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.
Agenda

- Introduction
- Rules
- Quick Drool Overview
- Processes
- Complex Event Processing
- Community Projects
- Road Map
Introduction
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

<table>
<thead>
<tr>
<th>Open Source</th>
<th>Drools has a business friendly open source license that makes it free to download, use, embed, and distribute.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declarative Programming</td>
<td>Allow you to say “What to do” not “How to do it”.</td>
</tr>
<tr>
<td>Accessible business rules</td>
<td>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 GUls for managing rule assets.</td>
</tr>
<tr>
<td>Logic and Data Separation</td>
<td>Rule engines are the ultimate in logic and data de-coupling. Increases rule reliability, reduces development time and testing.</td>
</tr>
</tbody>
</table>
Tuesday, November 06, 2007

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.
Rules
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>

increase balance for AccountPeriod Credits

```sql
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

```sql
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>

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

Account

<table>
<thead>
<tr>
<th>accountNo</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>150</td>
</tr>
</tbody>
</table>

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>
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”);
  }
}

- Rule “Hello Mark”
  when
  Person( name == “mark” )
  then
    System.out.println(“Hello Mark”);
  end

Rules can never be called directly

Specific instances cannot be passed.

Methods that must be called directly

Specific passing of instances
**What is a Pattern**

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

Literal Restriction

field Binding

Variable Restriction

Multri Restriction - Variable Restriction

Consequence (RHS)
<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
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>AccountingPeriod</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
</tr>
<tr>
<td>01-Apr-07</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountNo</td>
</tr>
<tr>
<td>1</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
What is a Production Rule System

- Production Memory (rules)
- Working Memory (facts)

Inference Engine

Pattern Matcher

Agenda

Repository of asserted Java instances

Codification of the business knowledge

.insert
.update
.retract
Production Rule System Approximated by SQL and Views

Tables
- Account
- AccountingPeriod
- Cashflow

Views
- view1
- view2

View
- main view

Object Types
- Account
- AccountingPeriod
- Cashflow

Rules
- rule1
- rule2

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

<table>
<thead>
<tr>
<th>Agenda</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>increase balance</td>
<td>arbitrary</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 += $ammount;
end

ruleflow-group

rule “Print balance for AccountPeriod”
ruleflow-group “report”
when
  ap : AccountPeriod()
  acc : Account( )
then
  System.out.println( acc.accountNo + “ : “ acc.balance );
end
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 += $amount;
end

not Bus( color == "red" )
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
Two Phase System

Working Memory Action
- insert
- modify
- retract

Determine possible rules to fire

Fire Rule

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

exit
Quick Drools Overview
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-jointen 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; 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
```

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 );

derule "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");

derule "B to C"

   when
       State(name == "B", state == State.FINISHED)
       c : State(name == "C", state == State.NOTRUN)

   then
       System.out.println(c.getName() + " finished");
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( 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 );
    m.setMessage( message );
end
Guided Editor (Eclipse)
DSLs (Eclipse)

```
@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

@rule "Driver unsafe for third party"
    when
        Policy type is 'THIRD_PARTY'
        Driver has had more than 2 prior claims
    then
        Reject Policy with explanation: 'Driver in that area is too risky -'
    end
```
<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></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></td>
</tr>
<tr>
<td>Log &quot;{message}&quot;</td>
<td>System.out.println(&quot;{message}&quot;);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Escalate the ticket</td>
<td>ticket.setStatus(&quot;Escalate&quot;); update...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Send escalation email</td>
<td>sendEscalationEmail(customer, ticket);</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Expression: 

Mapping: 

Object: 

Sort by: 

Edit
Remove
Add
Sort

Copy

DS Ls (Eclipse)
<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>18.24</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>0</td>
<td></td>
<td>COMPREHENSIVE</td>
<td>120</td>
<td>Cheapest possible</td>
</tr>
<tr>
<td></td>
<td>25.30</td>
<td>1</td>
<td></td>
<td>COMPREHENSIVE</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td></td>
<td>25.30</td>
<td>2</td>
<td></td>
<td>COMPREHENSIVE</td>
<td>590</td>
<td></td>
</tr>
<tr>
<td></td>
<td>25.35</td>
<td>3</td>
<td></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 );
Processes
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
Architectural Overview

Pluggeable Nodes
Pluggeable Work Items
Semantic Module Framework
Pluggable XML representations for DSLs
Rules and Process Overlap

- Process can work independent of rules
- Rules (including cep) can work independent of processes
- Process can control rule execution
  - ruleflow-groups
- Rules (including cep) can control processes
  - Splits
  - Event wait states (listeners)
    - start node
  - Timers
Ruleflow features

- Rule set nodes
- Control flow
  - Sequence
  - Parallelism (split / join)
  - Choice
- Nodes
  - Actions
  - Milestone (= state)
  - Subflows
  - Looping
Rule Flow - Rules and Processes
Split Constraint
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
Different Dialect Actions

- **Start**
- **MVELAction**
- **JavaAction**
- **End**

Dialect: `java`  
Imports...  
Globals...

Textual `java`

```java
list.add("java was here");
```
WorkItem With Custom Editor
Work Item With Properties Editor
// Email
TaskDefinitionExtensionImpl emailDefinition = new TaskDefinitionExtensionImpl();
emailDefinition.setName("Email");
emailDefinition.addParameter(new ParameterDefinitionImpl("From", new StringDataType()));
emailDefinition.addParameter(new ParameterDefinitionImpl("To", new StringDataType()));
emailDefinition.addParameter(new ParameterDefinitionImpl("Subject", new StringDataType()));
emailDefinition.addParameter(new ParameterDefinitionImpl("Text", new StringDataType()));
emailDefinition.setDisplayName("Email");
emailDefinition.setIcon("icons/import_statement.gif");
addTaskDefinition(emailDefinition);

// Log
TaskDefinitionExtensionImpl logDefinition = new TaskDefinitionExtensionImpl();
logDefinition.setName("Log");
logDefinition.addParameter(new ParameterDefinitionImpl("Message", new StringDataType()));
logDefinition.setDisplayName("Log");
logDefinition.setIcon("icons/open.gif");
addTaskDefinition(logDefinition);
<process xmlns="http://drools.org/drools-4.0/process"
    xmlns:mydsl="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"
    name="process name" id="process name" package-name="org.domain" >

    <nodes>
        <start name="start node" />

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

        <mydsl:logger name="test logger" type="warn">
            This is my message
        </mydsl:logger>

        <end name="end node" />
    </nodes>

    <connections>
        <connection from="start node" to="action node" />
        <connection from="action node" to="test logger" />
        <connection from="test logger" to="end node" />
    </connections>

</process>
SemanticModule module = new DefaultSemanticModule("http://domain.org/mydsl");
module.addHandler("logger", new LoggerHandler());
public Object start(String uri, Attributes attrs, ExtensibleXmlParser xmlPackageReader) throws SAXException {
    xmlPackageReader.startConfiguration( localName, attrs );
    RuleFlowProcessImpl process = (RuleFlowProcessImpl) xmlPackageReader.getParent();
    ActionNodeImpl actionNode = new ActionNodeImpl();
    actionNode.setName( attrs.getValue( "name" ) );
    process.addNode( actionNode );
    ((ProcessBuildData)xmlPackageReader.getData()).addNode( actionNode );
    return actionNode;
}
public Object end(String uri, String localName,
                     ExtensibleXmlParser xmlPackageReader) throws SAXException {
    Configuration config = xmlPackageReader.endConfiguration();
    RuleFlowProcessImpl process = (RuleFlowProcessImpl) xmlPackageReader.getParent();
    ActionNodeImpl actionNode = (ActionNodeImpl) xmlPackageReader.getCurrent();
    actionNode.setAction("logger.Warn(" + config.getText() + ")");
    return actionNode;
}
Unified auditing

- Activation executed: Rule Start Clinical Pathway X if diagnosed d=Diagnose: Diagnose disease X: Type unknown
- Object removed (2): Diagnose: Diagnose disease X: Type unknown
  - Activation cancelled: Rule RuleFlow-org.drools.examples.cdss.ClinicalPathwayX-16-17
  - Activation cancelled: Rule Remove old diagnose d=Diagnose: Diagnose disease X: Type unknown
- RuleFlowGroup activated: Examinations[size=2]
- RuleFlow started: ClinicalPathwayX[org.drools.examples.cdss.ClinicalPathwayX]
- Activation executed: Rule Examination1
- Activation executed: Rule Examination2
- RuleFlowGroup deactivated: Examinations1[size=0]
- RuleFlowGroup activated: AdditionalExaminations[size=2]
- Object inserted (2): Diagnose: Diagnose disease X: Type unknown
  - Activation created: Rule Start Clinical Pathway X if diagnosed d=Diagnose: Diagnose disease X: Type unknown
  - Activation created: Rule RuleFlow-org.drools.examples.cdss.ClinicalPathwayX-16-17
  - Activation created: Rule Remove old diagnose d=Diagnose: Diagnose disease X: Type unknown
  - Activation created: Rule RuleFlow-org.drools.examples.cdss.ClinicalPathwayX-12
- Activation executed: Rule Remove old diagnose d=Diagnose: Diagnose disease X: Type unknown
  - Object removed (2): Diagnose: Diagnose disease X: Type unknown
    - Activation cancelled: Rule Start Clinical Pathway X if diagnosed d=Diagnose: Diagnose disease X: Type unknown
    - Activation cancelled: Rule RuleFlow-org.drools.examples.cdss.ClinicalPathwayX-16-17
- Activation executed: Rule Examination3
- RuleFlowGroup deactivated: AdditionalExaminations[size=0]
- RuleFlow completed: TreatmentY[org.drools.examples.cdss.TreatmentY]
- RuleFlow started: TreatmentY[org.drools.examples.cdss.TreatmentY]
- RuleFlow completed: ClinicalPathwayX[org.drools.examples.cdss.ClinicalPathwayX]
- Object inserted (2): Diagnose: Diagnose disease X: Type 2
### Technical rule assets

<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>
Complex Event Processing
Reason over time windows (ESP)
- fixed, time, count and relative
  - sum, average, max, min
- absence of events
- remove duplicate events

Reason over time differences on facts (CEP)
- detect invalid ordering of events
- detect specific sequence of events
- correlate out-of-order events
Clock Abstractions

- **System Clock**: a clock that periodically synchronizes with the machine clock.
- **Event based clock**: a clock that is updated every time a given event arrives
- **Pseudo Clock**: a generic implementation that is arbitrarily updated by application code
- **Event Attribute Clock**: a clock that is updated by a configured event attribute
Temporal Reasoning (CEP/ESP)

- Reasoning over absolute time windows
  - This is the common case of reasoning over a slide time-window and/or aggregation of values: "when the average transaction throughput calculated over 1 minute goes below the threshold, raise the alarm"

rule "absolute time window"
when
  $throughput : Number( doubleValue < threshold )
  from accumulate( Throughput( [time:win:(30days)]
    $current : current ),
    average( $current ) )
then
  // raise the alarm
end
Temporal Reasoning (CEP/ESP)

- Support separate time-windows per CE:
  rule "absolute time window per CE"
  when
  S tock( $id : id )
  Number( $monthAvg : doubleValue )
  from accumulate( DayClose tockPrice( [time:win:(30days)]
                          id == $id,
                          $value : value ),
                   average( $value ) )
  Number( intValue >= 3 )
  from accumulate( $o : SellOrder( [time:win:(30secs), distinct]
                                 completed == true,
                                 id == $id,
                                 value < $monthAvg )
                 count( $o ) )
Temporal Reasoning (CEP/ESP)

- Reasoning over relative time windows
  - this is a powerful feature of reasoning over variable time-windows defined in relation to other patterns. Security example: "since a user logs in, until the user logs off, when there is any privileged action for this user, allow it and log it to the audit log"

rule "relative time window"

when
  $evt : Event( user == $user, privileged == true )
  since
    LogInEvent( $user : user )
  until
    LogOffEvent( user == $user )
then
  // log event
  // allow action
end
Events have implicit time attributes and it must be possible to constraint events on its time attributes, using operators "after", "before", "between":

rule "time constraints between events"
when
    $order : StockBuyOrder( $id : id )
    StockBuyConfirmation( relatedEvent == id,
        this after [0,10] $order)
then
    // buy order confirmed
end
Temporal Reasoning (CEP/ESP)

- Support to reason over the absence of events:
  rule "absence of events"
when
    $temp : TemperatureReading( celciusGrade > $threshold )
    not SplinkerActivation( this after [0, 10] $temp )
then
  //raise the alarm
end
Community Projects
Uncertainty Systems to express truth degrees and reason over partial data - Davide Sottara (dsottara@deis.unibo.it)

- **Forms of Uncertainty (measures)**
  - **Probability**
    - Frequentist
    - Subjective (Bayesian)
  - **Possibility / Necessity**
    - Belief / Plausibility
    - Basic Mass Assignment Functions
    - Aggregator (sum)
    - Plausibility Functions
    - Belief Functions
Uncertainty Systems

- Traditional Pattern
  - Shower( temperature == "hot" )

- Pattern with uncertainty evaluator
  - Shower( temperature == ~"hot" )

- Pattern with uncertainty evaluator and parameters
  - Shower( temperature == ~(10, $x, 15, $y) "hot" )

Shower( temperature == "hot" )
Solves planning problems, such as:

- Employee shift rostering
- Freight routing
- Supply sorting
- Lesson scheduling
- Exam scheduling
- The traveling salesmen problem
- The traveling tournament problem
The n queens example

- Place n queens that cannot attach each other on a n*n chessboard
- One of the examples of drools-solver
- Implementation explained in detail in the reference manual
Different search algorithms

- Simple local search
- Tabu search
- Simulated annealing

You can easily switch the search algorithm, by simply changing the configuration.
Road Map
Road Map

- **Engine**
  - Stateful High Availability

- **Event Stream Processing, Complex Event Processing**
  - time windows (fixed, length, since, until)
  - date comparisons between objects (before, same, after)

- **RuleFlow will handle full Process definitions**
  - Persistence
  - Timers
  - Dialect support
  - better rule/code usage
  - More complex workflow patterns
  - Pluggable Nodes
  - Pluggable XML
  - Pluggable WorkItems
Road Map

- Rule Analysis
  - Verification
  - Impact Analysis
- BRMS
  - UI improvements
  - ACL Security
  - Scenario Testing
  - Decision Tables
- Agent based computing
- Drools Solver
- Uncertainty
  - Bayesian Network subsystems
- Decision Trees
- SBVR
**Questions?**

- **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?