Simplifying the Development of Rules
Using Domain Specific Languages in DROOLS

Ludwig Ostermayer, Geng Sun, Dietmar Seipel

University of Würzburg, Dept. of Computer Science

INAP 2013 – Kiel, 12.09.2013
<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
</tr>
<tr>
<td>2. <strong>DROOLS</strong></td>
</tr>
<tr>
<td>- DROOLS Rule Language</td>
</tr>
<tr>
<td>- DSLs in DROOLS</td>
</tr>
<tr>
<td>3. <strong>DSL Design</strong></td>
</tr>
<tr>
<td>- DSL Templates</td>
</tr>
<tr>
<td>- Annotations</td>
</tr>
<tr>
<td>4. <strong>The Tool</strong></td>
</tr>
<tr>
<td>- DSL Editor</td>
</tr>
<tr>
<td>- DSL Rule Editor</td>
</tr>
<tr>
<td>- Attribute Editor</td>
</tr>
<tr>
<td>5. <strong>PROLOG–Based Analysis</strong></td>
</tr>
<tr>
<td>- Templates</td>
</tr>
<tr>
<td>- Rules</td>
</tr>
</tbody>
</table>
the business rules approach provides a methodology for system development creating applications as *white boxes*
business logic is visible, because it is separated into business rules
Business Rule Management Systems (BRMS) have been developed
in BRMS you can define, deploy, execute, monitor, and maintain decision logic
DROOLS is a BRMS
a Domain Specific Language (DSL) is a programming language of limited expressiveness for a particular problem domain
the DSL Rule Generator (DSLR) is a tool improving the development process in DROOLS
BRMS Project

Project with Trinodis GmbH:

- development of business rules applications
- usage of PROLOG technology and related technology
- several case studies in real business scenarios
- analysis of business rules
- business analyst–friendly annotation of business rules
Introducing DROOLS

- **DROOLS** – the Business Logic Integration Platform
- JAVA–based
- developed by the JBOSS community

**DROOLS** consists of several modules:
- Expert (Inference Engine)
- Guvnor (Business Rules Manager)
- Fusion (Event Processing/Temporal Reasoning)
- Planner (Automated Planning)
Production Rule System DROOLS EXPERT
DROOLS Rule Language

- formerly: rules were written in XML
- XML format is not supported any more
- now: rules are written basically in the DROOLS Rule Language
- simple text files with the extension .drl
- rules are packed by namespaces – referred to as package
- global variables can be defined and used within rules via the globals statement
- complex logic can be outsourced and used within rules via the functions statement
A Rule in the DROOLS Rule Language

```drl
package LoanApproval

rule "microfinance"
when
  application: Application(
    loan_amount < 10000,
    duration_year < 5
  )
  customer: Customer(
    credit_report == "good"
  )
then
  application.approval();
  application.setInterest_rate(0.028);
end
```
rules in a DSL are developed in DROOLS in two steps

first step:
- designing DSL expressions with the mapping information to the DROOLS Rule Language
- save to a file with extension .dsl

second step:
- use the expressions of the DSL to write rules
- save into a file with the extension .dslr

DROOLS transforms the rules of the .dslr-file internally into the DROOLS Rule Language

usage of the mapping information contained in the .dsl-file
**Fragment of a .dsl-File**

```plaintext
[when] The customer with
  monthly_income is greater than {value1} and credit_report is {value2}
  =
  customer: Customer(
    monthly_income > {value1},
    credit_report == {value2} )
```

- **[when]** indicates the expression as condition
- **[then]** is used for an action block
- the single equality sign "=" separates the expressions in DSL format from the corresponding DRL format
A Rule written in the DSL

rule "microfinance"

when

The loan with
  loan_amount is smaller than 5000
and duration_year is no more than 3
The customer with
  monthly_income is greater than 2000
and credit_report is "good"

then

The loan will be approved
Set the interest_rate of the loan to 0.025

end
DROOLS DSL Editor

- a very "basic" DSL editor
- lacks user friendliness and functionality
- no content assist
- no package explorer for JAVA classes, attributes or methods
- no component to simply create rules in a DSL
- lacks reusability
DSL Generator

- a few guided steps to create rules in a readable format and with correct syntax
- development a DSL with the aid of generic templates
- graphical editors help during the construction of syntactical correct rules
- a brief example illustrating the usage
A Template for a DSL Expression

The `instance` with

```plaintext
  #field is smaller than \{value\} =
#instance: #class(#field < \{value\})
```

- generic templates for expressions containing the mapping information between DSL and DRL
- keywords and parameters in a template can be replaced
- templates are designed in JAVA
- but transformed to XML to improve readability
Fragment of a Template in XML Format

```xml
<template>
  <class>%class</class>
  <instance>%instance</instance>
  ...
  <condition>
    <domain>Common</domain>
    <dsl>
      <expression>
        The %instance with %field is smaller than {value}
      </expression>
      <code>
        %instance:%class(%field < {value})
      </code>
    </dsl>
  </condition>
</template>
```
Annotations

- form of syntactic meta-data added to JAVA source code
- used to accomplish multilingual DSLs
- classes, attributes and methods can be annotated
- keywords are replaced by annotation values

```java
@EnExpression(value = "amount of loan")
@GerExpression(value = "Kredithoehe")
private double loan_amount;
```
Components

5 components for the creation of rules, each has a graphical user interface

- Basic DSL Editor – designing simple expressions
- Complex Condition Editor – composing conditions
- DSL Rule Editor – designing rules
- Value Editor – assigning values
- Attribute Editor – editing meta-data of rules
Basic DSL Editor
## Complex Condition Editor

### DSL - Based Analysis

<table>
<thead>
<tr>
<th>Scope</th>
<th>conjunction</th>
<th>Expression</th>
<th>Rule Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>[when] and</td>
<td>The application with loan_amount is greater than {value}</td>
<td>application:Application( loan_amount &gt; {value})</td>
<td></td>
</tr>
<tr>
<td>[when] and</td>
<td>The application with loan_amount is less than {value}</td>
<td>application:Application( loan_amount &lt; {value})</td>
<td></td>
</tr>
<tr>
<td>[when] and</td>
<td>The application with duration_year is less than {value}</td>
<td>application:Application( duration_year &lt; {value})</td>
<td></td>
</tr>
<tr>
<td>[when] and</td>
<td>The application with duration_year is greater than {value}</td>
<td>application:Application( duration_year &gt; {value})</td>
<td></td>
</tr>
<tr>
<td>[when] or</td>
<td>The application with collateral_evaluation value is greater than {value}</td>
<td>application:Application( collateral_EV &gt; {value})</td>
<td></td>
</tr>
<tr>
<td>[when] and</td>
<td>the collateral could be frozen</td>
<td>application:Application(isCollateralFrozen())</td>
<td></td>
</tr>
<tr>
<td>[when] and</td>
<td>The customer with credit_report is {value}</td>
<td>customer:Customer( credit_report == {value})</td>
<td></td>
</tr>
<tr>
<td>[when] and</td>
<td>The customer with monthly_income is greater than {value}</td>
<td>customer:Customer( monthly_income &gt; {value})</td>
<td></td>
</tr>
</tbody>
</table>

### DSLR

<table>
<thead>
<tr>
<th>Scope</th>
<th>Expression</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>[when]</td>
<td>The application with loan_amount is less than {value1} and duration_year is less than {value2} or c...</td>
<td>application:Application( loan_amount &lt;</td>
</tr>
</tbody>
</table>
DSL Rule Editor
Value Editor

Expression:
The application with loan_amount is at least \( value1 \)
and duration_year is no more than \( value2 \)
and collateral evaluation value is at least \( value3 \)
and collateral could be frozen

Code:
application: Application
\( loan_amount \geq value1 \)
\&\& duration_year \leq value2
\&\& collateral_EV \geq value3
\&\& isCollateralFrozen()

Package Explorer:
- adsdasd.dsl
- Base.dsl
- bin
- lib
- LoanBase(En).dsl
- pic
- rules
- src
  - anno
  - booking
  - Drools
  - LoanApproval
    - Application.java
    - Customer.java
    - main
    - org

Value Editor:
- NAME: value1
  - VALUE: 10000
- NAME: value2
  - VALUE: 5
- NAME: value3
  - VALUE: collateral_EV collateral evaluation value

Save
# Attribute Editor

![Attribute Editor](image)

<table>
<thead>
<tr>
<th>name</th>
<th>enabled</th>
<th>salience</th>
<th>no-loop</th>
<th>activation-group</th>
<th>date-effective</th>
<th>date-expires</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;microfinance A&quot;</td>
<td></td>
<td>100</td>
<td>true</td>
<td></td>
<td>&quot;2013-01-28 23:59:...&quot;</td>
<td>&quot;2013-01-29 23:59:...&quot;</td>
</tr>
<tr>
<td>&quot;microfinance C&quot;</td>
<td>true</td>
<td>80</td>
<td>true</td>
<td></td>
<td>&quot;2012-06-28 23:59:...&quot;</td>
<td>&quot;2012-06-29 23:59:...&quot;</td>
</tr>
<tr>
<td>&quot;Big finance A&quot;</td>
<td></td>
<td>60</td>
<td></td>
<td></td>
<td>bigfinance</td>
<td></td>
</tr>
<tr>
<td>&quot;Big finance B&quot;</td>
<td></td>
<td>60</td>
<td></td>
<td></td>
<td>bigfinance</td>
<td></td>
</tr>
<tr>
<td>&quot;Big finance C&quot;</td>
<td></td>
<td>60</td>
<td></td>
<td></td>
<td>bigfinance</td>
<td></td>
</tr>
<tr>
<td>&quot;large loan A&quot;</td>
<td></td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;large loan B&quot;</td>
<td></td>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;large loan C&quot;</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Simple Rule created with DSLR Generator

rule "microfinance"
when
  The loan with
    amount of loan is smaller than 5000  
    and the duration in years is no more than 3
  The customer with
    monthly income in dollar is greater than 2000
    and the credit report is "good"
then
  The loan will be approved
  Set the rate of interest of the loan to 0.075
end
PROLOG–Based Analysis

Templates
- duplicates
- keywords
- ...

Rules
- analysis of the rules created with templates
- analysis and visualization of the interaction of rules
- anomalies:
  - duplicates
  - contradictions
  - ambiguities
Analysis of Templates

```xml
<template>
  <class>#class</class>
  <instance>#instance</instance>
  ...
  <condition>
    <domain>Common</domain>
    <dsl>
      <expression>
        The #instance with #field is smaller than {value}
      </expression>
      <code>
        #instance:#class(#field < {value})
      </code>
    </dsl>
  </condition>
</template>
```
Anomalies in Templates

- analysis and update with the XML query, transformation and update language FNQUERY
- dsl_anomaly(+DSL_1, +DSL_2, -Anomaly): checks <dsl> elements for anomalies

\[
\text{dsl\_anomaly}(Dsl\_1, Dsl\_2, \text{Anomaly}) :- \\
\text{member}(\text{Tag, [expression, code]}), \\
X := Dsl\_1/\text{Tag}, \\
Y := Dsl\_2/\text{Tag}, \\
\text{equivalent}(X, Y), \\
\text{Anomaly} = \text{duplicate}(\text{Tag, X, Y}).
\]
Analysis of Rules

```prolog
package LoanApproval

rule "microfinance"
when
    application: Application(
        loan_amount < 10000,
        duration_year < 5 )
    customer: Customer(
        credit_report == "good")
then
    application.approval();
    application.setInterest_rate(0.028);
end
```
set the status to approved and the interest rate to 0.028

```
application(Cid, Loan, Duration, A, B, 0.028, approved) :-
    application(Cid, Loan, Duration, A, B, _, _),
    Loan < 5000,
    Duration < 3,
    customer(Cid, _, Credit_Report, Income),
    Income > 2000,
    Credit_Report = good.
```

Transformed rules are analyzed, but usually cannot be executed.
Rule Anomalies

If there is

- a condition referencing a fact with identifier Id, e.g., application, and
- an instruction `modifyObject(Id)` in the action block,

then DROOLS fires all appropriate rules again, which results in a loop. This can be avoided by the no-loop attribute.

- `drools_anomaly(+Prolog, -Anomaly)`

reports the anomalies on backtracking.

Further anomalies:

- duplicates,
- contradictions,
- in connection with prioritization.
Visualization: Proof Trees

- red circles: derived atoms
- blue boxes: rule nodes are labeled by numbers
- orange circles: basic predicates from the configuration
Conclusions

What we have presented:

- a tool, DSLR Generator, for handling DSLs
- graphical user interfaces supporting the rule development
- reusable and generic DSL templates
- maintenance of the meta-data for the rules
- analysis of templates

Future work:

- extension of the PROLOG–based anomaly analysis, especially for rules
- a library of DSL templates for various problem domains
Thank You for Your Attention

Questions?

http://www1.informatik.uni-wuerzburg.de/en/staff/ostermayer_ludwig/ludwig.ostermayer@uni-wuerzburg.de