JBoss Rules – Viva Le Drools

Declarative Behavioural Modelling
An Integrated AI approach

Mark Proctor
General Improvements

- 4.0 is faster and uses less memory than 3.0.x
- API Redesign
  - Explicit stateless and stateful sessions apis
  - Both apis support async working memory action method
  - Thread safety improvements
More Expression

- 3.0.x only allows comma separated field constraints. 'or' could be used at the CE level, but resulted in subrule generation.
  - Can now use && and || inside the pattern for multiple values on the same field and across files – no subrule generation.
  - Person( age > 30 && < 40 || hair =="black" )

- 3.0.x auto-have autovivification of variables in dialect expressions
  - Before: Cheese( oldPrice : oldPrice, newPrice == ( oldPrice * 1.10 ) )
  - Now: Cheese( newPrice == ( oldPrice * 1.10 ) )
More Expression

• 3.0.x had to always declare the variable, causing clutter, can now access direct properties of pattern variables.
  • Before: p : Person(personId : id)
    i : Item(id == personId, value > 100 )
  • Now: p : Person()
    i : Item(id == p.id, value > 100 )

• Eval rewrite for complex expressions
  • Before: Person($pets:pets
eval($pets['rover'].type == "dog")
  • Now: Person( pets['rover'].type == "dog" )
Pluggeable Dialects

• Return-value, predicate, evals and consequences can now specify dialects, now supports Java and MVEL.
  • Cheese(type == "stilton",
    eval(price == (new Integer(5) + 5)),
    price == (new Integer(5) + 5))
  • Assert (new Person()) ( name = “mark”, age = 31 );
Why MVEL

- Reflection/bytecode (JIT) compilation and execution modes.
  - For huge systems we need to be able to avoid excessive bytecode generation, but still have the option for bytecode JIT for performance sensitive areas.

- Fast reflection mode.
  - We originally started with our own language JFDI, which was designed to be a simple and fast reflection based language, the idea is all work is done at compile time so runtime is just a series of reflection invokers. This design has been carried through to MVEL, so that it has good enough reflection performance. Where as other languages have to drop reflection mode and use bytecode to get any reasonable level of performance.

- Pluggeable resolvers.
  - Dictionary population is too slow, MVEL can resolve it’s variable direct from the provided resolvers, which we make array based for performance.

- Size.
  - MVEL is currently <>
Why MVEL

- Custom language extensions.
  - MVEL is extending the language to support rule friendly constructs, in particular block setters. So I can do "modify (person) ( age += 1, location = "london")" with the ability to treat that as a transaction block so I can run before and after interceptors on the entire block. This is made easier through the use of macros, so we can define our own keywords and have them expanded into mvel code.

- Static/Inferred typed or dynamic modes.
  - Variables can be untyped and totally dynamic.
  - Variables can be statically typed or type can be inferred, casting is supported.
  - Optional verifier for "typed mode", disallows dynamic variables and ensures all types and method calls are correct. Which helps with.
    - Authoring time validation.
    - Code completion.
    - Refactoring.

- Configurable language feature support.
  - Language features can be turned off.
  - We don't want imperative flow structures in the "then" part, no 'if' 'switch' etc. Rules should be declarative, "when this do that" not "when this maybe do that".
Powerful new CEs

• Forall
  • True when the pattern is true for all facts
  • Forall( Bus(color == “red”) )

• From
  • Pulls and unifies against none working memory data
    • Can call hibernate queries
    • Sub fields
    • Restaurant( rating == “five star” )
      from hbSession.getNamedQuery( “restaurant query” ).
      setProperties( key1 : value1, key2 : value2).list()
Powerful new CEs

• Collect
  • Allows you to use cardinality
  • When there are more than 6 red buses
  • List(size > 6) from collect ( Bus(color == "red") )
  • 'from' can be chained. Following is true if all items in a cart have a price greater than 10
  • List(size == ($list.size)) from collect(Item(price > 10) ) from $cart.items
Powerful new CEs

- **Accumulate**
  - More powerful 'collect' allows you to execute actions on each matched fact in the set
  - $total: Integer()
    from accumulate( $item : Item( )
    init(count = 0; total=0)
    action(count++; total += $item.price)
    result( return total/count )


Line Debugger and new Rete Viewer
Eclipse Guided Editor

Rule Builder

**IF**
- age is less than 42
- Person name is equal to Bob
- Vehicle [car1] type is not equal to (code here)
  - There is a Storm alert of type (code here)
    - severity rating is not more than (code here)

**THEN**
Rule Flow

![Rule Flow Diagram](image)
Rule Flow

Constraint editor

Name: constraint
Priority: 1
Always true

Textual Editor
Pet(type = fish)

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraints</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Split</td>
</tr>
<tr>
<td>Type</td>
<td>XOR</td>
</tr>
</tbody>
</table>
RuleFlow

- Execution control of sets of rules, a node can fire 1 or it can fire 10K rules.
- Is not transactional
- Does not persist per propagation
- No configurable services
BRMS

- Web 2.0 based BRMS using
  - Built with JackRabbit JCR and GWT/Seam
  - Rule/package management
    - version control, categorisation, configuration, deployment
  - Upload
    - drls, dsls, excel decision tables, dependencies (jars)
- Web Authoring
  - Text pasting
  - Guided editor
MyDT

Upload new version: Browse... Upload

Download current version: Download

This is a decision table in a spreadsheet (XLS). Typically they contain many rules in one sheet.

<documentation>
IF

Person

\[ \text{age} \leq 42 \]

\[ \text{age} > 21 \]

Board \( b_1 \)

There is no Board

THEN

Set \( b_1 \) cost 1200

(options)

<documentation>
BRMS

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 snapshot called: NewSnapshot was successfully created.
The image shows a screenshot of the JBoss Business Rules Management System (BRMS) interface. The interface displays a rule file known as `Rule_1`. The rule is written in Drools, a rule-engine language, and the rule body contains logic that adds costs based on certain conditions.