Modernizing Drools: From Production Rules to Blended AI

Mark Proctor
Chief Architect - Rules, BPM, Optimisation
Red Hat
OPS5 Onwards (Slowly)
Drools Improvements
  Property Reactive
  OO Path
  Prolog(ish) Backward Chaining
  Game Loops
Slow Progress
Foundations
Canonical Model
Pluggable Knowledge
  Bayesian Belief
Pluggable Beliefs Systems
  Simple Truth Maintenance
  Defeasible Reasoning
  Bayesian Network
Units
Papers
  Recommended Reading
(p mb17
  (goal ^status active ^type on ^object <o>)
  (object ^name <o> ^at <p>)
  (monkey ^at <p> ^holds nil)
  -->
  (write (crlf) "climb onto" <o>)
  (modify 3 ^on <o>)
  (modify 1 ^status satisfied))

OPS

• Official Production System.
• pre OPS5 linear Search.
• Conflict Resolution

McDermott, Charles L (Charles Lanny) Forgy.

OPS5 1981

• Language
• Rete
• LEX/MEA
CLIPS

CLIPS
• Syntax based on ART (Haley Systems)
• Modules based on GoldWorks
• PIE Clips Modules

CLIPS 1993
• Lisp (ish)
• exists/forall
• Subnetworks
• Modules
• Simple TMS

(defrule climb-directly ""
  ?goal <- (goal-is-to (action on) (arguments ?obj))
  (thing (name ?obj) (location ?place) (on-top-of ?on))
  ?monkey <- (monkey (location ?place) (on-top-of ?on) (holding blank))
  =>
  (printout t "Monkey climbs onto the " ?obj "." crlf)
  (modify ?monkey (on-top-of ?obj))
  (retract ?goal))
Jess

Jess
- Java implementation of Clips syntax

Jess 2006
- Accumulate
- Slot Specific
- Backward Chaining (ish)

(deftemplate employee (slot salary) (slot name))
(defrule count-highly-paid-employees
  ?c <- (accumulate (bind ?count 0) ;; initializer
                  (bind ?count (+ ?count 1))) ;; action
    ?count ;; result
  (employee (salary ?s&:(> ?s 100000)))) ;; CE
=>
  (printout t ?c " employees make more than $100000/year." crlf))
Drools

• Clips/Jess derivative with Java-like language

Drools 7.x
• Accumulate
• Temporal Operators
• Property Reactive
• Prolog(ish) Backward Chaining
• OOPath
Drools Improvements

Operations 5 Onwards (Slowly)

**Drools Improvements**
- Property Reactive
- OO Path
- Prolog(ish) Backward Chaining
- Game Loops

Slow Progress
Foundations
Canonical Model

Pluggable Knowledge
- Bayesian Belief

Pluggable Beliefs Systems
- Simple Truth Maintenance
- Defeasible Reasoning
- Bayesian Network

Units
Papers
Recommended Reading
OPS5 Onwards (Slowly)

**Drools Improvements**
- Property Reactive
  - OO Path
  - Prolog(ish) Backward Chaining
  - Game Loops
- Slow Progress
- Foundations
- Canonical Model
- Pluggable Knowledge
  - Bayesian Belief
- Pluggable Beliefs Systems
  - Simple Truth Maintenance
  - Defeasible Reasoning
  - Bayesian Network
- Units
- Papers
  - Recommended Reading
@PropertyReactive
public class Employee {
    int salary;
    int lengthOfService

    // getters and setters below
}

rule “Salary award for min 2 years service” when
e : Employee( lengthOfService >= 2 )
then
    modify( e ) { setSalary( e.getSalary() * 1.05 ) }
end

rule “Salary award for min 8 years service” when
e : Employee( lengthOfService >= 8 )
then
    modify( e ) { setSalary( e.getSalary() * 1.02 ) }
end

“Advances in RETE Pattern Matching” (1986)
rule “Salary award for min 2 years service” when
e : Employee( lengthOfService > 2 )
then
  modify( e ) { setSalary( e.getSalary() * 1.05 ) };
end

rule “Salary award for min 8 years service” when
e : Employee( lengthOfService > 8 )
then
  modify( e ) { setSalary( e.getSalary() * 1.02 ) };
end

rule “Salary award for min 2 years service” when
e : Employee( lengthOfService >= 2 )
  @Watch( !salary )
then
  modify( e ) { setSalary( e.getSalary() * 1.05 ) };
end

rule “Salary award for min 8 years service” when
e : Employee( lengthOfService >= 8 )
  @Watch( !salary )
then
  modify( e ) { setSalary( e.getSalary() * 1.02 ) };
end

rule “Record Salary Changes” when
e : Employee( ) @Watch( salary )
then
  insert( new SalaryChange( e, e.getSalary() ) );
end
Drools Improvements

- Property Reactive

**OOPath**
- Prolog(ish) Backward Chaining
- Game Loops

Slow Progress

Foundations

Canonical Model

Pluggable Knowledge
- Bayesian Belief

Pluggable Beliefs Systems
- Simple Truth Maintenance
- Defeasible Reasoning
- Bayesian Network

Units

Papers
- Recommended Reading
rule R1 when
    $student : Student ()
    $plan : Plan ( owner == $student.name )
    $exam : Exam( plan == $plan.code, course == "Big Data" )
    $grade : Grade( exam == $exam.code )
then
    // RHS
end

rule R3 when
    Student( $grade: /plan/exams{course == "Big Data"}/grades )
then
    /* RHS */
end

<table>
<thead>
<tr>
<th></th>
<th>Batch (ms)</th>
<th>Incremental (ms)</th>
<th>Total (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td>5596</td>
<td>1844</td>
<td>7440</td>
</tr>
<tr>
<td>From</td>
<td>2698</td>
<td>2912</td>
<td>5610</td>
</tr>
<tr>
<td>OPath</td>
<td>3373</td>
<td>1368</td>
<td>4741</td>
</tr>
</tbody>
</table>

Table 3. Benchmark results with 400,000 items (ms.)
OOPath

Access by index

Student( $grade : /plan/exams[0]{ course == "Big Data"}/grades )

Inline cast for type safety

Student( $grade : /plan/exams{ #PracticalExam, lab == "hazard safe", course == "Material Explosions"}/grades )

Indexed back reference

A( $var: /b/c/d{ f1 == ../../../f2}/e ) // the ../../../ back references to the ‘b’ field access

Variable back reference

A( $var: /$b : b/c/d{ f1 == $b.f2}/e ) // the $b is inline bound for later use

Back tracking

A( $var: /$b : b/c/d{ f1 == $b.f2}/$b/f2 ) // $var is bound to results of the f2 access

Out of Pattern use

$student : Student()
$grade : /$student/plan/exams{course == "Big Data"}/grades;
Drools Improvements

Prolog(ish) Backward Chaining

Game Loops

Slow Progress

Foundations

Canonical Model

Pluggable Knowledge

Bayesian Belief

Pluggable Beliefs Systems

Simple Truth Maintenance

Defeasible Reasoning

Bayesian Network

Units

Papers

Recommended Reading
Prolog(ish) Backward Chaining

query isContainedIn(String x, String y)
Location(x, y;)
or
( Location(z, y; ) and isContainedIn(x, z; ) )
end
query isContainedIn( String x, String y )
  Location( x, y; )
or
  ( Location( z, y; ) and isContainedIn( x, z; ) )
end

rule “Find Key Rule” when
  Goal( name == "Find Key" )
  isContainedIn("Find", "House"; )
then
  System.out.println( "Key in the House" );
end

rule “Find Key Rule” when
  Goal( name == "Find Key" )
  ?isContainedIn("Find", "House"; )
then
  System.out.println( "Key in the House" );
end
OPS5 Onwards (Slowly)

**Drools Improvements**
- Property Reactive
- OO Path
- Prolog(ish) Backward Chaining

**Game Loops**
- Slow Progress
- Foundations
- Canonical Model
- Pluggable Knowledge
  - Bayesian Belief
- Pluggable Beliefs Systems
  - Simple Truth Maintenance
  - Defeasible Reasoning
  - Bayesian Network

Units
Papers
Recommended Reading

Drools Improvements

Game Loops
Games Loops

```plaintext
rule GameLoop when
    r : Run()
then
    setFocus( "Draw" );
    setFocus( "Bullet" );
    setFocus( "Move" );
    setFocus( "Keys" );
end

rule Draw when
    r : Run()
then
    modify( r ) {} // force loop
    fpsTimer.incFrame();
end

rule Collision agenda-group "Bullet" when
    b : Bullet() @watch( y )
    i : Invader( x < b.x, x + width > b.x, y > b.y)
    Run()
then
    modify( i ) { alive = false }
end
```
Games Loops (TODO)

- Existence-Drive and Data-Drive
- “Design of a Rule-Oriented System for Implementing Expertise,” ROSIE (1979)
- Incremental and NonIncremental

```plaintext
rule GameLoop when
  r : Run()
then
  setFocus( "Draw" );
  setFocus( "Bullet" );
  setFocus( "Move" );
  setFocus( "Keys" );
end

rule Draw when
  r : Run()
then
  modify( r ) {} // force loop
  fpsTimer.incFrame();
end

rule Collision existenceDriven nonIncremental agenda-group "Bullet" when
  b : Bullet( ) @watch( y )
  i : Invader( x < b.x, x + width > b.x, y > b.y)
then
  modify( i ) { alive = false }
end
```
OPS5 Onwards (Slowly)
Drools Improvements
   Property Reactive
   OO Path
   Prolog(ish) Backward Chaining
   Game Loops
**Slow Progress**
Foundations
Canonical Model
Pluggable Knowledge
   Bayesian Belief
Pluggable Beliefs Systems
   Truth Maintenance
   Defeasible Reasoning
   Bayesian Network
Units
Papers
   Recommended Reading

**Slow Progress**
Is Still Progress
Still Here

Inference Engine

Modules

Pattern Matching

Working Memory

Agenda
“Today, I want to make clear that the future prospects for production rule technology are diminishing..... production rule technology is inadequate ”
(Paul Haley 2013)

http://haleyai.com/wordpress/2013/06/22/confessions-of-a-production-rule-vendor-part-1/
“Today, I want to make clear that the future prospects for production rule technology are diminishing..... production rule technology is inadequate” (Paul Haley 2013)

http://haleyai.com/wordpress/2013/06/22/confessions-of-a-production-rule-vendor-part-1/
Offshoots

Active DataBases
• Validation enforcement, triggers.
• Simpler semantics, no forward chaining.
• Without 2 phase. Does not allow mutation of selected tables.
• Scoped deterministic execution within the transaction

Decisioning
• DMN

CEP
• Power of production system LHS, with additional temporal reasoning
• No inference, rules are isolated

Reactive Programming with Pattern Matching
• Light Weight, simple facet of a production system
• No Reactive Joins, but can do passive joins
• Built into native language
public static class Customer {
    int String name;
    int discount = 0;
}

public static class GoldCustomer extends Customer {
}
public static class SilverCustomer extends Customer {
}
public static class BronzeCustomer extends Customer {
}

public static void main( String[] args ) {
    observable.subscribe( customer -> {
        switch (customer) {
        case GoldCustomer g:   {
            CriteriaQuery<Double> cr = cb.createQuery(Double.class);
            Root<ShoppingCart> root = cr.from(ShoppingCart.class);
            cr.select(cb.sum(root.get("items.value"))).where("customer = " + g.name);
            Query<Double> query = session.createQuery(cr);
            double total = query.getResultList().get(0);
            g.discount = total > 100 ? 20 : 15;
            break;
        }
        case SilverCustomer s: s.discount = 10; ........ break;
        case BronzeCustomer b: b.discount = 5; ........ break;
        default:               customer.discount = 0;
        }
    });
}
```csharp
var result = 
(from data in myData
join inData in incomingData
  on data.ID equals inData.ID
select new
{
    data.ID,
    inData.LOCATION,
    data.COUNT
}).ToList();
```
“It’s time to trade rule technology dating back to the 80’s for state of the art AI…

Artificial intelligence (AI) and natural language processing (NLP) have improved dramatically.

Logical reasoning technology has advanced while production rule technology remains stagnant” (Paul Haley 2018)

Still Here

Inference Engine

Pattern Matching

Agenda

Modules

Working Memory
Still Here
Foundations

OPS5 Onwards (Slowly)
Drools Improvements
  Property Reactive
  OO Path
  Prolog(ish) Backward Chaining
  Game Loops
Slow Progress

Foundations

Canonical Model
Pluggable Knowledge
  Bayesian Belief
Pluggable Beliefs Systems
  Simple Truth Maintenance
  Defeasible Reasoning
  Bayesian Network

Units
Papers
  Recommended Reading
Swiss Army Knife
The Building Blocks of a Swiss Army Knife
OPS5 Onwards (Slowly)
Drools Improvements
  Property Reactive
  OO Path
  Prolog(ish) Backward Chaining
  Game Loops
Slow Progress
Foundations
Canonical Model
  Pluggable Knowledge
    Bayesian Belief
Pluggable Beliefs Systems
  Simple Truth Maintenance
  Defeasible Reasoning
  Bayesian Network
Units
Papers
  Recommended Reading

Canonical Model
Canonical Production Rules

DRL Parser

Rule
- In memory object
- LHS has 1..n Patterns
- Pattern has 1..n Expressions
- RHS is compiled Java bytecode
- Expressions not yet bytecode
- expr.eval( fact[] )

Building
- Analyse Expr to determine
- indexes, property reactive masks
- Optimise expressions to bytecode

Run
- Lazy Rete derivative
- Single efficient incremental algorithm
- Sequential supported as an optimization of this same algorithm
Canonical Production Rules

- In memory object
  - LHS has 1..n Patterns
  - Pattern has 1..n Expressions
  - RHS is compiled Java bytecode
  - Expressions not yet bytecode
  - `expr.eval(fact[])`

Building
- Analyse Expr to determine
  - indexes, property reactive masks
  - Optimise expressions to bytecode

Run
- Lazy Rete derivative
  - Single efficient incremental algorithm
  - Sequential supported as an optimization of this same algorithm

- Class definitions in `.java` files
  - Every expression is a black box function
  - indexes, property reactive masks

Building
- No analysis or code generation needed
  - Canonical model contains everything it needs.

Run
- Support a pluggable API for network evaluation.
  - Discrimination network of functions, is useful no matter what algorithm you use.
Canonical Production Rules

Rule rule = rule("beta")
  .build(expr("exprA", markV, p -> p.getName().equals("Mark"))
    .indexedBy(String.class, ConstraintType.EQUAL, 1, Person::getName, "Mark")
    .reactOn("name"), // also react on age, see RuleDescr.lookAheadFieldsOfIdentifier
  bind(markAge).as(markV, Person::getAge).reactOn("age"),
  expr("exprB", olderV, p -> !p.getName().equals("Mark"))
    .indexedBy(String.class, ConstraintType.NOT_EQUAL, 1, Person::getName, "Mark")
    .reactOn("name"),
  expr("exprC", olderV, markAge, (p1, age) -> p1.getAge() > age)
    .indexedBy(int.class, ConstraintType.GREATER_THAN, 0, Person::getAge, int.class::cast)
    .reactOn("age"),
  on(olderV, markV).execute((drools, p1, p2) ->
    drools.insert(p1.getName() + " is older than " + p2.getName()));
OPS5 Onwards (Slowly)
Drools Improvements
  Property Reactive
  OO Path
  Prolog(ish) Backward Chaining
  Game Loops
Slow Progress
Foundations
Canonical Model

**Pluggable Knowledge**

Bayesian Belief

Pluggable Beliefs Systems
  Simple Truth Maintenance
  Defeasible Reasoning
  Bayesian Network

Units
Papers
  Recommended Reading
Pluggable Knowledge

- NonIncremental Cursor
- Incremental
- Discrimination Network
- Canonical Model
- Model
- Parser
  - DRL Parser
  - Prolog Parser
  - DMN Parser
- Runtime
- Weaver
- Assembler
Pluggable Knowledge - Bayesian Networks

- Bayesian Networks

```
<table>
<thead>
<tr>
<th>CLOUDY</th>
<th>SPRINKLER</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>0.5</td>
</tr>
<tr>
<td>T</td>
<td>0.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLOUDY</th>
<th>RAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>0.8</td>
</tr>
<tr>
<td>T</td>
<td>0.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WET GRASS</th>
<th>SPRINKLER</th>
<th>RAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>0.1</td>
<td>0.9</td>
</tr>
<tr>
<td>T</td>
<td>0.1</td>
<td>0.9</td>
</tr>
<tr>
<td>T</td>
<td>0.01</td>
<td>0.99</td>
</tr>
</tbody>
</table>
```
Bayesian Networks

XMLBIF

```java
[ 'assemblers' : [ new org.drools.beliefs.bayes.assembler.BayesAssemblerService() ],
  'weavers' : [ new org.drools.beliefs.bayes.weaver.BayesWeaverService() ],
  'runtimes' : [ new org.drools.beliefs.bayes.runtime.BayesRuntimeService() ] ]
```
KnowledgeBuilder kBuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kBuilder.add( ResourceFactory.newClassPathResource("Garden.xmlbif"), ResourceType.BAYES );

KnowledgeBase kBase = KnowledgeBaseFactory.newKnowledgeBase();
kBase.addKnowledgePackages( kBuilder.getKnowledgePackages() );

StatefulKnowledgeSession kSession = kBase.newStatefulKnowledgeSession();

BayesRuntime bayesRuntime = kSession.getKieRuntime(BayesRuntime.class);
BayesInstance<Garden> instance = bayesRuntime.createInstance(Garden.class);
Pluggable Knowledge - Bayesian Networks

KnowledgeBuilder kBuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kBuilder.add( ResourceFactory.newClassPathResource("Garden.xmlbif"), ResourceType.BAYES );

KnowledgeBase kBase = KnowledgeBaseFactory.newKnowledgeBase();
kBase.addKnowledgePackages( kBuilder.getKnowledgePackages() );

StatefulKnowledgeSession kSession = kBase.newStatefulKnowledgeSession();
BayesRuntime bayesRuntime = ksession.getKieRuntime(BayesRuntime.class);
BayesInstance<Garden> instance = bayesRuntime.createInstance(Garden.class);

BayesVariable var = ( BayesVariable ) instance.getFieldNames().get( "WetGrass" );
bayesInstance.setDecided(var, true);
bayesInstance.setLikelyhood( var, [1.0, 0.0] );

BayesVariable var = ( BayesVariable ) instance.getFieldNames().get( "Cloudy" );
bayesInstance.setDecided(var, true);
bayesInstance.setLikelyhood( var, [1.0, 0.0] )
Pluggable Knowledge - Bayesian Networks

```java
KnowledgeBuilder kBuilder = KnowledgeBuilderFactory.newKnowledgeBuilderFactory();
kBuilder.add( ResourceFactory.newClassPathResource("Garden.xmlbif"), ResourceType.BAYES );

KnowledgeBase kBase = KnowledgeBaseFactory.newKnowledgeBase();
kBase.addKnowledgePackages( kBuilder.getKnowledgePackages() );

StatefulKnowledgeSession kSession = kBase.newStatefulKnowledgeSession();

BayesRuntime bayesRuntime = kSession.getKieRuntime(BayesRuntime.class);
BayesInstance<Garden> instance = bayesRuntime.createInstance(Garden.class);

BayesVariable var = ( BayesVariable ) instance.getFieldNames().get("WetGrass");
bayesInstance.setDecided(var, true);
bayesInstance.setLikelihood( var, [1.0,0.0] );

BayesVariable var = ( BayesVariable ) instance.getFieldNames().get("Cloudy");
bayesInstance.setDecided(var, true);
bayesInstance.setLikelihood( var, [1.0,0.0] );

Garden garden = instance.marginalize();
assertTrue(garden.isDecided());
assertTrue(garden.isRained());
```

KnowledgeBuilder kBuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kBuilder.add( ResourceFactory.newClassPathResource("Garden.xmlbif"), ResourceType.BAYES );

KnowledgeBase kBase = KnowledgeBaseFactory.newKnowledgeBase();
kBase.addKnowledgePackages( kBuilder.getKnowledgePackages() );

StatefulKnowledgeSession kSession = kBase.newStatefulKnowledgeSession();

BayesRuntime bayesRuntime = kSession.getKieRuntime(BayesRuntime.class);
BayesInstance<Garden> instance = bayesRuntime.createInstance(Garden.class);

BayesVariable var = ( BayesVariable ) instance.getFieldNames().get("WetGrass");
bayesInstance.setDecided(var, true);
bayesInstance.setLikelihood( var, [1.0,0.0] );

BayesVariable var = ( BayesVariable ) instance.getFieldNames().get("Cloudy");
bayesInstance.setDecided(var, true);
bayesInstance.setLikelihood( var, [1.0,0.0] );

Garden garden = instance.marginalize();
assertTrue(garden.isDecided());
assertTrue(garden.isRained());
Pluggable Belief Systems

Simple Truth Maintenance

Pluggable Beliefs Systems

Simple Truth Maintenance

Defeasible Reasoning
Bayesian Network

Pluggable Knowledge
Bayesian Belief
Bayesian Network

Units
Papers
Recommended Reading
Simple TMS

• A rule “logically” inserts an object
• When the rule is no longer true, the object is retracted.

rule "IsChild"
when
  $p : Person( age < 16 )
then
  logicalInsert( new IsChild( $p ) )
end

rule "IsAdult"
when
  $p : Person( age >= 16 )
then
  logicalInsert( new IsAdult( $p ) )
end

rule "Issue Child Bus Pass"
when
  $p : Person()
  IsChild( person =$p )
then
  logicalInsert( new ChildBusPass( $p ) );
end

rule "Issue Adult Bus Pass"
when
  $p : Person()
  IsAdult( person =$p )
then
  logicalInsert( new AdultBusPass( $p ) );
end
Pluggable Belief Systems

Defeasible Reasoning

Pluggable Beliefs Systems
Simple Truth Maintenance

Defeasible Reasoning
Bayesian Network

Units
Papers
Recommended Reading
Defeasible

rule "All Birds Fly" @Defeasible
when
  $b : \text{Bird}()$
then
  logicalInsert(new Fly( $b$ ));
end

rule "Penguins Don't Fly"
  @Defeasible @Defeater
when
  $b : \text{Bird}()$
  \text{Penguin}(b);
then
  logicalInsert(new Fly( $b$ ), "neg");
end

rule "Penguins With Rockets Fly"
  @Defeasible @Defeats("Penguins Don't Fly")
when
  $b : \text{Penguin}()$
  \text{Rocket}(b);
then
  logicalInsert(new Fly( $b$ ));
end
Defeasible

- Defeasible for Legal
  - ‘claim’, ’objection,’ ‘rebuttal,’ ‘attack,’ ‘refutation,’ ‘rebutting defeater’ and ‘undercutting defeater’

- Deontic logic and Temporal
  - Obligatory, Permitted, Forbidden
Pluggable Belief Systems

Bayesian Network

Pluggable Beliefs Systems
Simple Truth Maintenance
Defeasible Reasoning

Bayesian Network

Units
Papers
Recommended Reading
KnowledgeBuilder kBuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kBuilder.add( ResourceFactory.newClassPathResource("rules.drl"), ResourceType.DRL );
kBuilder.add( ResourceFactory.newClassPathResource("Garden.xmlbif"), ResourceType.BAYES );

KnowledgeBase kBase = KnowledgeBaseFactory.newKnowledgeBase();
kBase.addKnowledgePackages( kBuilder.getKnowledgePackages() );
BayesVariable var = (BayesVariable) instance.getFieldNames().get("WetGrass");
bayesInstance.setDecided(var, true);
bayesInstance.setLikelyhood(var, [1.0, 0.0]);

Garden garden = instance.marginalize();
FactHandle fh = ksession.insert(garden);
ksession.insert(new Cloud());
ksession.fireAllRules();

rule "I see clouds" when
g : Garden(decided == false)
exists Cloud()
then
insertLogical(bayesVar(g, 'Cloudy'),
evidenceFactory.create(new double[] {1.0, 0.0}));
end

rule "It probably rained" when
Garden(decided == true, rain == true)
then
System.out.println("Rain");
end
OPS5 Onwards (Slowly)
Drools Improvements
  Property Reactive
  OO Path
  Prolog(ish) Backward Chaining
  Game Loops
Slow Progress
Foundations
Canonical Model
Pluggable Knowledge
  Bayesian Belief
**Pluggable Beliefs Systems**
  Simple Truth Maintenance
  Defeasible Reasoning
  Bayesian Network
**Units**
Papers
  Recommended Reading
Venus Rule Language

- Rules and data are decoupled
- Declarative Guards

```plaintext
module stipulate_relations(standby[], same[])

    rule same_is_symmetric;
    if (TRUE) {enforce_symmetry(same[]);} 

    rule same_is_transitive;
    if (TRUE) {enforce_transitivity(same[]);} 

    rule standby_is_symmetric;
    if (TRUE) {enforce_symmetry(standby[]);} 

module enforce_symmetry(Relation r[])
{
    rule enforce;
    from r[*] s;
    from r[?] t;
    if(! (s.domain_oid == t.range_oid &&
        s.range_oid == t.domain_oid)) {
        Relation *ins = new Relation;
        ins->domain_oid = s.range_oid;
        ins->range_oid = s.domain_oid;
        r.insert(ins);
    }
}
```
package org.mypackage.myunit
public class AdultUnit implements RuleUnit {

    private int adultAge;

    private DataSource<Person> persons;

    public AdultUnit() { }
    public AdultUnit( DataSource<Person> persons, int age ) {
        this.persons = persons;
        this.age = age;
    }
}

package org.mypackage.myunit
public class AdultUnit implements RuleUnit {

    private int adultAge;

    private DataSource<Person> persons;

    public AdultUnit() {}
    public AdultUnit(DataSource<Person> persons, int age) {
        this.persons = persons;
        this.age = age;
    }
}

package org.mypackage.myunit
unit AdultUnit

rule Adult when
    $p : /persons{age >= adultAge}
then
    System.out.println($p.getName() + " is adult and greater than " + adultAge);
end
package org.mypackage.myunit
public class AdultUnit implements RuleUnit {

    private int adultAge;

    private DataSource<Person> persons;

    public AdultUnit() {}
    public AdultUnit(DataSource<Person> persons, int age) {
        this.persons = persons;
        this.age = age;
    }

    @Override
    public void onStart() { System.out.println(getName() + " started.");}

    @Override
    public void onEnd() { }
}

package org.mypackage.myunit
unit AdultUnit

rule Adult when
    $p : /persons{age >= adultAge}
then
    System.out.println($p.getName() + " is adult and greater than " + adultAge);
end
Guards

public class BoxOfficeUnit implements RuleUnit {
    private DataSource<BoxOffice> boxOffices;
}

package org.mypackage.myunit
unit BoxOfficeUnit

rule BoxOfficeIsOpen when
    $box: /boxOffices{ open }
then
drools.guard( TicketIssuerUnit.class );
end

public class TicketIssuerUnit implements RuleUnit {
    private DataSource<Person> persons;
    private DataSource<AdultTicket> tickets;
    private List<String> results;
}

package org.mypackage.myunit
unit TicketIssuerUnit

rule IssueAdultTicket when
    $p: /persons{ age >= 18 }
then
tickets.insert( new AdultTicket($p) );
end

rule RegisterAdultTicket when
    $t: /tickets
then
results.add( $t.getPerson().getName() );
end
Papers

Recommended Reading


Papers - Recommended Reading