JBoss Rules – Viva Le Drools

Declarative Behavioural Modelling

An Integrated AI approach

Mark Proctor
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 ) )
More Expression

• 3.0.x had to always declare the variable, causing cluter, 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.
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

IF

- **Person**
  - age is less than 42
  - 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 Builder  DRL Preview
Rule Flow
Rule Flow
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>
Rule 1

IF

Person

age less than or equal to 42
age greater than 21

Board [b]

There is no

THEN

Set [b|f] cost 1200

(options)

View source Validate

<documentation>
Categories aid in managing large numbers of rules/assets. A shallow hierarchy is recommended.
The page at http://localhost:8080 says:
The snapshot called: NewSnapshot was successfully created.

Create a snapshot for deployment.

A package snapshot is essentially a read only 'locked in' and labelled view of a package at a point in time, which can be used for deployment.

Choose or create snapshot name: NEW: NewSnapshot
Comment: commenting

Create new snapshot.
rule "Rule_1"
when
  Person( age <= 42 , age > 21 )
  b : Board()
  not Board( cost > 1200 )
then
  b.setCost( 1200 )
end