

*Training Session on RuleML Technology*

# Loan Processor Suite: Transforming, Visualizing, and Querying Datalog RuleML Decision Rules

Harold Boley

Faculty of Computer Science  
University of New Brunswick  
Fredericton, NB, Canada

Decision CAMP, 4-6 November 2013

eBay Town Hall, San Jose, CA

# How to Process a Loan?

“Loans are processed after a credit application is filled out and turned in to the loan officer. This will start the loan processing course, first by pre-qualifying the buyer by reviewing the credit history and debt load of the applicant.”

<http://www.ask.com/question/how-to-process-a-loan>

# Use Decision Rules and Data Facts!

The Loan Processor Suite is a series of formal documents for testing and learning ***Graph inscribed logic (Grailog), RuleML, SVG, XSLT, POSL, OO jDREW, etc.***

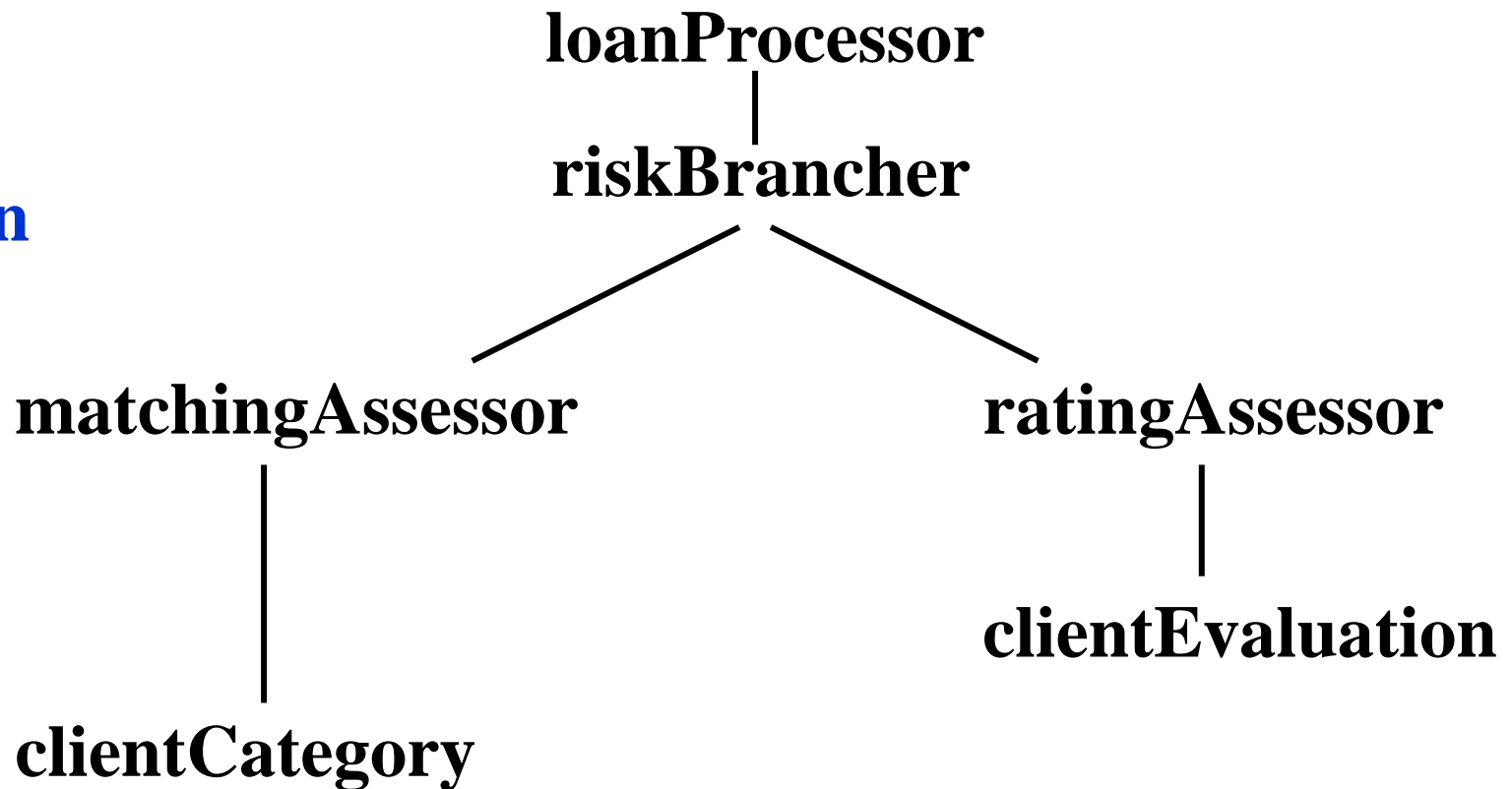
They exemplify Datalog RuleML decision rules and data facts that are being transformed, visualized, and queried

See: <http://ruleml.org/papers/Primer>

[http://wiki.ruleml.org/index.php/Grailog#Test\\_Suites](http://wiki.ruleml.org/index.php/Grailog#Test_Suites)

# Decision Rule / Data Fact Architecture

**Decision  
Rules:**



**Data  
Facts:**

See: <http://www.cs.unb.ca/~boley/Grailog/LoanProcessor/LoanProcessor.txt>

# Joint Agile Development of Decision Rules and Data Facts

The predicate **loanProcessor** considers requests with ?AmountAsk in (0 500000],

also fixing ?RiskLevel = 1000 and ?RatiMin = 0.8 in a call to **riskBrancher**:

Its "<" rule invokes **matchingAssessor**, hence **clientCategory**, for table lookup.

Its ">=" rule uses **ratingAssessor**, hence **clientEvaluation**, for deep analysis.

Possible outcomes are either failure, without ?AmountGrant binding, or success, with ?AmountGrant bound to the loan.

Predicate definitions can flexibly combine rules and facts using Prolog/Datalog-like POSL syntax (e.g., ":" as the "IF" infix):

<http://ojs.academypublisher.com/index.php/jetwi/article/view/0204343353>

# Decision Rule: loanProcessor

```

loanProcessor(?Client,?AmountAsk,?AmountGrant) :-
    % loanProcessor(In,In,Out)

    greaterThan(?AmountAsk,0),
    % Positive loan request

    lessThanOrEqual(?AmountAsk,500000),
    % up to half a million.

    riskBrancher(?Client,?AmountAsk,?AmountGrant,1000,0.8).
    % ...,?RiskLevel,?RatiMin).

```

# Decision Rules: riskBrancher

**riskBrancher**(?Client,?AmountAsk,?AmountGrant,  
?RiskLevel,?RatiMin) :-

**lessThan**(?AmountAsk,?RiskLevel),  
% *Deterministic ...*

**matchingAssessor**(?Client,?AmountAsk,?AmountGrant).

**riskBrancher**(?Client,?AmountAsk,?AmountGrant,  
?RiskLevel,?RatiMin ) :-

**greaterThanOrEqualTo**(?AmountAsk,?RiskLevel),  
% *... branch*

**ratingAssessor**(?Client,?AmountAsk,?AmountGrant,  
?RatiMin).

# Decision Rules: matchingAssessor

**matchingAssessor**(?Client,?AmountAsk,?AmountGrant) :-

**clientCategory**(?Client,gold),

*% ?Client fact matches gold category*

**multiply**(?AmountGrant,?AmountAsk,0.75).

*% ?AmountGrant = ?AmountAsk \* 0.75*

**matchingAssessor**(?Client,?Amount,?Amount) :-

*% ?Amount = ?AmountGrant = ?AmountAsk IF*

**clientCategory**(?Client,platinum).

*% ?Client fact matches platinum category*



# Decision Rule: ratingAssessor

```
ratingAssessor(?Client,?AmountAsk,?AmountGrant,  
               ?RatiMin) :-  
  
    clientEvaluation(?Client,?AmountAsk,?Rating),  
        % Data analysis binds ?Rating in [0,1].  
  
    greaterThanOrEqual(?Rating,?RatiMin),  
        % For ?Rating < ?RatiMin: not approved  
  
    multiply(?AmountGrant,?AmountAsk,?Rating).  
        % ?AmountGrant = ?AmountAsk * ?Rating
```

# Data Facts: clientCategory

Facts store database table  
which captures qualitative categories  
of previous quantitative analysis

**clientCategory**(nilper,silver).

**clientCategory**(bold,gold).

**clientCategory**(claritum,platinum).

# Data Facts: clientEvaluation

(Non-ground) facts cache ratings  
from deep client data analysis,  
which could be made conditional  
on (currently free) ?AmountAsk

**clientEvaluation**(nilper,?AmountAsk,0.77).

**clientEvaluation**(bold,?AmountAsk,0.79).

**clientEvaluation**(claritum,?AmountAsk,0.91).

**clientEvaluation**(ralcitum,?AmountAsk,0.91).

# Query Rules: negTest

**negTest(1,?AG) :- loanProcessor(nilper,200,?AG).**

*% Failure since 200 < 1000 and  
% no silver match*

**negTest(2,?AG) :- loanProcessor(nilper,1100,?AG).**

*% Failure since 1100 > 1000 and  
% 0.77 < 0.8*

**negTest(3,?AG) :- loanProcessor(bold,1100,?AG).**

*% Failure since 1100 > 1000 and  
% 0.79 < 0.8*

**negTest(4,?AG) :- loanProcessor(ralcitum,200,?AG).**

*% Failure since 200 < 1000 and  
% no category*

# Query Rule: posTest

**posTest(?AG1,?AG2,?AG3,?AG4) :-**

**loanProcessor(bold,200,?AG1),**

*% Success with ?AG1 = 150 since 200 < 1000 and  
% gold match*

**loanProcessor(claritum,200,?AG2),**

*% Success with ?AG2 = 200 since 200 < 1000 and  
% platinum match*

**loanProcessor(claritum,400000,?AG3),**

*% Success with ?AG3 = 364k since 400k > 1k and  
% 0.91 > 0.8*

**loanProcessor(ralcitum,500000,?AG4).**

*% Success with ?AG4 = 455k since 500k > 1k and  
% 0.91 > 0.8*

# Transforming POSL to RuleML/XML

**loanProcessor**(?Client,?AmountAsk,?AmountGrant) :-  
                                   *% loanProcessor(In,In,Out)*

greaterThan(?AmountAsk,0),  
                                   *% Positive loan request*

lessThanOrEqualTo(?AmountAsk,500000),  
                                   *% up to half a million.*

**riskBrancher**(?Client,?AmountAsk,?AmountGrant,1000,0.8).  
                                   *% ...,?RiskLevel,?RatiMin).*

[OO jDREW 1.0 POSL/RuleML Translator](#)



```
<Implies>
  <And>
    <Atom>
      <Rel>greaterThan</Rel>
      <Var>AmountAsk</Var>
      <Ind>0</Ind>
    </Atom>
    <Atom>
      <Rel>lessThanOrEqualTo</Rel>
      <Var>AmountAsk</Var>
      <Ind>500000</Ind>
    </Atom>
    <Atom>
      <Rel>riskBrancher</Rel>
      <Var>Client</Var>
      <Var>AmountAsk</Var>
      <Var>AmountGrant</Var>
      <Ind>1000</Ind>
      <Ind>0.8</Ind>
    </Atom>
  </And>
  <Atom>
    <Rel>loanProcessor</Rel>
    <Var>Client</Var>
    <Var>AmountAsk</Var>
    <Var>AmountGrant</Var>
  </Atom>
</Implies>
```

# Visualizing RuleML/XML as Grailog/SVG

## <Implies>

<And>

<Atom>

<Rel>greaterThan</Rel>

<Var>AmountAsk</Var>

<Ind>0</Ind>

</Atom>

<Atom>

<Rel>lessThanOrEqual</Rel>

<Var>AmountAsk</Var>

<Ind>500000</Ind>

</Atom>

<Atom>

<Rel>riskBrancher</Rel>

<Var>Client</Var>

<Var>AmountAsk</Var>

<Var>AmountGrant</Var>

<Ind>1000</Ind>

<Ind>0.8</Ind>

</Atom>

</And>

<Atom>

<Rel>loanProcessor</Rel>

<Var>Client</Var>

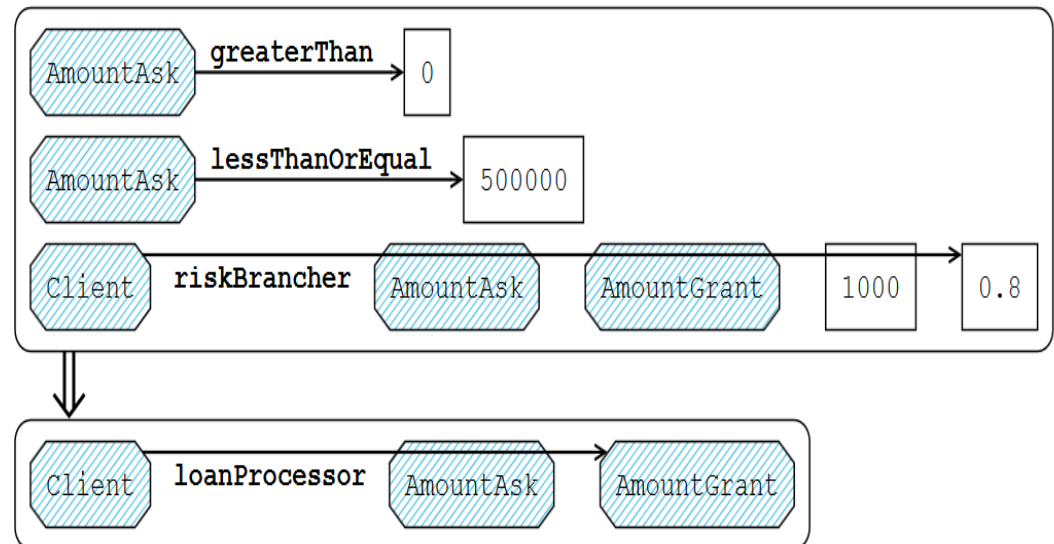
<Var>AmountAsk</Var>

<Var>AmountGrant</Var>

</Atom>

## </Implies>

[Grailog KS Viz](#)



Complete: <http://www.cs.unb.ca/~boley/Grailog/LoanProcessor/LoanProcessor.svg>  
(View Page Source)

# Querying with OO jDREW

OO jDREW

File Options Run

Type definition Knowledge base Query

Query: `posTest(?AG1,?AG2,?AG3,?AG4)`

Input format: ☐ RuleML ☒ POSL ☐ Type query

Issue query Next

Solution:

```

Stop( ^ );-posTest( ^ 150.0 : Real, 200 : Integer, 364000.0 : Real, 455000.0 : Real).
posTest( ^ 150.0 : Real, 200 : Integer, 364000.0 : Real, 455000.0 : Real):-loanProcessor( ^
loanProcessor( ^ bold : String, 200 : Integer, 150.0 : Real):-greaterThan("sidrew-at-200>0" ^ 200 : Integer, 0 : Integer),lessThanOrEqual("sidrew-lte-200<=500000" ^ 200 : Integer, 500000 : Integer).
greaterThan("sidrew-at-200>0" ^ 200 : Integer, 0 : Integer).
lessThanOrEqual("sidrew-lte-200<=500000" ^ 200 : Integer, 500000 : Integer).
riskBrancher( ^ bold : String, 200 : Integer, 150.0 : Real, 1000 : Integer, 0.8 : Real):-lessThan("sidrew-lt-200<1000" ^ 200 : Integer, 1000 : Integer),lessThan("sidrew-lt-200<1000" ^ 200 : Integer, 1000 : Integer).
lessThan("sidrew-lt-200<1000" ^ 200 : Integer, 1000 : Integer).
matchingAssessor( ^ bold : String, 200 : Integer, 150.0 : Real):-clientCategory( ^ bold : String, gold : String),multiply("sidrew-mul-200.0x0.75" ^ 150.0 : Real, 200 : Integer, 0.75 : Real).
clientCategory( ^ bold : String, gold : String).
multiply("sidrew-mul-200.0x0.75" ^ 150.0 : Real, 200 : Integer, 0.75 : Real).
loanProcessor( ^ claritum : String, 200 : Integer, 200 : Integer):-greaterThan("sidrew-at-200>0" ^ 200 : Integer, 0 : Integer),lessThanOrEqual("sidrew-lte-200<=500000" ^ 200 : Integer, 500000 : Integer).
greaterThan("sidrew-at-200>0" ^ 200 : Integer, 0 : Integer).
lessThanOrEqual("sidrew-lte-200<=500000" ^ 200 : Integer, 500000 : Integer).
riskBrancher( ^ claritum : String, 200 : Integer, 200 : Integer, 1000 : Integer, 0.8 : Real):-lessThan("sidrew-lt-200<1000" ^ 200 : Integer, 1000 : Integer),lessThan("sidrew-lt-200<1000" ^ 200 : Integer, 1000 : Integer).
lessThan("sidrew-lt-200<1000" ^ 200 : Integer, 1000 : Integer).
matchingAssessor( ^ claritum : String, 200 : Integer, 200 : Integer):-clientCategory( ^ claritum : String, platinum : String).
clientCategory( ^ claritum : String, platinum : String).
loanProcessor( ^ claritum : String, 400000 : Integer, 364000.0 : Real):-greaterThan("sidrew-at-400000>0" ^ 400000 : Integer, 0 : Integer),lessThanOrEqual("sidrew-lte-400000<=500000" ^ 400000 : Integer, 500000 : Integer).
greaterThan("sidrew-at-400000>0" ^ 400000 : Integer, 0 : Integer).
lessThanOrEqual("sidrew-lte-400000<=500000" ^ 400000 : Integer, 500000 : Integer).
riskBrancher( ^ claritum : String, 400000 : Integer, 364000.0 : Real, 1000 : Integer, 0.8 : Real):-greaterThanOrEqual("sidrew-qte-400000>=1000" ^ 400000 : Integer, 1000 : Integer),greaterThanOrEqual("sidrew-qte-400000>=1000" ^ 400000 : Integer, 1000 : Integer).
greaterThanOrEqual("sidrew-qte-400000>=1000" ^ 400000 : Integer, 1000 : Integer).
ratingAssessor( ^ claritum : String, 400000 : Integer, 364000.0 : Real, 0.8 : Real):-clientEvaluation( ^ claritum : String, 400000 : Integer, 0.91 : Real),clientEvaluation( ^ claritum : String, 400000 : Integer, 0.91 : Real).
greaterThanOrEqual("sidrew-qte-0.91>=0.8" ^ 0.91 : Real, 0.8 : Real).
multiply("sidrew-mul-400000.0x0.91" ^ 364000.0 : Real, 400000 : Integer, 0.91 : Real).
loanProcessor( ^ claritum : String, 500000 : Integer, 455000.0 : Real):-greaterThan("sidrew-at-500000>0" ^ 500000 : Integer, 0 : Integer),lessThanOrEqual("sidrew-lte-500000<=500000" ^ 500000 : Integer, 500000 : Integer).
greaterThan("sidrew-at-500000>0" ^ 500000 : Integer, 0 : Integer).
lessThanOrEqual("sidrew-lte-500000<=500000" ^ 500000 : Integer, 500000 : Integer).
riskBrancher( ^ claritum : String, 500000 : Integer, 455000.0 : Real, 1000 : Integer, 0.8 : Real):-greaterThanOrEqual("sidrew-qte-500000>=1000" ^ 500000 : Integer, 1000 : Integer),greaterThanOrEqual("sidrew-qte-500000>=1000" ^ 500000 : Integer, 1000 : Integer).
greaterThanOrEqual("sidrew-qte-500000>=1000" ^ 500000 : Integer, 1000 : Integer).
ratingAssessor( ^ claritum : String, 500000 : Integer, 455000.0 : Real, 0.8 : Real):-clientEvaluation( ^ claritum : String, 500000 : Integer, 0.91 : Real),clientEvaluation( ^ claritum : String, 500000 : Integer, 0.91 : Real).
greaterThanOrEqual("sidrew-qte-0.91>=0.8" ^ 0.91 : Real, 0.8 : Real).
multiply("sidrew-mul-500000.0x0.91" ^ 455000.0 : Real, 500000 : Integer, 0.91 : Real).

```

Variable bindings:

Variable	Binding
?AG4	455000.0 : Real
?AG3	364000.0 : Real
?AG2	200 : Integer
?AG1	150.0 : Real

See: [OO jDREW 1.0 snapshot \(Java Web Start\)](#)



# Exercises

- 1) Give client ralcitum the category platinum and explain what will happen for the positive and negative tests
- 2) Augment the decision rules so that clients of category silver (e.g., client nilper) will obtain half of the amount asked. Hint: Model category silver in analogy to category gold
- 3) Update the entire LP Suite (transformation, visualization, and querying) using 1) and 2)