Drools
BeJUG 2010
Kris Verlaenen
Geoffrey De Smet
Agenda

Introduction
Drools Expert: rule engine
Drools Guvnor: knowledge repository
Drools Planner: automated planning
Drools Flow: workflow engine
Drools Fusion: complex event processing
Summary
Drools Platform

Introduction
Drools History

Drools 2
  Rete like XML Scripting language

Drools 3
  Based on Clips functionality
  Iterative improves to JRules syntax with Clips functionality

Drools 4
  More declarative
  Basic Rule Flow
  Basic BRMS
Drools 5: BLiS

Drools Expert

Drools Flow

Drools Fusion

Drools Guvnor

Business Logic integration System
Motivation

A business solution usually involves the interaction between these technologies.

- Technology overlap
- Business overlap

Several (good) products on the market, better either at rule or business process or event processing

Attribute the same importance to the three complementary business modeling techniques
Drools Expert

Rule engine
Drools Expert agenda

Use case example
Initialization and usage
Rules
Use case example
Initialization and usage
Rules
Bob's phone call service
Network hardware/software works

Bob's problem
Calculate phone call invoice
Per customer
Calculation rules change regularly
Discounts, promotions, …

*If you call someone 30 minutes on Saturday and Sunday then you can call him/her 10 free minutes on Monday!*
Customers and subscribers

Bob's phone call service
  1 Network

Customer: 1 invoice
  Company A
  Company B

Subscriber
  Fred of A
  Greg of A

Phone call
  Call 1 of Fred
  Call 2 of Fred
Bob's problem

Calculate phone call invoice

Bob's solution

Hire 3 consultants

Java

SQL (MySQL, HSQLDB, ...)

JPA (Hibernate, TopLink, ...)

Rule engine (Drools, Jess, ...)

Java

SQL

Rule
Drools Expert agenda

Use case example

Initialization and usage

Rules
private DataSource dataSource;

private void initSql() {
    jdbcDataSource jdbcDataSource = new jdbcDataSource();
    jdbcDataSource.setDatabase("jdbc:hsqldb:mem:droolsphone");
    jdbcDataSource.setUser("sa");
    jdbcDataSource.setPassword("");
    this.dataSource = jdbcDataSource;
}
private KnowledgeBase knowledgeBase;

private void initDrools() {
    KnowledgeBuilder builder = KnowledgeBuilderFactory.newKnowledgeBuilder();
    builder.add(ResourceFactory.newClassPathResource("org/drools/examples/droolsphone/phoneRules1.drl"),
                 ResourceType.DRL);
    if (builder.hasErrors()) {
        throw new IllegalStateException(builder.getErrors().toString());
    }

    knowledgeBase = KnowledgeBaseFactory.newKnowledgeBase();
    knowledgeBase.addKnowledgePackages(builder.getKnowledgePackages());
}

Build a KnowledgeBase
Thread-safe singleton
Heavy weight
Using SQL

Build a Connection

Not thread-safe
Heavy weight (pooling needed)
1 Transaction per service call

Execute query's

```java
public void chargeWithSql() {
    Invoice invoice = new Invoice();
    try {
        Connection connection = dataSource.getConnection();
        PreparedStatement statement = connection.prepareStatement(...);
        ResultSet resultSet = statement.executeQuery();
        ...
    } catch (SQLException e) {
        throw new IllegalStateException("SQL example failed.", e);
    }
    System.out.println(invoice.toString());
}
```
Using SQL: fill database

Data not in database
  NoSQL, webservice, XML file, ...
Data not in the correct model
  Extra indexes, temp tables, ...

```java
public void fillDatabase(List<PhoneCall> phoneCallList) {
    try {
        Connection connection = jdbcDataSource.getConnection();
        connection.createStatement().executeUpdate("CREATE TABLE PhoneCall(id BIGINT PRIMARY KEY)");
        PreparedStatement phoneCallInsert = connection.prepareStatement("INSERT INTO PhoneCall(id) VALUES (?)");
        for (PhoneCall phoneCall : phoneCallList) {
            phoneCallInsert.clearParameters();
            phoneCallInsert.setLong(1, phoneCall.getId());
            phoneCallInsert.executeUpdate();
        }
    } catch (SQLException e) {...}
}
```
Using Drools (Stateless)

Build a KnowledgeSession
Not thread-safe
Light weight
1 stateless KnowledgeSession per service call

Execute on facts
Fact = POJO inserted into the session

```java
public void chargeWithDrools(List<PhoneCall> phoneCallList) {
    Invoice invoice = new Invoice();

    StatelessKnowledgeSession session = knowledgeBase.newStatelessKnowledgeSession();
    session.setGlobal("invoice", invoice);
    session.execute(phoneCallList); // Insert facts and fire all rules

    System.out.println(invoice.toString());
}
```
Using Drools (Stateful)

Stateful splits up:
- insert/update/remove facts
- Fire all rules on those facts
- Session is reusable across service calls

```java
public void chargeWithDrools(List<PhoneCall> phoneCallList) {
    Invoice invoice = new Invoice();

    StatefulKnowledgeSession session = knowledgeBase.newStatefulKnowledgeSession();
    session.setGlobal("invoice", invoice);
    for (PhoneCall p : phoneCallList) {
        session.insert(p);
    }
    session.fireAllRules();
    System.out.println(invoice.toString());
}
```
Use case example
Initialization and usage
Rules
One euro per phone call: java

Each phone call costs 1 euro

```java
Invoice invoice = new Invoice();
...

// Find PhoneCalls
for (PhoneCall p : phoneCallList) {

    // Charge PhoneCalls
    System.out.println("Charging phone call with Java.");
    invoice.addPrice(new BigDecimal("1.00"));
}
```
Invoice invoice = new Invoice();
...

// Find PhoneCalls
PreparedStatement statement = connection.prepareStatement("SELECT count(*) * 1.00" + " FROM PhoneCall");
ResultSet resultSet = statement.executeQuery();
resultSet.next();

// Charge PhoneCalls
System.out.println("Charging phone calls with SQL.");
BigDecimal total = resultSet.getBigDecimal(1);
invoice.setTotal(total);
Each phone call costs 1 euro

```java
global Invoice invoice;
...

rule "oneEuroPerPhoneCall"

  // Find PhoneCalls
  when // Left hand side (DRL pattern syntax)
    PhoneCall()

  // Charge PhoneCalls
  then // Right hand side (Java code)
    System.out.println("Charging phone call with Drools.");
    invoice.addPrice(new BigDecimal("1.00"));

end
```
One euro per phone call

```java
for (PhoneCall p : phoneCallList) {
    System.out.println(" Charging phone call with Java.");
    invoice.addPrice(new BigDecimal("1.00"));
}
```

```sql
PreparedStatement statement = connection.prepareStatement(
    "SELECT count(*) * 1.00"
+ " FROM PhoneCall");
ResultSet resultSet = statement.executeQuery();
resultSet.next();
System.out.println(" Charging phone calls with SQL.");
BigDecimal total = resultSet.getBigDecimal(1);
invoice.setTotal(total);
```

```drools
rule "oneEuroPerPhoneCall"
when // Left hand side (DRL pattern syntax)
    PhoneCall()
then // Right hand side (Java code)
    System.out.println(" Charging phone call with Drools.");
    invoice.addPrice(new BigDecimal("1.00"));
end
```
Pattern binding

Variables have a dollar sign

```
rule "oneEuroPerPhoneCall"

  when
    $p : PhoneCall()  // Pattern binding

  then
    System.out.println(" Charging phone call (" + $p.getId() + ") with Drools.");
    invoice.addPrice(new BigDecimal("1.00");

end
```
charged uses the getter
public boolean isCharged() {...}
== uses equals() method except for primitive types

rule "oneEuroPerPhoneCall"

when
  $p : PhoneCall(charged == false) // Literal restriction
then
  $p.setCharged(true);
  invoice.addPrice(new BigDecimal("1.00"));
end
Multiple restrictions

AND logic

```java
rule "morningPhoneCall"
    when
        $p : PhoneCall(charged == false, startDateTime.hourOfDay < 12)
    then
        modify ($p) {
            setBasePricePerSecond(new BigDecimal("0.05"));
        }
end

rule "afternoonPhoneCall"
    when
        $p : PhoneCall(charged == false, startDateTime.hourOfDay >= 12)
    then
        modify ($p) {
            setBasePricePerSecond(new BigDecimal("0.20"));
        }
end
```

PhoneCall

- id : long
- startDateTime : LocalDateTime
- durationPeriod : Period
- basePricePerSecond : BigDecimal
- charged : boolean
for (PhoneCall p : phoneCallList) {
    if (!p.isCharged()) {
        if (p.getStartDateTime().getHourOfDay() < 12) {
            p.setBasePricePerSecond(new BigDecimal("0.05"));
        } else {...}
    }
}

PreparedStatement statement = connection.prepareStatement(
    "SELECT SUM(0.05 * durationPeriod)"
    + "  FROM PhoneCall"
    + "  WHERE charged = 0"
    + "  AND EXTRACT (HOUR FROM startDateTime) < 12");

rule "morningPhoneCall"
when
    $p : PhoneCall(charged == false, startDateTime.hourOfDay < 12)
then
    modify ($p) {
        setBasePricePerSecond(new BigDecimal("0.05"));
    }
end
for (PhoneCall p : phoneCallList) {
    if (!p.isCharged() && p.getBasePricePerSecond() != null) {
        BigDecimal price = p.getBasePricePerSecond().multiply(
            BigDecimal.valueOf(p.getDurationPeriodInSeconds()));
        invoice.addPrice(price);
        p.setCharged(true);
    }
}

rule "chargePhoneCalls"
when
    $p : PhoneCall(charged == false, basePricePerSecond != null)
then
    BigDecimal price = $p.getBasePricePerSecond().multiply(
        BigDecimal.valueOf($p.getDurationPeriodInSeconds()));
    invoice.addPrice(price);
    modify ($p) {
        setCharged(true);
    }
end
DRL versus Java

DRL is declarative
Java is imperative

Find a pattern in a String
Use a regular expression (declarative)
"\d+\.(\d+)\)*"
Process the result with Java

Find a pattern in a Object heap
Use a rule engine (declarative)
PhoneCall(charged == false,
            startDateTime.hourOfDay >= 12)
Process the result with Java
for (PhoneCall p : phoneCallList) {
    if (!p.isCharged()) {
        if (HolidayUtil.isHoliday(p.getStartDate())) {
            p.setBasePricePerSecond(new BigDecimal("0.07"));
        } else {...}
    }
}

// SQL has no isHoliday function

rule "holidayCall"
when
    $p : PhoneCall(charged == false,
        eval(HolidayUtil.isHoliday(startDate)))
then
    modify ($p) {
        setBasePricePerSecond(new BigDecimal("0.07"));
    }
end
rule "gold basePricePerSecond" when
   $s : Subscriber(subscription == Subscription.GOLD)
   $p : PhoneCall(subscriber == $s)
then ... setBasePricePerSecond 0.05 ... end

rule "friends basePricePerSecond good days" when
   $s : Subscriber(subscription == Subscription.FRIENDS)
   $p : PhoneCall(subscriber == $s,
       startDateDateTime.dayOfWeek == DateTimeConstants.SATURDAY ||
       startDateDateTime.dayOfWeek == DateTimeConstants.SUNDAY)
then ... setBasePricePerSecond 0.01 ... end

rule "work basePricePerSecond good hours" when
   $s : Subscriber(subscription == Subscription.WORK)
   $p : PhoneCall(subscriber == $s,
       startDateDateTime.hourOfDay >= 9 && < 17)
then ... setBasePricePerSecond 0.02 ... end
Promotions are a problem

Promotion: discount percentage
For subscription
Between beginDate and endDate

No association!
No foreign key!
for (PhoneCall p : phoneCallList) {
    LocalDate startDate = p.getStartDate();
    Promotion promo = null;
    // TODO Does not scale, look up in Map instead!
    for (Promotion possiblePromo : promotionList) {
        if (p...getSubscription().equals(possiblePromo.getSubscription())
            && possiblePromo.getBeginDate().compareTo(startDate) <= 0
            && possiblePromo.getEndDate().compareTo(startDate) >= 0) {
            promo = possiblePromo; break;
        }
    }
    if (promo != null) {
        // Apply discount of promotions
        p.setDiscountPercentage(promo.getDiscountPercentage());
    } else {...}
Promotions: rules

**Rule** "Apply discount of promotions"
when
$p : \text{PhoneCall}(\$subscription : \text{subscriber.subscription},\n\$startDate : \text{startDate})$
$\text{pr} : \text{Promotion}(\text{subscription} == \$subscription,\n\text{beginDate} <= \$startDate, \text{endDate} >= \$startDate)"
then ... setDiscountPercentage $\text{pr}\text{.getDiscountPercentage()} ... end

**Rule** "Apply discount without promotion"
when
$p : \text{PhoneCall}(\$subscription : \text{subscriber.subscription},\n\$startDate : \text{startDate})$
not \text{Promotion}(\text{subscription} == \$subscription,\n\text{beginDate} <= \$startDate, \text{endDate} >= \$startDate) 
then ... setDiscountPercentage 0.00 ... end
DRL is powerful

Conditional elements
   not, exists, forall
from
   collect
   accumulate
      sum, max, … (and custom functions)
hibernate query

rule "Find People for given zip code"
when
   $zipCode : ZipCode()
Person() from $hibernate.getNamedQuery("Find People")
   .setParameters( [ "zipCode" : $zipCode ] )
   .list()
Charge only longest call per hour

salience

Rules with higher salience are run first
A rule can stop another rule from firing

rule "only longest call per hour is charged"
  salience 10 // Do this rule first
  when
    $p : PhoneCall(basePricePerSecond == null,
    $id : id, $s : subscriber,
    $startDate : startDate, $hourOfDay : startDateTime.hourOfDay,
    $durationPeriodInSeconds : durationPeriodInSeconds)
  exists PhoneCall(id != $id, subscriber == $s,
    startDate == $startDate,
    startDateTime.hourOfDay == $hourOfDay,
    durationPeriodInSeconds : durationPeriodInSeconds > $durationPeriodInSeconds)
  then
    modify ($p) {
      setBasePricePerSecond(new BigDecimal("0.00"));
    }
  end

Features

- Wizards
- Drools runtimes (classpath)
- Textual Rule Editor
  - Syntax highlighting, code completion, error detection, outline view, etc.
- Guided Editor
- Debugging
  - Rule breakpoints
  - Debug views
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}"
Guided rule editor

**WHEN**

Person [p]

- age must be less than 18

**THEN**

Modify [p] name

**options**

salience 10
import org.drools.examples.State;

rule Bootstrap
    when
        a : State(name == "A", state == State.NOTRUN) 
    then
        System.out.println(a.getName() + " 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");
Drools Guvnor

Knowledge Repository
Guvnor

- Technology
  - JCR (JSR-170) backend
  - Seam + GWT frontend
  - WebDav
  - Eclipse synchronisation plugin
  - Role based security

- Authoring
  - Decision Tables
  - Guided Editor

- QA
  - Scenario Testing + Rule Verification
# Business asset storage

![Business asset storage](image)

<table>
<thead>
<tr>
<th>Name</th>
<th>Last modified</th>
<th>Status</th>
<th>Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Underage</td>
<td>Dec 19, 2008</td>
<td>Draft</td>
<td>Eligibility rules</td>
</tr>
<tr>
<td>Bankruptcy history</td>
<td>Oct 1, 2008</td>
<td>Draft</td>
<td>Eligibility rules</td>
</tr>
<tr>
<td>No bad credit checks</td>
<td>Oct 1, 2008</td>
<td>Draft</td>
<td>Eligibility rules</td>
</tr>
<tr>
<td>no NINJAs</td>
<td>Oct 2, 2008</td>
<td>Draft</td>
<td>Eligibility rules</td>
</tr>
<tr>
<td>Pricing loans</td>
<td>Jan 27, 2009</td>
<td>Draft</td>
<td>Pricing rules</td>
</tr>
<tr>
<td>CreditApproval</td>
<td>Oct 22, 2008</td>
<td>Draft</td>
<td>Eligibility rules</td>
</tr>
<tr>
<td>DateDslRule</td>
<td>Oct 24, 2008</td>
<td>Draft</td>
<td>Technical</td>
</tr>
<tr>
<td>CheckBoxDslRule</td>
<td>Oct 23, 2008</td>
<td>Draft</td>
<td>Technical</td>
</tr>
<tr>
<td>RegexDslRule</td>
<td>Oct 23, 2008</td>
<td>Draft</td>
<td>Technical</td>
</tr>
<tr>
<td>wee</td>
<td>Jan 27, 2009</td>
<td>Draft</td>
<td>Home Mortgage</td>
</tr>
</tbody>
</table>
Guided Editor

WHEN

LoanApplication [application]

Applicant
age is less than 21

THEN

Set value of application approved false

Retract application

(options)

Attributes:
salience 10
## Decision Tables

### Advertiser type: Agency (3 Items)

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Advertiser type</th>
<th>age is at least</th>
<th>Postcode gre than</th>
<th>Postcode les than</th>
<th>Set the value</th>
<th>Set the reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Good suburbs</td>
<td>Agency</td>
<td>10</td>
<td>4000</td>
<td>4100</td>
<td>→ 42</td>
<td>Loyal</td>
</tr>
<tr>
<td>2</td>
<td>Good suburbs</td>
<td>Agency</td>
<td>2000</td>
<td>2100</td>
<td>2100</td>
<td>→ 43</td>
<td>Good region</td>
</tr>
<tr>
<td>4</td>
<td>Good suburbs</td>
<td>Agency</td>
<td>2200</td>
<td>2300</td>
<td>2300</td>
<td>→ 43</td>
<td>Good region</td>
</tr>
</tbody>
</table>

### Advertiser type: Partner (1 Item)

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Advertiser type</th>
<th>age is at least</th>
<th>Postcode gre than</th>
<th>Postcode les than</th>
<th>Set the value</th>
<th>Set the reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Partners</td>
<td>Partner</td>
<td>1</td>
<td></td>
<td></td>
<td>→ 49</td>
<td>Other</td>
</tr>
</tbody>
</table>
Scenario Testing
Drools Planner

Automated planning
Drools Planner agenda

Bin packaging
  What is NP complete?

Employee shift rostering
  AKA Nurse rostering
  Hard constraints
  Soft constraints

Patient admission schedule
  AKA Hospital bed planning
  How many possible solutions?

Other use cases
Bin packaging

Place each item on a location in a container.

Largest size first

\[
\begin{align*}
1 \times 3 &= 3 \\
2 \times 3 &= 6 \\
2 \times 4 &= 8 \\
3 \times 3 &= 9 \\
1 \times 5 &= 5
\end{align*}
\]

Largest side first

\[
\begin{align*}
1 \times 3 &= 3 \\
1 \times 3 &= 3 \\
2 \times 4 &= 8 \\
1 \times 5 &= 5 \\
2 \times 4 &= 8
\end{align*}
\]

Drools Planner

\[
\begin{align*}
1 \times 3 &= 3 \\
2 \times 3 &= 6 \\
3 \times 3 &= 9 \\
1 \times 5 &= 5
\end{align*}
\]
Bin packaging is NP complete

When do we put $2 \times 4 = 8$ into the container?

Second, of course!

First, of course!

Last, of course!

A given solution can be verified fast.
There is no efficient way to find a solution
Bin packaging is NP complete

$3 \times 6 = 18$

This container of size 18 cannot hold these 2 items with a total size of 14.

$3 \times 3 = 9$

$1 \times 5 = 5$

There is no easy way to verify if there is even a feasible solution.
Employee shift rostering

Populate each work shift with a nurse.

Maternity nurses
A Ann  B Beth  C Cory

Emergency nurses
D Dan  E Elin  G Greg

Basic nurses
H Hue  I Ilse

Largest staff first

Maternity nurses

<table>
<thead>
<tr>
<th></th>
<th>Sat</th>
<th>Sun</th>
<th>Mon</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>12</td>
<td>11</td>
<td>21</td>
</tr>
<tr>
<td>14</td>
<td>C</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>22</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

Emergency nurses

<table>
<thead>
<tr>
<th></th>
<th>Sat</th>
<th>Sun</th>
<th>Mon</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>D</td>
<td>G</td>
<td>D</td>
</tr>
<tr>
<td>22</td>
<td>E</td>
<td></td>
<td>E</td>
</tr>
</tbody>
</table>

Any nurses

<table>
<thead>
<tr>
<th></th>
<th>Sat</th>
<th>Sun</th>
<th>Mon</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>I</td>
<td></td>
<td>I</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>I</td>
<td>I</td>
</tr>
</tbody>
</table>

Drools Planner

Maternity nurses

<table>
<thead>
<tr>
<th></th>
<th>Sat</th>
<th>Sun</th>
<th>Mon</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>12</td>
<td>11</td>
<td>21</td>
</tr>
<tr>
<td>14</td>
<td>C</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>22</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

Emergency nurses

<table>
<thead>
<tr>
<th></th>
<th>Sat</th>
<th>Sun</th>
<th>Mon</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>D</td>
<td>G</td>
<td>D</td>
</tr>
<tr>
<td>22</td>
<td>E</td>
<td></td>
<td>E</td>
</tr>
</tbody>
</table>

Any nurses

<table>
<thead>
<tr>
<th></th>
<th>Sat</th>
<th>Sun</th>
<th>Mon</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>I</td>
<td></td>
<td>I</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>I</td>
<td>I</td>
</tr>
</tbody>
</table>
Employee shift rostering

Hard constraints

All required shifts must be assigned

Only one shift per day per employee

No hard constraint broken => solution is feasible
// a nurse can only work one shift per day
rule "oneShiftPerDay"
  when
    $left : EmployeeAssignment(
    $employee : employee,
    $shiftDate : shiftDate,
    $leftId : id
  );
    $right : EmployeeAssignment(
    employee == $employee,
    shiftDate == $shiftDate,
    id > $leftId);
  then
    // Lower the hard score with a weight ...
end
Employee shift rostering

Soft constraints

Maximum consecutive working days for Ann: 5

Minimum consecutive free days for Beth: 2

Day off wish for Carla: Sunday

After a night shift sequence: 2 free days

Unwanted pattern: E-L-E

There are many more soft constraints...
rule "dayOffRequest"
  when
    $dayOffRequest : DayOffRequest(
      $employee : employee,
      $shiftDate : shiftDate,
      $weight : weight
    );
    $employeeAssignment : EmployeeAssignment(
      employee == $employee,
      shiftDate == $shiftDate
    );
  then
    // Lower the soft score with the weight $weight ...
end
Patient admission schedule

Assign each patient a hospital bed.

Largest admission first

<table>
<thead>
<tr>
<th>General ward</th>
<th>November</th>
</tr>
</thead>
<tbody>
<tr>
<td>Room 11 bed 1</td>
<td>D 1-3</td>
</tr>
<tr>
<td></td>
<td>C 4-7</td>
</tr>
<tr>
<td>Room 11 bed 2</td>
<td>H 1-2</td>
</tr>
<tr>
<td></td>
<td>E 2-4</td>
</tr>
<tr>
<td></td>
<td>G 5-7</td>
</tr>
</tbody>
</table>

Drools Planner

<table>
<thead>
<tr>
<th>November</th>
</tr>
</thead>
<tbody>
<tr>
<td>H 1-2</td>
</tr>
<tr>
<td>E 2-4</td>
</tr>
<tr>
<td>C 4-7</td>
</tr>
<tr>
<td>D 1-3</td>
</tr>
<tr>
<td>F 3-5</td>
</tr>
<tr>
<td>G 5-7</td>
</tr>
</tbody>
</table>

Intensive care

| Room 21 bed 1 | A 1-6   |
| Room 22 bed 1 | B 1-5   |

no space
Patient admission schedule

Hard constraints

- No 2 patients in same bed in same night
- Room gender limitation
- Department minimum or maximum age
- Patient requires specific room equipment(s)

Soft constraints

- Patient prefers maximum room size
- Department specialization
- Room specialization
- Patient prefers specific room equipment(s)
Needle in a haystack

How many possible solutions?

310 beds
in 105 rooms
in 4 departments
84 nights
2750 patients (admissions)

Numbers from a real dataset
Needle in a haystack

How many possible solutions?

310 beds

in 105 rooms

in 4 departments

84 nights

2750 patients (admissions)

> inhabitants of Gent (BE)?

250 000 inhabitants

Source: wikipedia
Needle in a haystack

How many possible solutions?

- 310 beds
- in 105 rooms
- in 4 departments
- 84 nights
- 2750 patients (admissions)

> humans?

- 7,000,000,000 humans

Source: NASA (wikipedia)
Needle in a haystack

How many possible solutions?

- 310 beds
- in 105 rooms
- in 4 departments
- 84 nights
- 2750 patients (admissions)

> minimum atoms in the observable universe?

$10^{80}$

Source: NASA and ESA (wikipedia)
Needle in a haystack

How many possible solutions?

- 310 beds
- in 105 rooms
- in 4 departments
- 84 nights
- 2750 patients (admissions)

> atoms in the universe if every atom is a universe of atoms?

\[(10^{80})^{80} = 10^{6400}\]

Source: NASA and ESA (wikipedia)
Needle in a haystack

How many possible solutions?

310 beds
in 105 rooms
in 4 departments
84 nights
2750 patients (admissions)
A little over $10^{6851}$
Do the math

1 patient
310 beds
310 ways to schedule 1 patient

2 patients
310 * 310 = 96 100

3 patients
310 * 310 * 310 = 29 791 000

2750 patients
310 * 310 * ... * 310
310^{2750}
= a little over 10^{6851}
A little over $10^{6851}$
A little over $10^{6851}$
A little over $10^{6851}$
The search space is big!

Compare with WWW size
22 020 000 000 pages

Each possible solution
2750 patients scheduled into 310 beds
Still need to calculate the score!
Calculate $10^9$ scores per ms

Impossible today!

31,579,200,000 ms in 1 year

$< 10^{11}$ ms in 1 year

$10^9 \times 10^{11}$ scores per year

$= 10^{20}$ scores per year

How many years? $10^{6851} / 10^{20}$

$= 10^{6831}$ years

CPU 1000 times faster

It becomes $10^{6828}$ years
A dose of reality

Find the optimal solution?
Of a real world planning problem?
Not in our lifetimes!
Who cares?
Beat the human planner(s): easy!
Spend less resources
Save more money
Save the environment
Make more people happy
Always room to improve our algorithms
Examination timetabling

Assign each exam a period and a room.

<table>
<thead>
<tr>
<th>A</th>
<th>Ann</th>
<th>History</th>
<th>Math</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Bobby</td>
<td>History</td>
<td>Math</td>
</tr>
<tr>
<td>C</td>
<td>Carla</td>
<td>History</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Dan</td>
<td>Math</td>
<td>Chem</td>
</tr>
<tr>
<td>E</td>
<td>Edna</td>
<td>Chem</td>
<td>Bio</td>
</tr>
<tr>
<td>F</td>
<td>Fred</td>
<td>Bio</td>
<td>Geo</td>
</tr>
<tr>
<td>G</td>
<td>Greg</td>
<td>Geo</td>
<td>Eng</td>
</tr>
</tbody>
</table>

Most students first

<table>
<thead>
<tr>
<th>Room X</th>
<th>Room Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 seats</td>
<td>3 seats</td>
</tr>
</tbody>
</table>

Drools Planner

<table>
<thead>
<tr>
<th>Room X</th>
<th>Room Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 seats</td>
<td>3 seats</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mon 09:00</th>
<th>History</th>
<th>Chem</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>Fri 09:00</td>
<td>Math</td>
<td>Bio</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>Fri 14:00</td>
<td>Geo</td>
<td>Eng</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>Fr</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

same day

same time
Traveling tournament
Schedule each match in a timeslot.

Drools Planner
Total distance: 8.276

Team distance: 2.127

Team distance: 2.011

Team distance: 2.011

Team distance: 2.127
A **workflow** is a process that describes the order in which a series of steps need to be executed, using a flow chart.
Advantages

- Transparency
- Automation
- Higher-level
- End-to-end process visibility and monitoring
- Increased agility (change process more easily)
- Lower cost of development
- Process analysis / continuous improvement
Drools Flow Palette

Predefined set of generic *node types*
- Start, end
- Gateway
- Script
- Sub-process
- Event
- ...

Components:
- Start Event
- End Event
- Rule Task
- Gateway [diverge]
- Gateway [converge]
- Reusable Sub-Process
- Script Task
- Timer Event
- Error Event
- Message Event
- User Task
- Embedded Sub-Process
- Multiple Instances

Select
Marquee
Sequence Flow
Characteristics

Standard-based
  BPMN 2.0 for process modeling
  WS-HT for human tasks

Advanced, declarative modeling

Extensible, open-source process engine
  Persistence, transactions, logging, etc.

Domain-specific processes

Integration and unification of processes, rules and event processing

Full life cycle support
// creating the knowledge base
KnowledgeBuilder builder
    = KnowledgeBuilderFactory.newKnowledgeBuilder();
builder.add(ResourceFactory.newClassPathResource("org/drools/examples/exampleProcess.bpmn"),
            ResourceType.BPMN);
KnowledgeBase kbase = builder.newKnowledgeBase();

// Starting a process
StatefulKnowledgeSession session
    = knowledgeBase.newStatefulKnowledgeSession();
session.startProcess("org.drools.examples.ExampleProcess");

// Signaling a process
session.signalEvent("myEvent", data);
Domain-specific Work Items

Extend palette with domain-specific, declarative work items
Processes vs. Rules

**Process**
- Focus on control flow
- Procedural
- Generic scope
- Long-living
- Stable
- Independent

**Rules**
- Focus on data
- Declarative
- Specific case
- Immediate
- Dynamic
- Additive
Rules and Processes

SCOPE

generic

specific

COUPLING

tightly coupled

Decision Services

loosely coupled

Process Rules
Decision Service

Extract decision logic from process definition using business rules

- Identify decision points
- Separate life cycle
- Permanent or temporary
- Other advantages
  - Reuse
  - Higher-level
  - Complexity

```java
rule "High Risk if age < 21"
  ruleflow-group "RiskAssessment"
  when
    Person( age < 21 )
  then
    insert ( new RiskFactor(
      0.1, "Person is less than 21." ) );
end
```
Process Rules

Using rules inside process ...

- for decisions (gateway, wait state)
- define
  - exceptional situations
  - escalation
  - cancellation
  - inclusion / exclusion criteria
  - assignment
- override default process logic
  - could be temporary
  - could be automatic (listen to own events)
Exceptional Control Flow

90%

Split ➔ Task ➔ Wait ➔ Split ➔ Human Task

5%

Task ➔ End

3%

Task ➔ Task ➔ End

2%

Fault
Agility

= 

Variability

+ 

Change
Non-linear processes
Knowledge Life Cycle

- Model
- Analyze
- Deploy
- Monitor
- Execute

Drools
Employee evaluation

As part of your performance evaluation, you have to do a self-assessment.

Please fill in the following evaluation form:
Rate the overall performance:
Outstanding

Check any that apply:
- Displaying initiative
- Thriving on change
- Good communication skills

Complete
Business Activity Monitoring
Process com.sample.evaluation

Process Definition Id: com.sample.evaluation
Total number of instances: 1
Number of instances last 24h: 1
Number of active instances: 1

Start Process Instances

<table>
<thead>
<tr>
<th>Id</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11 september 2009</td>
<td>18:23:37</td>
</tr>
</tbody>
</table>
Drools Fusion

Complex Event Processing (CEP)
Event
a significant change of state at a particular point in time

Complex event
an abstraction of other events called its members

Complex event processing
processing multiple events with the goal of identifying the meaningful events within the event cloud
Characteristics

- Huge volume of events, but only a few of real interest
- Usually events are immutable
- Usually queries/rules have to run in reactive mode
- Strong temporal relationships between events
- Individual events are usually not important
- The composition and aggregation of events is important
A "Simple" Example
Features

- Event Detection
  - Events are facts
  - Time + duration
  - Event streams + pipelines

- Event Correlation
  - 13 temporal operators
  - Sliding windows
    - Accumulate ... over window:time( 12h )
  - Temporal dimension support

- Event abstraction
  - Compose complex events
declare StockTick

@role( event )
symbol : String
price : double
end

rule “Shipment not picked up in time”
when
   Shipment( $pickupTime : scheduledPickupTime )
   not ShipmentPickup( this before $pickupTime )
then
   // shipment not picked up, action required.
end
Summary
Summary

- 3 different paradigms
  - Business Rules
  - Business Processes
  - Complex Event Processing

- Unification and integration
  - Life cycle
  - Tooling
  - API
Questions?

Useful links

Website
   http://www.jboss.org/drools/

Reference manual
   http://www.jboss.org/drools/documentation.html

Blog
   http://blog.athico.com/

Mailing lists (forum through nabble)
   http://www.jboss.org/drools/lists.html