The SkyNet funding bill is passed.
The system goes online on August 4th, 1997.
Human decisions are removed from strategic defense.
SkyNet begins to learn at a geometric rate.
It becomes self-aware at 2:14am Eastern time, August 29th.
In a panic, they try to pull the plug.
And, Skynet fights back.

JBoss Drools - Viva Le Drools
Declarative Behavioural Modelling
An Integrated AI approach

Mark Proctor
Project Lead
Drools is a business rule management system (BRMS) and an enhanced Rules Engine implementation, ReteOC, based on Charles Forgy's Rete algorithm tailored for the Java language. More importantly, Drools provides for Declarative Programming and is flexible enough to match the semantics of your problem domain with Domain Specific Languages, graphical editing tools, web based tools and developer productivity tools.

Why Choose Drools

Open Source

Drools has a business friendly open source license that makes it free to download, use, embed, and distribute.

Declarative Programming

Allow you to say "What to do" not "How to do it".

Accessible business rules

Drools is friendly to both developers and business users. DSLs allow developers to write almost natural language semantics for rule authors. GUIs and visual metaphors (RuleFlow, Decision tables in Spreadsheets) also reduce the gap between business and IT. A web based BRMS (Business Rule Management System) provides GUIs for managing rule assets.

Logic and Data Separation

Rule engines are the ultimate in logic and data de-coupling. Increases performance, decreases the number of days it takes to deploy rules, and reduces the number of bugs obtained during the deployment.
Drools now has 1725 unit and integration tests

Posted by Mark Proctor

One of the great things about Open Source is we are totally open and transparent, so it's very easy to make a judgement on the level of quality of the software and the efforts gone into QA. On this note we would like to bring to everyone's attention that Drools now has 1725 unit and integration tests - which I think is high by anyone's standard - none of these tests were produced by code generation. This report is shown as part our Hudson built test results page, https://hudson.jboss.org/hudson/job/drools/983/testReport/.

Our Hudson build server, https://hudson.jboss.org/hudson/job/drools/, builds Drools after every commit and makes distribution zips publicly available here, so you can always get the latest trunk build for your own testing.

Technorati Links • Subscribe to this feed • Save to del.icio.us • Digg This! • Share on Facebook • Stumble it! • Post to dzone

at 4:46 PM 0 comments  ▶ Links to this post
Drools Classes

Account
- long accountNo
- double balance

Cashflow
- Date date
- double amount
- int type
- long accountNo

AccountingPeriod
- Date start
- Date end
increasing balance for AccountPeriod Credits

```
select * from Account acc, Cashflow cf, AccountPeriod ap
where acc.accountNo == cf.accountNo and
  cf.type == CREDIT
  cf.date >= ap.start and
  cf.date <= ap.end

trigger : acc.balance += cf.amount
```

decreasing balance for AccountPeriod Debits

```
select * from Account acc, Cashflow cf, AccountPeriod ap
where acc.accountNo == cf.accountNo and
  cf.type == DEBIT
  cf.date >= ap.start and
  cf.date <= ap.end

trigger : acc.balance -= cf.amount
```
Creating Views with Triggers

<table>
<thead>
<tr>
<th>date</th>
<th>amount</th>
<th>type</th>
<th>accountNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-Jan-07</td>
<td>100</td>
<td>CREDIT</td>
<td>1</td>
</tr>
<tr>
<td>2-Feb-07</td>
<td>200</td>
<td>DEBIT</td>
<td>1</td>
</tr>
<tr>
<td>18-May-07</td>
<td>50</td>
<td>CREDIT</td>
<td>1</td>
</tr>
<tr>
<td>9-Mar-07</td>
<td>75</td>
<td>CREDIT</td>
<td>1</td>
</tr>
</tbody>
</table>

AccountingPeriod

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-Apr-07</td>
<td>30-Jun-07</td>
</tr>
</tbody>
</table>

Account

<table>
<thead>
<tr>
<th>accountNo</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

increase balance for AccountPeriod Credits

select * from Account acc,
    Cashflow cf, AccountPeriod ap
where acc.accountNo == cf.accountNo and
    cf.type == CREDIT
    cf.date >= ap.start and
    cf.date <= ap.end

trigger : acc.balance += cf.amount

decrease balance for AccountPeriod Debits

select * from Account acc,
    Cashflow cf, AccountPeriod ap
where acc.accountNo == cf.accountNo and
    cf.type == DEBIT
    cf.date >= ap.start and
    cf.date <= ap.end

trigger : acc.balance -= cf.amount

CashFlow

<table>
<thead>
<tr>
<th>date</th>
<th>amount</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-Feb-07</td>
<td>200</td>
<td>CREDIT</td>
</tr>
</tbody>
</table>

Account

<table>
<thead>
<tr>
<th>accountNo</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>150</td>
</tr>
</tbody>
</table>
What is a Rule

Quotes on Rule names are optional if the rule name has no spaces.

```
" rule <name>
  <attribute> <value>
when
  <LHS>
then
  <RHS>
end
```

- salience <int>
- agenda-group <string>
- no-loop <boolean>
- auto-focus <boolean>
- duration <long>

RHS can be any valid java. Future versions will support other languages, i.e Groovy
What is a Rule

```
public void helloMark(Person person) {
    if (person.getName().equals(mark)) {
        System.out.println(Hello Mark);
    }
}
```

Rules can never be called directly

```
"rule Hello Mark
when
    Person( name == mark )
then
    System.out.println(Hello Mark);
end
```

Methods that must be called directly

Specific instances cannot be passed.

LHS

RHS

specific passing of instances
Shower( temperature == "hot" )
Our First Rule

rule "increase balance for AccountPeriod Credits"
when
    ap : AccountPeriod()
    acc : Account( $accountNo : accountNo )
    CashFlow( type == CREDIT,
        accountNo == $accountNo,
        date >= ap.start && <= ap.end,
        $ammount : ammount
    )
then
    acc.balance  += $ammount;
end

select * from Account acc,
    Cashflow cf, AccountPeriod ap
where acc.accountNo == cf.accountNo and
    cf.type == CREDIT
    cf.date >= ap.start and
    cf.date <= ap.end

Pattern
Pattern Binding
field Binding
Literal Restriction
Variable Restriction
Multri Restriction - Variable Restriction
field Binding
Consequence (RHS)
### Rules as a “view”

<table>
<thead>
<tr>
<th>date</th>
<th>amount</th>
<th>type</th>
<th>accountNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-Jan-07</td>
<td>100</td>
<td>CREDIT</td>
<td>1</td>
</tr>
<tr>
<td>2-Feb-07</td>
<td>200</td>
<td>DEBIT</td>
<td>1</td>
</tr>
<tr>
<td>18-May-07</td>
<td>50</td>
<td>CREDIT</td>
<td>1</td>
</tr>
<tr>
<td>9-Mar-07</td>
<td>75</td>
<td>CREDIT</td>
<td>1</td>
</tr>
</tbody>
</table>

#### AccountingPeriod

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-Jan-07</td>
<td>31-Mar-07</td>
</tr>
</tbody>
</table>

#### Account

<table>
<thead>
<tr>
<th>accountNo</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

rule “increase balance for AccountPeriod Credits”
when
- ap : AccountPeriod()
- acc : Account( $accountNo : accountNo )

CashFlow( type == CREDIT,
    accountNo == $accountNo,
    date >= ap.start && <= ap.end,
    $ammount : ammount )

then
- acc.balance += $amount;
end

rule “decrease balance for AccountPeriod Debits”
when
- ap : AccountPeriod()
- acc : Account( $accountNo : accountNo )

CashFlow( type == DEBIT,
    accountNo == $accountNo,
    date >= ap.start && <= ap.end,
    $ammount : ammount )

then
- acc.balance -= $amount;
end

#### CashFlow

<table>
<thead>
<tr>
<th>date</th>
<th>amount</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-Jan-07</td>
<td>100</td>
<td>CREDIT</td>
</tr>
<tr>
<td>18-May-07</td>
<td>50</td>
<td>CREDIT</td>
</tr>
</tbody>
</table>

#### Account

<table>
<thead>
<tr>
<th>accountNo</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-50</td>
</tr>
</tbody>
</table>
### Rules as a “view”

<table>
<thead>
<tr>
<th>date</th>
<th>amount</th>
<th>type</th>
<th>accountNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-Jan-07</td>
<td>100</td>
<td>CREDIT</td>
<td>1</td>
</tr>
<tr>
<td>2-Feb-07</td>
<td>200</td>
<td>DEBIT</td>
<td>1</td>
</tr>
<tr>
<td>18-May-07</td>
<td>50</td>
<td>CREDIT</td>
<td>1</td>
</tr>
<tr>
<td>9-Mar-07</td>
<td>75</td>
<td>CREDIT</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-Apr-07</td>
<td>30-Jun-07</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>accountNo</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Rule: “Increase balance for AccountPeriod Credits”

```latex
\text{rule \textquote{increase balance for AccountPeriod Credits}}
\text{when}
\begin{align*}
\text{ap} &: \text{AccountPeriod()}, \\
\text{acc} &: \text{Account( accountNo : accountNo )}
\end{align*}
\text{CashFlow( type == CREDIT, accountNo == accountNo, date \geq ap.start \&\& \leq ap.end, amount : amount )}
\text{then}
\text{acc.balance += amount;}
\text{end}
```

#### Rule: “Decrease balance for AccountPeriod Debits”

```latex
\text{rule \textquote{decrease balance for AccountPeriod Debits}}
\text{when}
\begin{align*}
\text{ap} &: \text{AccountPeriod()}, \\
\text{acc} &: \text{Account( accountNo : accountNo )}
\end{align*}
\text{CashFlow( type == DEBIT, accountNo == accountNo, date \geq ap.start \&\& \leq ap.end, amount : amount )}
\text{then}
\text{acc.balance -= amount;}
\text{end}
```
What is a Production Rule System

- Codification of the business knowledge
- Repository of asserted Java instances
- Production Memory (rules)
- Working Memory (facts)
- Inference Engine
  - Pattern Matcher
  - Agenda
  - Insert, update, retract
Production Rule System
Approximated by SQL and Views

Tables
- Account
- AccountingPeriod
- Cashflow

Views
- view 1
- view 2

View
- main view

Object Types
- Account
- AccountingPeriod
- Cashflow

Rules
- rule1
- rule2

agenda
Conflict Resolution with Salience

rule “Print balance for AccountPeriod”
   salience -50
   when
     ap : AccountPeriod()
     acc : Account( )
   then
     System.out.println( acc.accountNo + " : " acc.balance );
   end

Agenda

<table>
<thead>
<tr>
<th></th>
<th>increase balance</th>
<th>arbitrary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>increase balance</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>decrease balance</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>increase balance</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>print balance</td>
<td></td>
</tr>
</tbody>
</table>
rule “increase balance for AccountPeriod Credits”
ruleflow-group “calculation”
when
    ap : AccountPeriod()
    acc : Account( $accountNo : accountNo )
    CashFlow( type == CREDIT,
               accountNo == $accountNo,
               date >= ap.start && <= ap.end,
               $ammount : ammount )
then
    acc.balance  += $amount;
end

rule “Print blance for AccountPeriod”
ruleflow-group “report”
when
    ap : AccountPeriod()
    acc : Account( )
then
    System.out.println( acc.accountNo + " : " acc.balance );
end
not Bus( color == "red" )

rule "increase balance for AccountPeriod Credits"
when
  ap : AccountingPeriod( )
  not AccountingPeriod( start < ap.start)
  acc : Account( $accountNo : accountNo )
  CashFlow( type == CREDIT,
             accountNo == $accountNo,
             date >= ap.start && <= ap.end,
             $ammount : ammount )
then
  acc.balance += $ammount;
end

'not', 'exists', 'forall'
rule "Test 01 - Credit Cashflow"
  salience 100
  when
    UpdatingAccount( $account : account )
    CurrentAccountingPeriod( $start : start, $end : end )
    Number( $sum : DoubleValue )
    from accumulate( $c : Cashflow( type==Cashflow.CREDIT,
      account == $account,
      date >= $start && <= $end,
      $amount : amount )
      sum( $c.getAmount() ) )
  then
    System.out.println( "CREDIT : " + $end + " : " + $sum );
    $account.balance += $sum);
end
package com.sample

import java.util.Map
import com.sample.Cheese

global Cheese cheese

function void exampleFunction(Cheese cheese) {
    System.out.println( cheese );
}

rule A Cheesy Rule
  when
    &.
  then
    &.
  end

Namespace for all package members

Imports can be used in functions and rules. Uses valid java import syntax
Two Phase System

Determine possible rules to fire

Working Memory Action
- insert
- modify
- retract

Fire Rule

Agenda Evaluation
- Select Rule to Fire
- Rule Found
- No Rule Found

exit
Features

- Engine
  - Full Rete Implementation -- with high performance indexing
  - Dynamic RuleBases
  - Stateful and Stateless Execution Modes
  - Async operations
  - Rete and Sequential Rete
  - Rule Agent
  - Optional Data Shadowing
  - Pluggeable Dialects

- Propositional Logic
  - Literal Restriction
  - Variable Restriction
  - Return Value Restriction
  - Jointed and dis-jointed Connectives allowed - '&&' '||'
  - inline-Eval
Features

- First Order Logic (Quantifiers)
  - And
  - Or
  - Exists
  - Not
  - Accumulate
  - Collect
  - From
  - Forall
  - Nesting of any CE inside of 'and' and 'or'
  - Support for both infix and prefix 'and'/or' CEs
  - Nesting and Chaining of 'from', 'accumulate', 'collect'
Features

- Execution Control
  - Conflict Resolution (salience) Now pluggeable
  - Agenda Filters
  - Agenda Groups
  - Activation Groups
  - Rule Flow
  - Attributes (no-loop, lock-on-active)

- Temporal Rules
  - Scheduler for rule duration will fire when a rule is true for X duration

- Truth maintenance
  - Logical Insertions

- Event Model
  - Working Memory, Agenda, Rule Flow and Rule Base
Eclipse IDE

import org.drools.examples.State;

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

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

rule "B to C"
salience 10
when
  State(B), state == State.FINISHED |
  c = State(C), state == State.NOTRUN |
then
  System.out.println(c.getName() + " finished");
end

rule "C to D"
when
  State(C), state == State.FINISHED |
then
  System.out.println("C finished");
end

rule "D to E"
when
  State(D)
then
  System.out.println("D finished");
end

rule "E to F"
when
  State(E)
then
  System.out.println("E finished");
end

rule "F to G"
when
  State(F)
then
  System.out.println("F finished");
end

rule "G to H"
when
  State(G)
then
  System.out.println("G finished");
end

rule "H to I"
when
  State(H)
then
  System.out.println("H finished");
end

rule "I to J"
when
  State(I)
then
  System.out.println("I finished");
end

rule "J to K"
when
  State(J)
then
  System.out.println("J finished");
end

rule "K to L"
when
  State(K)
then
  System.out.println("K finished");
end

rule "L to M"
when
  State(L)
then
  System.out.println("L finished");
end

rule "M to N"
when
  State(M)
then
  System.out.println("M finished");
end

rule "N to O"
when
  State(N)
then
  System.out.println("N finished");
end

rule "O to P"
when
  State(O)
then
  System.out.println("O finished");
end

rule "P to Q"
when
  State(P)
then
  System.out.println("P finished");
end

rule "Q to R"
when
  State(Q)
then
  System.out.println("Q finished");
end

rule "R to S"
when
  State(R)
then
  System.out.println("R finished");
end

rule "S to T"
when
  State(S)
then
  System.out.println("S finished");
end

rule "T to U"
when
  State(T)
then
  System.out.println("T finished");
end

rule "U to V"
when
  State(U)
then
  System.out.println("U finished");
end

rule "V to W"
when
  State(V)
then
  System.out.println("V finished");
end

rule "W to X"
when
  State(W)
then
  System.out.println("W finished");
end

rule "X to Y"
when
  State(X)
then
  System.out.println("X finished");
end

rule "Y to Z"
when
  State(Y)
then
  System.out.println("Y finished");
end

rule "Z to A"
when
  State(Z)
then
  System.out.println("Z finished");
end

The selected working memory has no globals defined.
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"
salience 10
when
    State(name == "B", state == State.FINISHED )
    c : State(name == "C", state == State.NOTRUN )
then
    System.out.println(c.getName() + " finished");
The selected working memory has no globals defined.

- Object asserted (1): A[NORUN]
  - Activation created: Rule Bootstrap a=A[NORUN](1)
- Object asserted (2): b[NORUN]
- Object asserted (3): c[NORUN]
- Object asserted (4): d[NORUN]
  - Activation executed: Rule Bootstrap a=A[NORUN](1)
    - Object modified (1): A[FINISHED]
      - Activation created: Rule A to B b=B[NORUN](2)
  - Activation executed: Rule A to B b=B[NORUN](2)
    - Object modified (2): B[FINISHED]
      - Activation created: Rule B to C c=C[NORUN](3)
      - Activation created: Rule B to D d=D[NORUN](4)
  - Activation executed: Rule B to C c=C[NORUN](3)
    - Object modified (3): C[FINISHED]
  - Activation executed: Rule B to D d=D[NORUN](4)
    - Object modified (4): D[FINISHED]

- MAIN: AgendaItem (id=1259)
  - AgendaItem (id=1262)
    - ruleName = "b to c"
    - c = State (id=1269)
  - AgendaItem (id=1263)
    - ruleName = "b to d"
    - d = State (id=1270)

- [0]= State (id=1268)
  - FINISHED = 1
  - NOTRUN = 0
  - changes = Property/ChangeSupport (id=1294)
    - name = "c"
      - state = 0
- [1]= State (id=1270)
  - [2]= State (id=1271)
package org.drools.examples

import org.drools.examples.HelloWorldExample.Message;

rule "Hello World"
  dialect "eclipsel" 
  when
    m : Message( status == Message.Hello, message : message )
  then
    System.out.println( "Hello, message : " + message);
  modify | m | { message = "Goodbye cruel world", status = Message.GOODBYE };
end

rule "GoodBye"
  dialect "java"
  no-loop true
  when
    m : Message( status == Message.GOODBYE, message : message )
  then
    System.out.println( "message : " + message);
    m.setMessage( message );
  end
Guided Editor (Eclipse)
```reasonml
@rule "Driver in unsafe area for marginal age"
  when
    Policy type is 'COMPREHENSIVE'
    Driver is less than 25 years old
    Driver has a location risk profile of 'HIGH'
  then
    Driver has a location risk profile of '{risk}''
    Driver has an age of at least {age}
    Driver has had more than {prior} prior claims
    Driver has had {number} prior claims
    Driver is between {lower} and {upper} years old
    Driver is greater than {age} years old
    Driver is less than {age} years old
    Policy has not been rejected
    Policy type is '{type}'
  end
end

@rule "Driver unsafe for third party"
  when
    Policy type is 'THIRD_PARTY'
    Driver has had more than 2 prior claims
```
<table>
<thead>
<tr>
<th>Language Expression</th>
<th>Rule Language Mapping</th>
<th>Object</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>There is a customer ticket with status of &quot;{status}&quot;</td>
<td>customer : Customer()  ticket : Ticket()</td>
<td></td>
<td>[condition]</td>
</tr>
<tr>
<td>There is a &quot;{subscription}&quot; customer with a ticket status of &quot;{status}&quot;</td>
<td>customer : Customer(subscription =...)</td>
<td></td>
<td>[condition]</td>
</tr>
<tr>
<td>Log &quot;{message}&quot;</td>
<td>System.out.println(&quot;{message}&quot;);</td>
<td></td>
<td>[consequence]</td>
</tr>
<tr>
<td>Escalate the ticket</td>
<td>ticket.setStatus(&quot;Escalate&quot;); update...</td>
<td></td>
<td>[consequence]</td>
</tr>
<tr>
<td>Send escalation email</td>
<td>sendEscalationEmail(customer, ticket...)</td>
<td></td>
<td>[consequence]</td>
</tr>
</tbody>
</table>

**Expression:**

**Mapping:**

**Object:**

**Sort by:**
<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></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MED</td>
<td></td>
<td>FIRE_THEFT</td>
<td>200</td>
<td>Priors not relevant</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></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></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>19.24</td>
<td>HIGH</td>
<td></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>
Create a new top level category.

Category name: Fibonacci
Description: fibonacci category

Categories aid in managing large numbers of rules/assets. A shallow hierarchy is recommended.

Current categories:
- HR
- Finance
- Draft

Refresh view:

Create a new category:
Delete the currently selected category:
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:
rule "Rule_1"
  when
  Person( age <= 42, age > 21 )
  b : Board( )
  not Board( cost > 1200 )
  then
  b.setCost( 1200 );
Rule Flow

- Unifies Rules and Processes in a single engine
  - Rules (LHS When) and expressions can be used in splits, milestones etc
    - creates a much richer model
  - Rules and Processes see, reason and react and process the same data
    - Do not have send messages between two different engines
  - Multiple instances, of different processes, can be executing at the same time in a single engine.
- Processes and Rules interactive with each other.
  - A Process or Rule can change data, which can impact how another rule or process is executing.
- Integrated Tooling and APIs
  - Single api for execution
  - Audit logging and visual Audit tools
  - Single server side tooling for storage, configuration and management and deployment
Ruleflow features

- Rule set nodes

- Control flow
  - Sequence
  - Parallelism (split / join)
  - Choice

- Nodes
  - Actions
  - Milestone (= state)
  - Subflows
  - Looping
Rule Flow - Rules and Processes
Rule Flow - Split Constraint Editor

Constraint editor

Name: More guesses
Priority: 1

Always true

Textual Editor

GameRules( allowed : allowedGuesses )
Game( guessCount < allowed )

and
eval
exists
forall
from
not
or
then
Game
<table>
<thead>
<tr>
<th>Name</th>
<th>Last modified</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insurance extra items percent</td>
<td>Sep 20, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Insurance Calculate</td>
<td>Sep 20, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Driver is underage</td>
<td>Sep 20, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>New licenced Driver</td>
<td>Sep 20, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Driver Single Young Male Driver factor</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Driver Mature Married With Young Child factor</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Priory Claimed Driver</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Day Vehicle Place</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Night Vehicle Place</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Driver wants an extra Car</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Driver wants glass coverage</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Driver wants non related expenses coverage</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>insuranceProcess</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>approve</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>rejection</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
</tbody>
</table>
What's coming in Q1?

- **Engine**
  - Stateful High Availability

- **Event Stream Processing, Complex Event Processing**
  - Time windows (fixed, since, until)
  - Date comparisons between objects (before, same, after)

- **RuleFlow**
  - Persistence
  - Timers
  - More complex workflow patterns
  - Pluggeable tasks

- **BRMS**
  - UI improvements
  - ACL Security
  - Scenario Testing
  - Decision Tables
RuleFlow Pluggeable Tasks
BRMS UI Improvements
Dave Bowman: All right, HAL; I'll go in through the emergency airlock.

HAL: Without your space helmet, Dave, you're going to find that rather difficult.

Dave Bowman: HAL, I won't argue with you anymore! Open the doors!

HAL: Dave, this conversation can serve no purpose anymore. Goodbye.

Joshua: Greetings, Professor Falken.
Stephen Falken: Hello, Joshua.
Joshua: A strange game. The only winning move is not to play. How about a nice game of chess?