals, queries, etc.
Drools Rule Formats

- DRL
- Domain-specific language
- Decision tables
- Guided rule editor
- XML
import org.drools.Person

global java.util.List myList

rule "RuleName" salience 20
when
    p: Person( age > 20, name == "John" )
    r: Request( personId == (p.id) )
then
    myList.add( r );
Domain-specific Language

```plaintext
expander DSL.dsl

rule "Your First Rule"
  when
    There is a Notification of type "{type}" 
    There is a Person 
    - with age between {x} and {y}
  then
    - with age between {x} and {y} 
    - with name "{name}"
    Instance is at least {number} and field is "{value}"
    There is a Notification of type "{type}"
    There is a Person
    There is an Instance with field of "{value}"
    There is no current Instance with field : "{value}"
end
```
## Decision Tables

<table>
<thead>
<tr>
<th>Base pricing rules</th>
<th>Age Bracket</th>
<th>Location risk profile</th>
<th>Number of prior claims</th>
<th>Policy type applying for</th>
<th>Base $ AUD</th>
<th>Record Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Young safe package</td>
<td>18, 24</td>
<td>LOW</td>
<td>1</td>
<td>COMPREHENSIVE</td>
<td>450</td>
<td>Priors not relevant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MED</td>
<td>1</td>
<td>FIRE_THEFT</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MED</td>
<td>0</td>
<td>COMPREHENSIVE</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LOW</td>
<td>1</td>
<td>FIRE_THEFT</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LOW</td>
<td>0</td>
<td>COMPREHENSIVE</td>
<td>150</td>
<td>Safe driver discount</td>
</tr>
<tr>
<td>Young risk</td>
<td>18, 24</td>
<td>MED</td>
<td>1</td>
<td>COMPREHENSIVE</td>
<td>700</td>
<td>Location risk</td>
</tr>
<tr>
<td></td>
<td>18, 24</td>
<td>HIGH</td>
<td>0</td>
<td>COMPREHENSIVE</td>
<td>700</td>
<td>Location risk</td>
</tr>
<tr>
<td></td>
<td>18, 24</td>
<td>HIGH</td>
<td>0</td>
<td>FIRE_THEFT</td>
<td>550</td>
<td>Location risk</td>
</tr>
<tr>
<td>Mature drivers</td>
<td>25, 30</td>
<td></td>
<td>0</td>
<td>COMPREHENSIVE</td>
<td>120</td>
<td>Cheapest possible</td>
</tr>
<tr>
<td></td>
<td>25, 30</td>
<td></td>
<td>1</td>
<td>COMPREHENSIVE</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td></td>
<td>25, 30</td>
<td></td>
<td>2</td>
<td>COMPREHENSIVE</td>
<td>590</td>
<td></td>
</tr>
<tr>
<td></td>
<td>25, 35</td>
<td></td>
<td>3</td>
<td>THIRD PARTY</td>
<td>800</td>
<td>High risk</td>
</tr>
</tbody>
</table>
Guided Rule Editor

Guided rule editor

WHEN

Person [p]

+ age is less than 18

THEN

Modify [p] + name

(options)

salience 10
<rule name="simple_rule">
  <rule-attribute name="salience" value="10" />
  <lhs>
    <pattern identifier="$i" object-type="Integer">
      <from>
        <accumulate>
          <pattern object-type="Cheese"></pattern>
          <init> int total = 0; </init>
          <action> total += $cheese.getPrice(); </action>
          <result> new Integer( total ) ); </result>
        </accumulate>
      </from>
    </pattern>
  </lhs>
  <rhs> list1.add( $cheese ); </rhs>
</rule>
Why rules?

- Separate logic from application
- Understandability
  - Declarative, higher-level
- Speed and scalability
  - ReteOO
- Global enforcement, maintainability and agility
  - Logic is centralized, embrace change
- Traceability
How does a rule engine work?

Production rule engine using forward chaining

Rules \rightarrow \text{Rule Base} \rightarrow \text{Pattern Matcher} \rightarrow \text{Agenda} \rightarrow \text{Working Memory} \rightarrow \text{Facts}

insert, update, remove
ReteOO

- Rule base is compiled into a discrimination network
- Data propagates through network
- Node sharing
Other Drools Features

• Static rule analysis
  - Detect redundancy, completeness, etc.
• Audit (event listeners)
• JSR-94 compliant (Java Rule Engine API)
• Binary persistence strategy
• Community extensions
  - Uncertainty
  - Planning
Drools Code example

```java
// Create a new rule base
PackageBuilder builder = new PackageBuilder();
builder.addPackageFromDrl( ... );
RuleBase ruleBase = RuleBaseFactory.newRuleBase();
ruleBase.addPackage( builder.getPackage() );

// Create working memory
WorkingMemory workingMemory = ruleBase.newStatefulSession();

// Insert facts
Message message = new Message( "Hello World" );
workingMemory.insert( message );

// Execute
workingMemory.fireAllRules();
```
Overview

• Drools Expert
  - What is a rule?
  - Why rules?
  - How does a rule engine work?

• Drools Eclipse IDE

• Drools Guvnor

• Drools Flow

• Drools Fusion
Eclipse IDE

• Assist developer in
  – Authoring
  – Executing
  – Testing
  – Debugging
  – Managing

• Extends Eclipse workbench with specific views, editors, wizards, perspective, actions, builder, etc.
package com.sample


rule "Hello World"
when
    m : Message( status == Message.HELLO, message : String )
then
    System.out.println( m.setMessage( m.getMessage() ) );
    m.setStatus( Message.UPDATE );
    update( m );
end

rule "GoodBye"
no-loop true
when
    m : Message( status == Message.GOODBYE, message : String )
then
    System.out.println( m.setMessage( m.getMessage() ) );
end
import org.drools.examples.State;

rule Bootstrap
when
  a : State(name == "A", state == State.NOTRUN)      
then
  System.out.println(a.getName() + " finished");    
  a.setState(State.FINISHED);                       
end

rule "A to B"
when
  State(name == "A", state == State.FINISHED)      
  b : State(name == "B", state == State.NOTRUN)   
then
  b.setState(State.FINISHED);                       
  System.out.println(b.getName() + " finished");  
end

rule "B to C"
when
  State(name == "B", state == State.FINISHED)      
  c : State(name == "C", state == State.NOTRUN)   
then
  System.out.println(c.getName() + " finished");  
end
Overview

- **Drools Expert**
  - What is a rule?
  - Why rules?
  - How does a rule engine work?
- **Drools Eclipse IDE**
- **Drools Guvnor**
- **Drools Flow**
- **Drools Fusion**
Drools Guvnor

Web-based rule management, storage, editing and deployment environment

- Rule editing
  - text, guided, decision tables, etc.
- Version control
- Categorization
- Build and deploy
- Scenarios
Guvnor Rule Editing
Guvnor Categorization

Current categories:
- HR
- Finance
- Draft

Create a new category:
Delete the currently selected category:
Guvnor Deployment

The page at http://localhost:8080 says:

The snapshot called: NewSnapshot was successfully created.

Create a snapshot for deployment.

A package snapshot is essentially a read only 'locked in' and labelled view of a package at a point in time, which can be used for deployment.

Choose or create snapshot name: NEW/NewSnapshot
Comment: commenting

Create new snapshot
Guvnor Test Scenarios
Overview

• Drools Expert
  – What is a rule?
  – Why rules?
  – How does a rule engine work?
• Drools Eclipse IDE
• Drools Guvnor
• Drools Flow
• Drools Fusion
Drools Flow

A workflow engine combining processes and rules

• Integration
  - From loose coupling (decision services)
  - To advance integration (process rules)

• Unification
  - Rules and processes are different types of business knowledge assets
  - Tooling (IDE, repository, management, etc.)
Why use rules in processes

- **Simplicity**
  - Complex decisions are sometimes easier to specify using rules

- **Separate life cycle and different scope => Agility**
  - Change rules without having to change process, rules can be reused across processes or outside processes

- **Declarative and higher-level**
  - Focus on what instead of how, rules are additive

- **Granularity**
  - Easy to write rules for specific circumstances, processes should be more generic

- **Performance**
Drools Flow

- Unifies rules and processes in a single engine
  - Ability to use rules everywhere in your process
    - Decision nodes, constraints, exception and event handling, task assignment, etc.
  - Processes and rules see, reason and react on the same data
    - No data passing or synchronization
  - Processes and rules interact
  - Integrated tooling
Drools Flow

- **Control flow**
  - Sequence, Parallelism
  - Choice, Looping
- **Data flow (variables)**
- **Nodes**
  - Action, Milestone (State), Subflow, Timer, Composite, Human Tasks
  - Exceptions, Timers
Additional Features

• Extensible process framework
  - Reusable set of core nodes
• Binary persistence of process instances
• Domain-specific work items
  - Plug in your own work nodes
  - Simplifies integration and testing
• Integrated debug and audit
Domain-specific Processes
Integrated debug and audit
Overview

- Drools Expert
  - What is a rule?
  - Why rules?
  - How does a rule engine work?

- Drools Eclipse IDE

- Drools Guvnor

- Drools Flow

- Drools Fusion
Drools Fusion

*Temporal reasoning system to provide Complex Event Processing (CEP) capabilities*

- Processing multiple events from an event cloud with the goal of identifying the meaningful events within the event cloud.
  - Event detection
  - Event correlation
  - Event abstraction
Requirements for event processing

• Event Detection
  – From an event cloud, select all the meaningful events and only them.

• [Temporal] Event Correlation
  – Ability to correlate events and facts declaring temporal and non-temporal constraints between them.
  – Ability to reason over event aggregation

• Event Abstraction
  – Ability to declare composite complex events from simple atomic events
Requirements for event processing

• Event Detection
  – Leverages all the power and expressiveness of the rule language
  – Extension for stream support: entry-points

• [Temporal] Event Correlation
  – Temporal extension to Rete Algorithm: before, after, meets, met-by, overlaps, overlapped-by, during, contains, starts, started-by, finishes, finished-by, concurrent

• Event Abstraction
  – Sliding window support
Overview

• Drools Expert
  – What is a rule?
  – Why rules?
  – How does a rule engine work?
• Drools Eclipse IDE
• Drools Guvnor
• Drools Flow
• Drools Fusion
Questions?

Drools Homepage
http://www.jboss.org/drools/

Drools Blog
http://blog.athico.com/

Drools Chat
irc.codehaus.org #drools

Drools Mailing List
rules-users@lists.jboss.org