

Introducing



by Mario Fusco

Red Hat – Senior Software Engineer

mario.fusco@gmail.com

twitter: [@mariofusco](https://twitter.com/mariofusco)



Drools Vision

Drools was born as a rule engine, but following the vision of becoming a **single platform for business modelling**, it realized it could achieve this goal only by leveraging **3 complementary business modelling techniques**:



Business Rule Management (Drools Expert)



Business Process Management (Drools Flow)



Complex Event Processing (Drools Fusion)

Drools 5 – Business Logic Integration Platform



**Drools
Expert**



**Drools
Flow**



**Drools
Fusion**



**Drools
Guvnor**



**Drools
Planner**



Business Logic integration System

What a rule-based program is

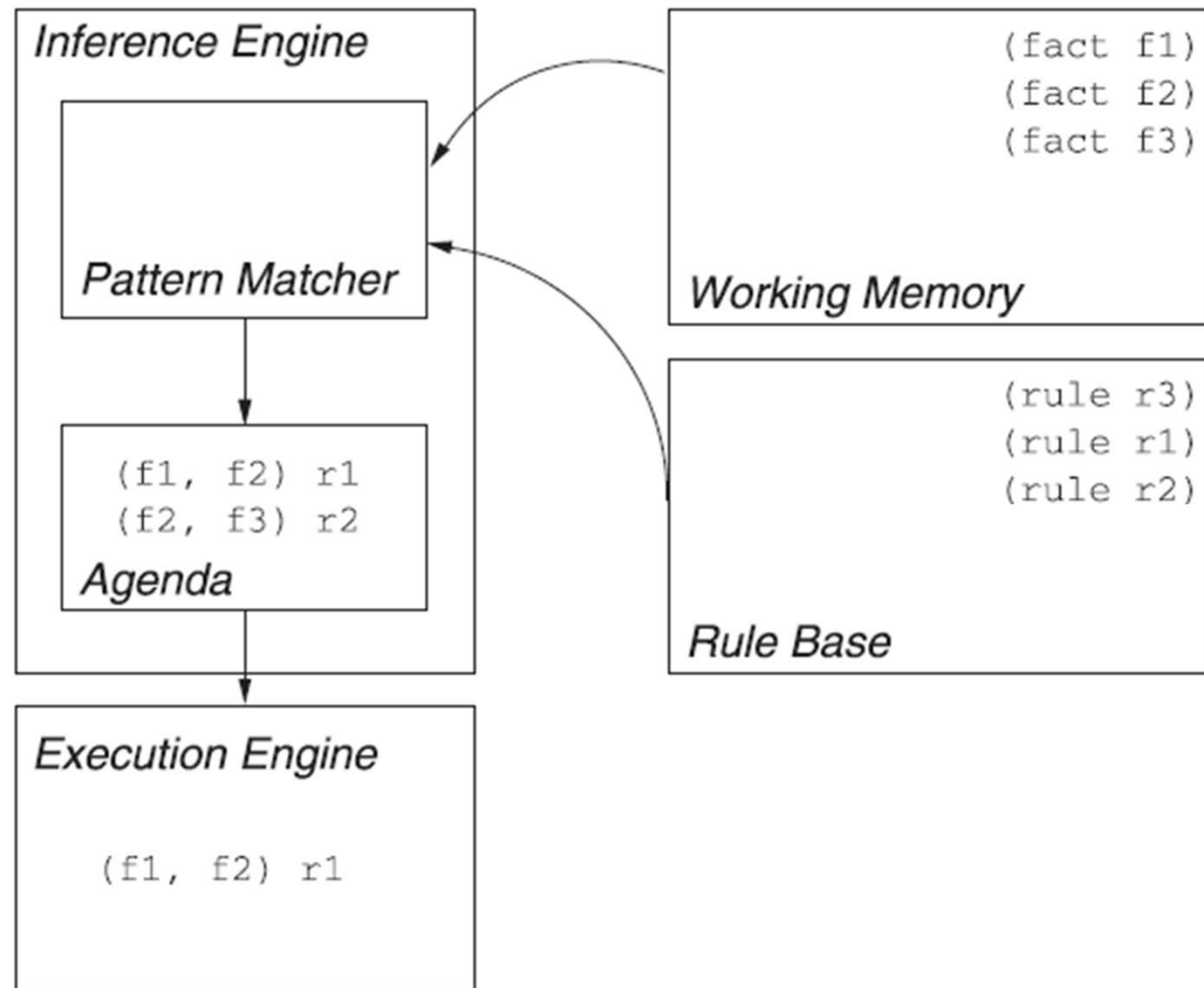
- A rule-based program is made up of **discrete rules**, each of which applies to some subset of the problem
- It is **simpler**, because you can concentrate on the rules for one situation at a time
- It can be more **flexible** in the face of fragmentary or poorly conditioned inputs
- Used for problems involving control, diagnosis, prediction, classification, pattern recognition ... in short, all problems without clear algorithmic solutions

Declarative Vs. Imperative
(What to do) (How to do it)

When should you use a Rule Engine?

- The problem is beyond any obvious algorithmic solution or it isn't fully understood
- The logic changes often
- Domain experts (or business analysts) are readily available, but are nontechnical
- You want to isolate the key parts of your business logic, *especially the really messy parts*

How a rule-based system works



Rule's anatomy

rule “<name>”

Quotes on Rule names are optional if the rule name has no spaces.

<attribute> <value>

when

Pattern-matching
against objects in the
Working Memory

<LHS>

then

salience	<int>
agenda-group	<string>
no-loop	<boolean>
auto-focus	<boolean>
duration	<long>
....	

<RHS>

end

Code executed when
a match is found

Imperative vs Declarative

A method must be called directly

Specific passing
of arguments

```
public void helloMark(Person person) {  
    if ( person.getName().equals( "mark" ) {  
        System.out.println( "Hello Mark" );  
    }  
}
```

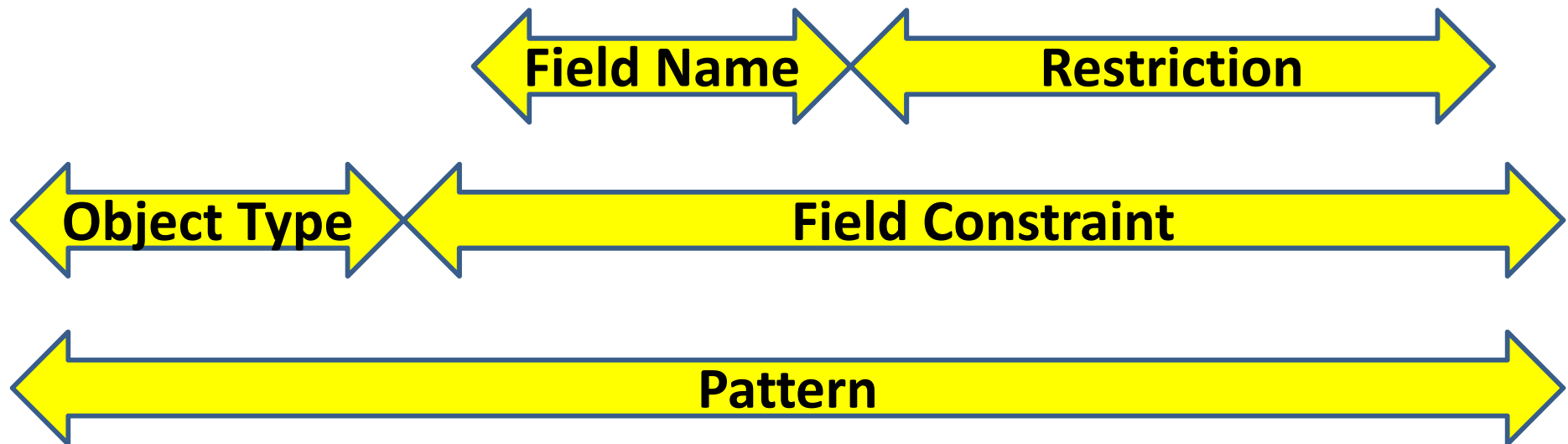
Rules can never be called directly

Specific instances cannot be
passed but are automatically
selected with pattern-matching

```
rule "Hello Mark"  
when  
    Person( name == "mark" )  
then  
    System.out.println( "Hello Mark" );  
end
```


What is a pattern

Person(name == “mark”)



Rule's definition

// Java

```
public class Applicant {  
    private String name;  
    private int age;  
    private boolean valid;  
    // getter and setter here  
}
```

// DRL

```
declare Applicant  
    name : String  
    age : int  
    valid : boolean  
end
```

```
rule "Is of valid age" when  
    $a : Applicant( age >= 18 )  
then  
    modify( $a ) { valid = true };  
end
```

Building

```
KnowledgeBuilder kbuilder =  
    KnowledgeBuilderFactory.newKnowledgeBuilder();  
kbuilder.add(  
    ResourceFactory.newClassPathResource("Rules.drl"),  
    ResourceType.DRL);  
KnowledgeBuilderErrors errors = kbuilder.getErrors();  
if (kbuilder.hasErrors()) {  
    System.err.println(kbuilder.getErrors().toString());  
}  
KnowledgeBase kbase =  
    KnowledgeBaseFactory.newKnowledgeBase();  
kbase.addKnowledgePackages(kbuilder.getKnowledgePackages());
```

Executing

```
StatelessKnowledgeSession ksession =  
    kbase.newStatelessKnowledgeSession();  
Applicant applicant = new Applicant( "Mr John Smith", 21 );  
assertFalse( applicant.isValid() );  
ksession.execute( applicant );  
assertTrue( applicant.isValid() );
```

More Pattern Examples

```
Person( $age : age )
```

```
Person( age == ( $age + 1 ) )
```

```
Person( age > 30 && < 40 || hair in ( "black", "brown" ) )
```

```
Person( pets contain $rover )
```

```
Person( pets[ 'rover' ].type == "dog" )
```

Conditional Elements

not Bus(color = “red”)

exists Bus(color = “red”)

forall (\$bus : Bus(color == “red”))

\$owner : Person(name == “mark”)

Pet(name == “rover”) **from** \$owner.pets

\$zipCode : ZipCode()

Person() **from** \$hbn.getNamedQuery(“Find People”)

.setParameters([“zipCode” : \$zipCode])

.list()

Hibernate session

'from' can work
on any expression

Timers & Calendars

rule R1

timer 1m30s

when

\$l : Light(status == "on")

then

modify(\$l) { status = "off" };

When the light is on, and has been on for 1m30s then turn it off

Execute now and after
1 hour duration only during weekday

rule R3

calendars "weekday"

timer (int:0 1h)

when

Alarm()

then

sendEmail("Alert!")

rule R2

timer (cron: 0 0/15 * * * *)

when

Alarm()

then

sendEmail("Alert!")

Send alert every
quarter of an hour

Truth Maintenance System (1)

```
rule "Issue Adult Bus Pass" when
    $p : Person( age >= 18 )
then
    insert(new AdultBusPass( $p ) );
end
```

Coupled logic

```
rule "Issue Child Bus Pass" when
    $p : Person( age < 18 )
then
    insert(new ChildBusPass( $p ) );
end
```

What happens
when the child
becomes adult?

Truth Maintenance System (2)

```
rule "Who Is Child" when  
    $p : Person( age < 18 )
```

De-couples the logic

```
then
```

```
    logicalInsert( new IsChild( $p ) )
```

```
end
```

```
rule "Issue Child Bus Pass" when
```

```
    $p : Person( )
```

```
    IsChild( person =$p )
```

```
then
```

```
    logicalInsert(new ChildBusPass( $p ) );
```

```
end
```

Encapsulate
knowledge providing
semantic abstractions
for this encapsulation

Maintains the truth by
automatically retracting

Drools Eclipse DEMO

Debug - StateExampleUsingSalience.drl - Eclipse SDK

File Edit Navigate Search Project Run Window Help

100%

Debug

StateExampleUsingSalience [Drools Application]

org.drools.examples.StateExampleUsingSalience at localhost:4861

Thread [main] (Suspended (breakpoint at line 8 in Rule_A_to_B_0))

Rule_A_to_B_0.consequence(KnowledgeHelper, State, FactHandle) line: 21

Rule_A_to_B_0ConsequenceInvoker.evaluate(KnowledgeHelper, WorkingMemory) line: 22

DefaultAgenda.fireActivation(Activation) line: not available

DefaultAgenda.fireNextItem(AgendaFilter) line: not available

ReteooWorkingMemory(AbstractWorkingMemory).fireAllRules(AgendaFilter) line: not available

ReteooWorkingMemory(AbstractWorkingMemory).fireAllRules() line: not available

StateExampleUsingSalience.main(String[]) line: 47

Variables

Breakpoints

Name Value

b State (id=1268)

changes PropertyChangeSupport (id=1297)

name "B"

state 1

StateExampleUsingSalience.drl

```
import org.drools.examples.State;  
  
rule "A to B"  
when  
    State(name == "A", state == State.FINISHED)  
    b : State(name == "B", state == State.NOTRUN)  
then  
    b.setState( State.FINISHED );  
    System.out.println(b.getName() + " finished");  
end  
  
rule "B to C"  
salience 10  
when  
    State(name == "B", state == State.FINISHED)  
    c : State(name == "C", state == State.NOTRUN)  
then  
    System.out.println(c.getName() + " finished");  
end
```

StateExampleUsingSalience.drl

Properties

Outline

org.drools.examples

A to B

B to C

B to D

C to D

Text Editor Rete Tree

Text Editor Rete Tree

Con... Sea... Global Data View Audit View Agenda View Working Memory View

StateExampleUsingSalience [Drools]

A finished

The selected working memory has no globals defined.

Object asserted (1): A[NOTRUN]

Activation created: Rule Bootstrap a=A[NOTRUN](1)

Object asserted (2): B[NOTRUN]

Object asserted (3): C[NOTRUN]

Object asserted (4): D[NOTRUN]

Activation executed: Rule Bootstrap a=A[NOTRUN](1)

Object modified (1): A[FINISHED]

Activation created: Rule A to B b=B[NOTRUN](2)

Activation executed: Rule A to B b=B[NOTRUN](2)

Object modified (2): B[FINISHED]

Activation created: Rule B to C c=C[NOTRUN](3)

Activation created: Rule B to D d=D[NOTRUN](4)

Activation executed: Rule B to C c=C[NOTRUN](3)

Object modified (3): C[FINISHED]

Activation executed: Rule B to D d=D[NOTRUN](4)

Object modified (4): D[FINISHED]

MAIN[focus]=AgendaGroupImpl (id=1259)

[0]=AgendaItem (id=1262)

ruleName="B to C"

c=State (id=1269)

[1]=AgendaItem (id=1263)

ruleName="B to D"

d=State (id=1270)

[0]=State (id=1268)

[1]=State (id=1269)

FINISHED=1

NOTRUN=0

changes=PropertyChangeSupport (id=1294)

name="C"

state=0

[2]=State (id=1270)

[3]=State (id=1271)

Writable Insert 21 : 1

Complex Event Processing

Event

A record of state change in the application domain at a particular point in time

Complex Event

An abstraction of other events called its members

Complex Event Processing

Processing multiple events with the goal of identifying the meaningful events within the event cloud

Drools Fusion



- Drools modules for Complex Event Processing
- Understand and handle events as a first class platform citizen (actually special type of Fact)
- Select a set of interesting events in a **cloud** or **stream** of events
- Detect the relevant relationship (patterns) among these events
- Take appropriate actions based on the patterns detected

Events as Facts in Time

Temporal
relationships
between events

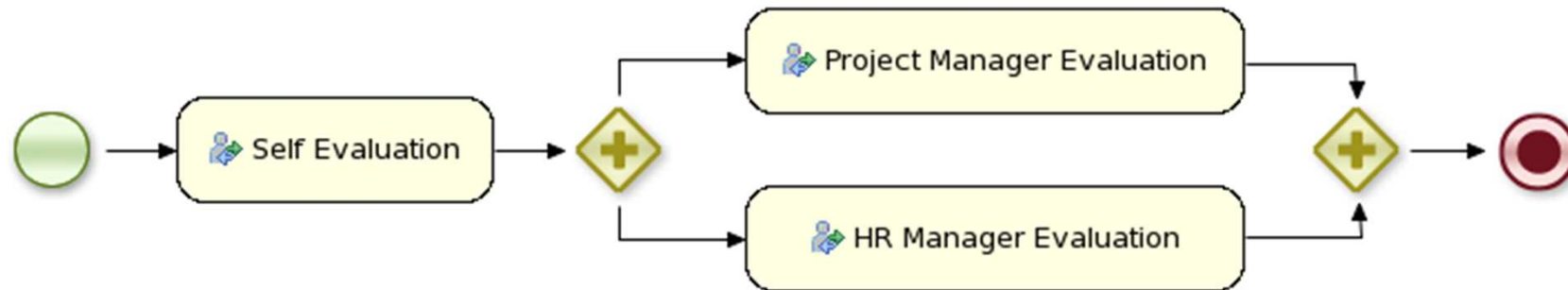
	Point-Point	Point-Interval	Interval-Interval
A before B			
A meets B			
A overlaps B			
A finishes B			
A includes B			
A starts B			
A coincides B			

```

rule
    "Sound the alarm"
when
    $f : FireDetected( )
    not( SprinklerActivated( this after[0s,10s] $f ) )
then
    // sound the alarm
end
    
```

Workflows as Business Processes

A ***workflow*** is a process that describes the order in which a series of steps need to be executed, using a flow chart.



Drools Flow (aka jBPM5)



- Allows to model business processes
- Eclipse-based editor to support workflows graphical creation
- Pluggable persistence and transaction based on JPA / JTA
- Based on BPMN 2.0 specification
- Can be used with Drools to model your business logic as combination of processes, events and rules

Drools Guvnor



- Centralised repository for Drools Knowledge Bases
- Web based GUIs
- ACL for rules and other artifacts
- Version management
- Integrated testing

Navigate Guvnor

Assets

Packages

Create New ▾

- Packages
 - Payments
 - defaultPackage
 - merchant
 - mortgages
 - Business rule assets**
 - Technical rule assets
 - (x)= Functions
 - DSL configurations
 - Model
 - Rule Flows
 - Enumerations
 - Test Scenarios
 - XML, Properties
 - Other assets, documentation
- test1

Find ✕ **Business rule as** ✕

Showing #10 of 10 items. [\[refresh list\]](#) [\[open selected\]](#)

Name	Last modified	Status	Categories
Underage	Dec 19, 2008	Draft	Eligibility rules
Bankruptcy history	Oct 1, 2008	Draft	Eligibility rules
No bad credit checks	Oct 1, 2008	Draft	Eligibility rules
no NINJAs No ninjas !	Oct 2, 2008	Draft	Eligibility rules
Pricing loans	Jan 27, 2009	Draft	Pricing rules
CreditApproval	Oct 22, 2008	Draft	Eligibility rules
DateDslRule	Oct 24, 2008	Draft	Technical
CheckBoxDslRule	Oct 23, 2008	Draft	Technical
RegexDslRule	Oct 23, 2008	Draft	Technical
wee	Jan 27, 2009	Draft	Home Mortgage

Drools Planner



- Works on all kinds of planning problems
 - NP-complete
 - Hard & soft constraints
 - Huge search space
- Planning constraints can be weighted and are written as declarative score rules
- The planner algorithm is configurable. It searches through the solutions within a given amount of time and returns the best solution found

Bin packaging

Place each item on a location in a container.

$$3 \times 3 = 9$$

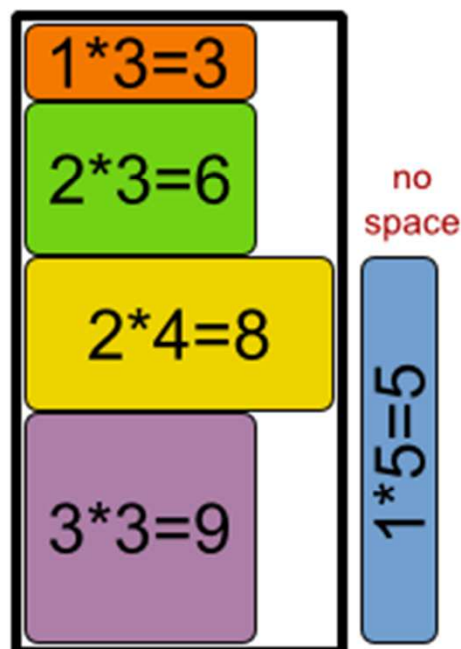
$$2 \times 4 = 8$$

$$2 \times 3 = 6$$

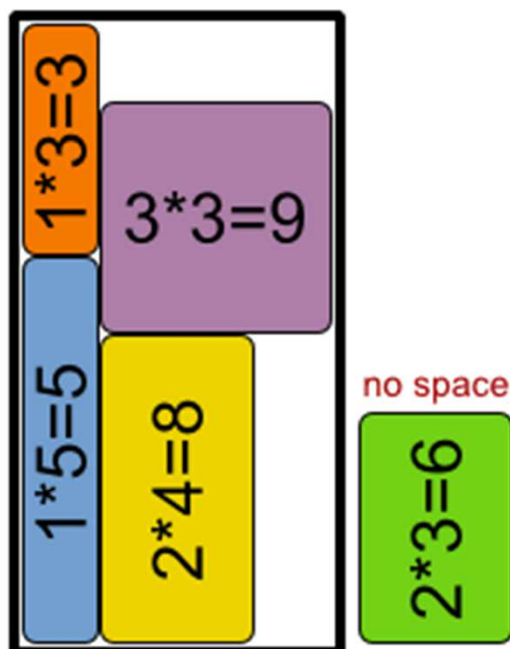
$$1 \times 5 = 5$$

$$1 \times 3 = 3$$

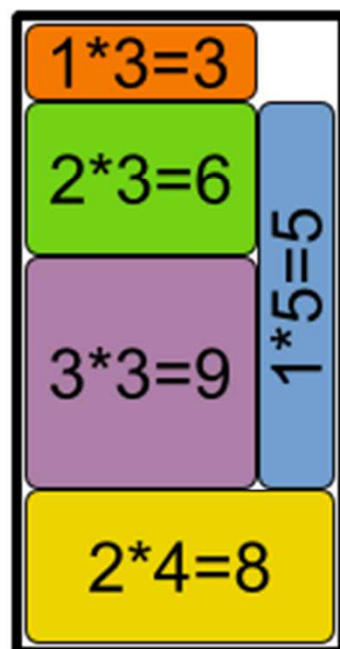
Largest size
first



Largest side
first



Drools
Planner





redhat.



Q



A

Mario Fusco
Red Hat – Senior Software Engineer

mario.fusco@gmail.com
twitter: @mariofusco