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

Introduction

- 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

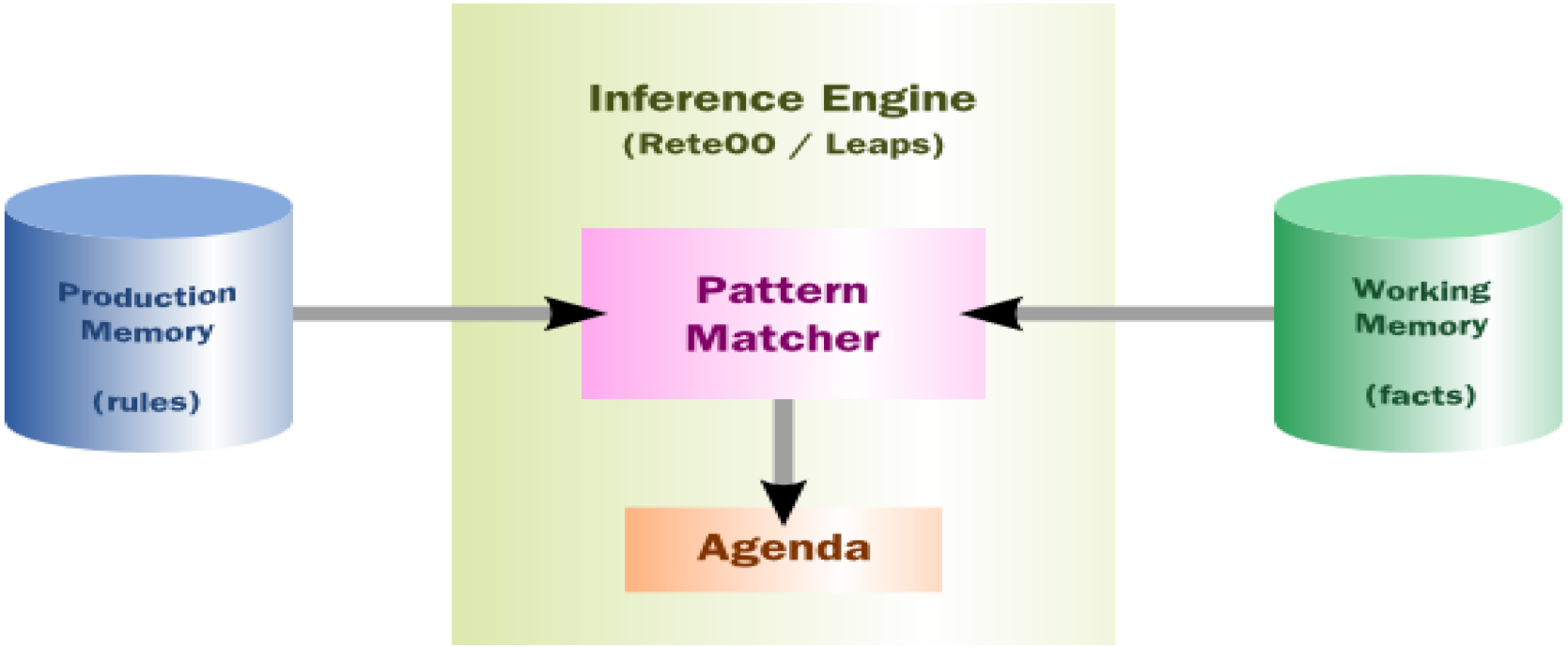
Project with Trinodis GmbH:

- 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 Languange
- 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

```
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
```

DSLs in DROOLS

- 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

```
[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

DSLR 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
   #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

```
<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>
```

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

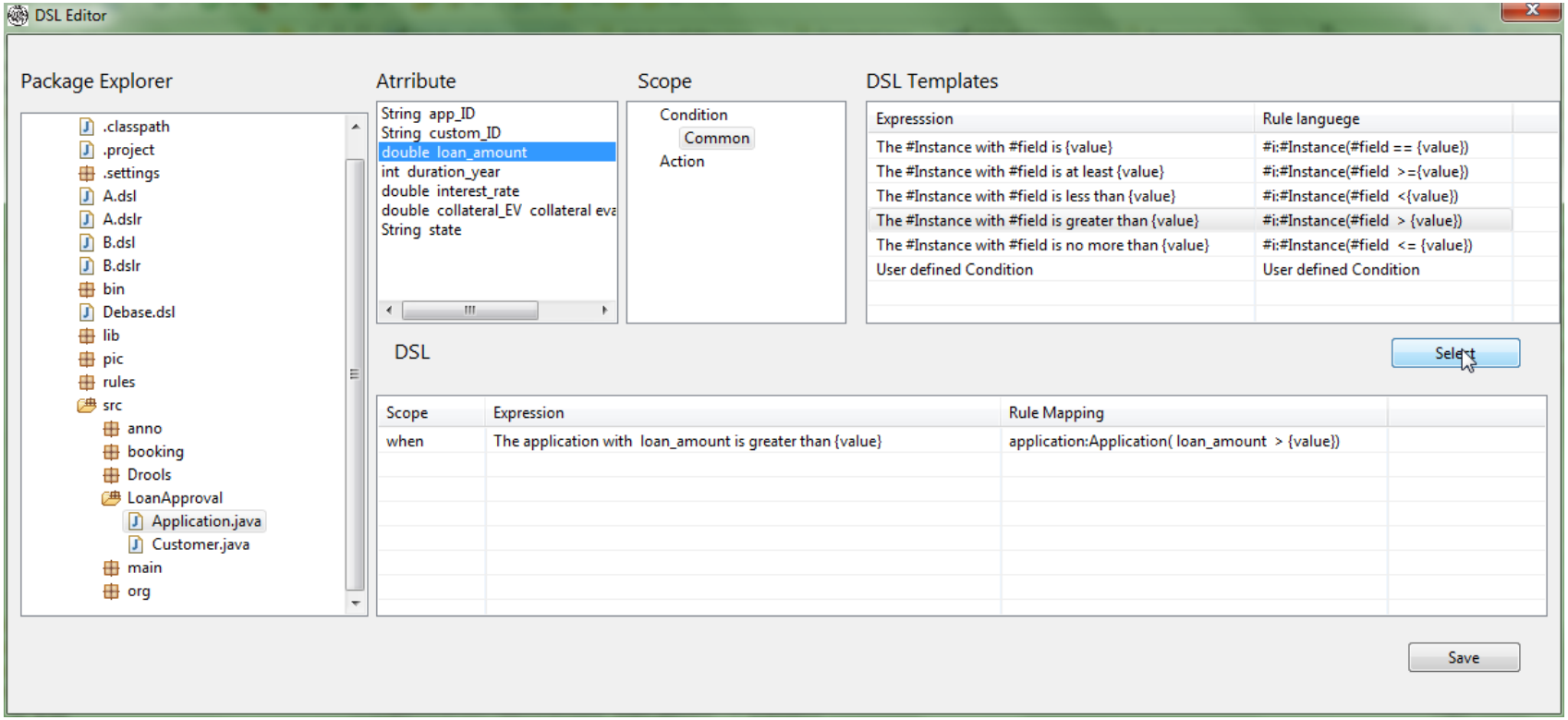
```
@EnExpression(value = "amount of loan")
@GerExpression(value = "Kredithoehe")
private double loan_amount;
```

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

Complex Conditions Editor

Open

DSL

Scope	conjuncti...	Expression	Rule Mapping
[when]	and	The application with loan_amount is greater than {value}	application:Application(loan_amount > {value})
[when]	and	The application with loan_amount is less than {value}	application:Application(loan_amount < {value})
[when]	and	The application with duration_year is less than {value}	application:Application(duration_year < {value})
[when]	and	The application with duration_year is greater than {value}	application:Application(duration_year > {value})
[when]	or	The application with collateral evaluation value is greater than {value}	application:Application(collateral_EV > {value})
[when]	and	the collateral could be frozen	application:Application(isCollateralFrozen())
[when]	and	The customer with credit_report is {value}	customer:Customer(credit_report == {value})
[when]	and	The customer with monthly_income is greater than {value}	customer:Customer(monthly_income > {value})

DSLR

Merge

Scope	Expression	Code
[when]	The application with loan_amount is less than {value1} and duration_year is less than {value2} or c...	application:Application(loan_amount <

Save

DSL Rule Editor

DSLR Editor

open

DSL

Scope	Expression	Rule Mapping
[when]	The application with (loan_amount is less than {value1} and duration_...	application:Application((loan_amount < {value1} .
[when]	The customer with credit_report is {value1} and monthly_income is gr...	customer:Customer(credit_report == {value1} &&
[then]	The application will be approved	application.approval();
[then]	The application will be denied	application.deny();
[then]	Set the interest_rate of the application to {value}	applicationnstance.setInterest_rate({value});

DSLR

Scope	Expression
[when]	The application with (loan_amount is less than 10000 and duration_year is less than 5) or collateral evaluation value is greate...
[when]	The customer with credit_report is "good" and monthly_income is greater than 2000
[then]	The application will be approved
[then]	Set the interest_rate of the application to 0.025

clear Rule Name example save

Value 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  >={value1}
 &&  duration_year  <= {value2}
 &&  collateral_EV  >={value3}
 && isCollateralFrozen())
```

Package Explorer

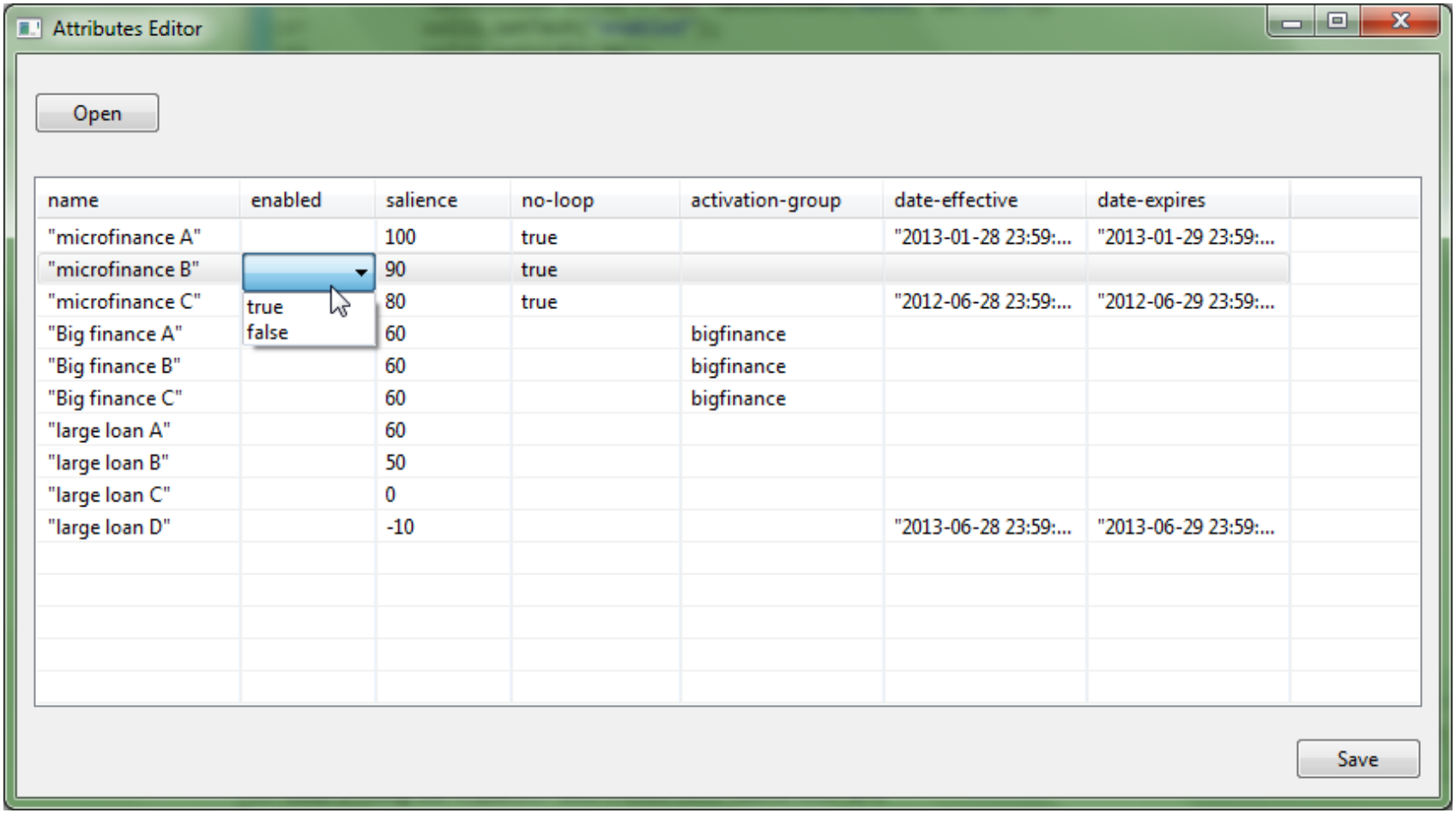
- adsdasd.dsl
- Base.dsl
- bin
- lib
- LoanBase(En).dsl
- pic
- rules
- src
  - anno
  - booking
  - Drools
  - LoanApproval
    - Application.java
    - Customer.java
  - main
  - org

```
String  app_ID
String  custom_ID
double  loan_amount
int  duration_year
double  interest_rate
double  collateral_EV  collateral ev
String  state
```

NAME	VALUE
value1	10000
value2	5
value3	collateral_EV collateral evaluation value

Save

Attribute Editor



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

```
<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>
```

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

```
dsl_anomaly(Dsl_1, Dsl_2, Anomaly) :-
   member(Tag, [expression, code]),
   X := Dsl_1/Tag,
   Y := Dsl_2/Tag,
   equivalent(X, Y),
   Anomaly = duplicate(Tag, X, Y).
```

Analysis of Rules

```
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
```

Transformation to Logical Rules

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.

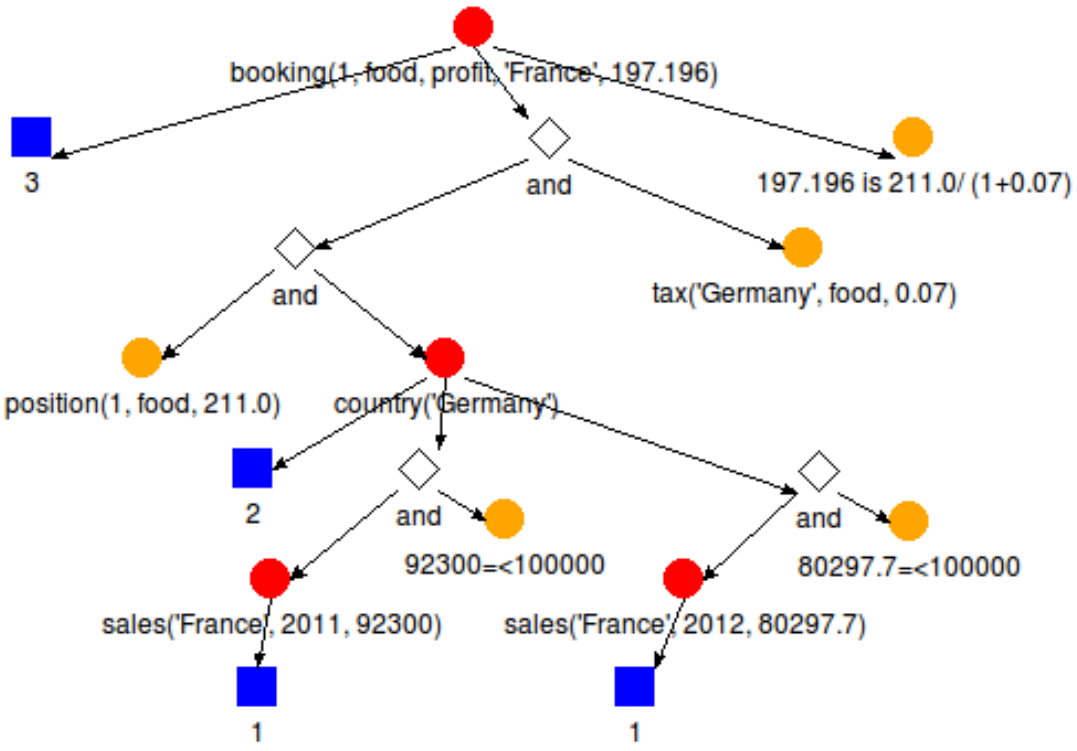
- `drools_anomaly(+Prolog, -Anomaly)`

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