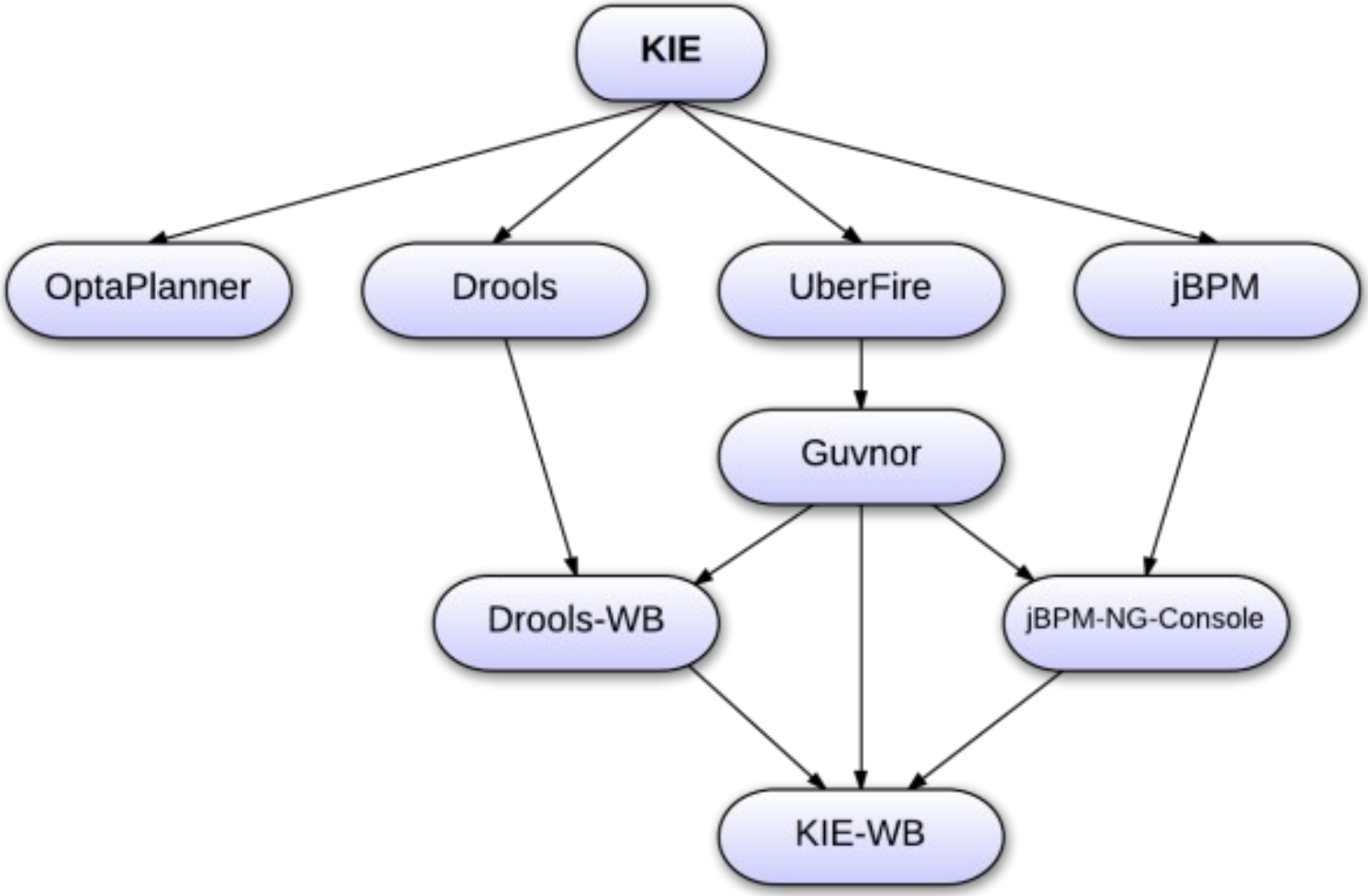
Introducing

by Mario Fusco
Red Hat – Senior Software Engineer
mfusco@redhat.com

redhat. | JBoss

KIE – Knowledge Is Everything



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

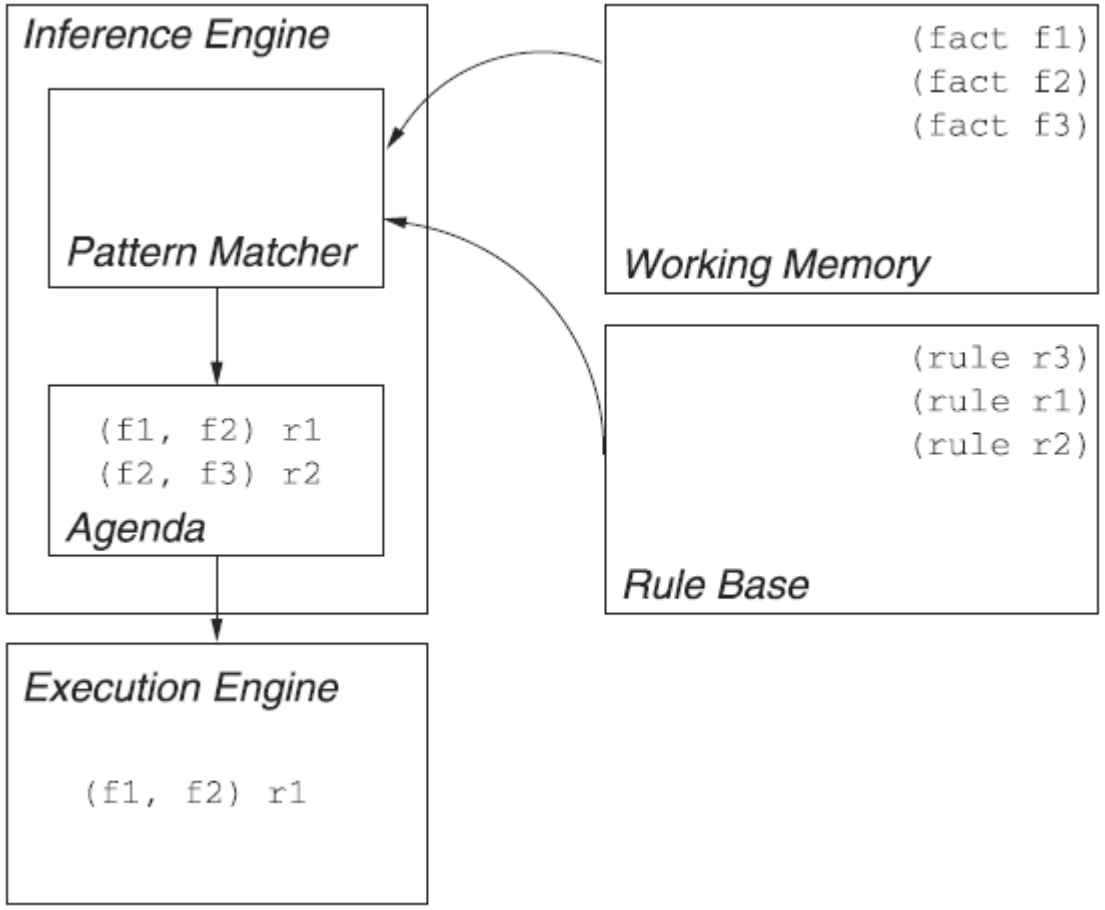
Advantages of Declarative Programming

- **Easier to understand →** It is more likely for a technically skilled business analyst to verify, validate or even change a rule than a piece of Java code
- **Improved maintainability →** We don't care about **how** to implement a solution only **what** needs to be done to solve a problem
- **Deals with evolving complexity →** It's easier to modify a rule than a Java program and to determine the impact of this change on the rest of the application
- **Modularity** → Each rule models an isolated and small portion of your business logic and is not part of a monolithic program
- **Requirements can be more naturally translated into rules**
- **Clear separation of business logic from the rest of the system**

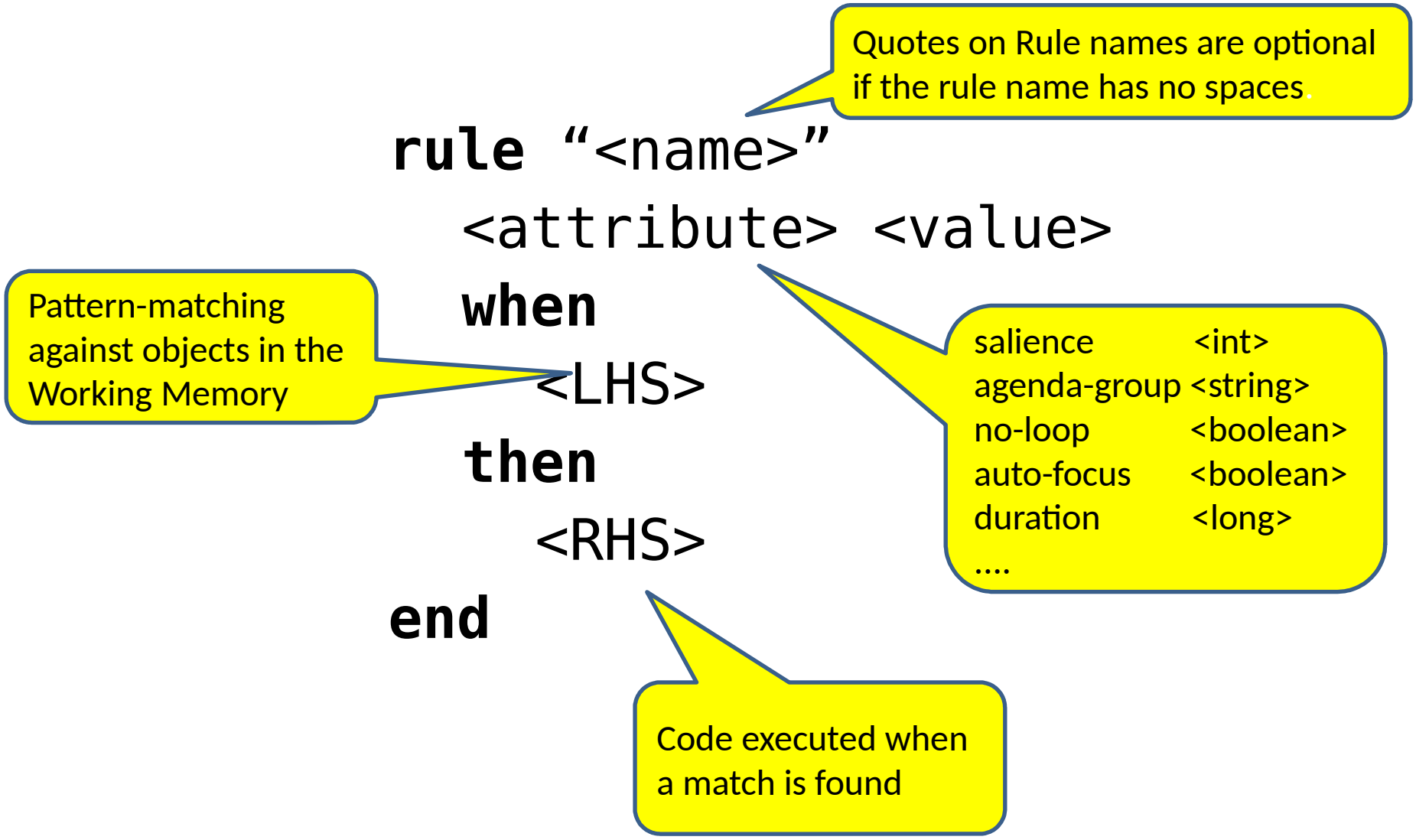
When should you use a Rule Engine?

- 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



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” );
   }
}
```

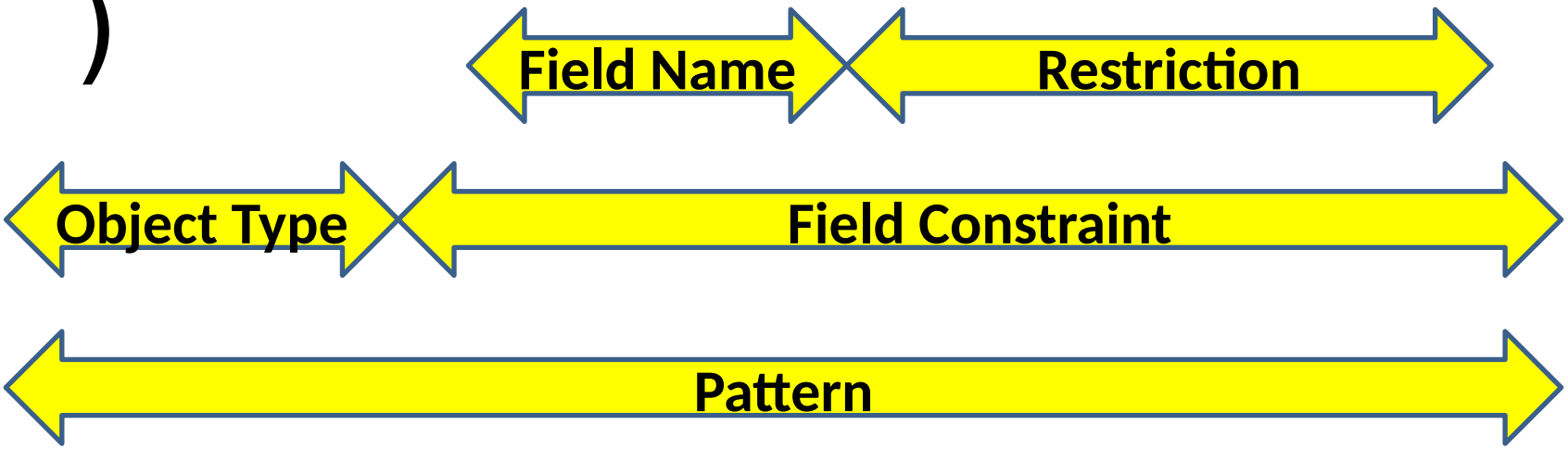
Rules can never be called directly

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
// 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
```

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

```
rule R3
   calendars "weekday"
   timer (int:0 1h)
when
    Alarm()
then
    sendEmail( ”Alert!” )
```

Execute now and after
1 hour duration only during weekday

```
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

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

Coupled logic

What happens when the child becomes adult?

Truth Maintenance System (2)

```
rule "Who Is Child" when
    $p : Person( age < 18 )
then
    logicalInsert( new IsChild( $p ) )
end

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

De-couples the logic

Encapsulate knowledge providing semantic abstractions for this encapsulation

Maintains the truth by automatically retracting

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 CEP

- 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

Cloud vs. Stream Mode

Cloud Mode (default)

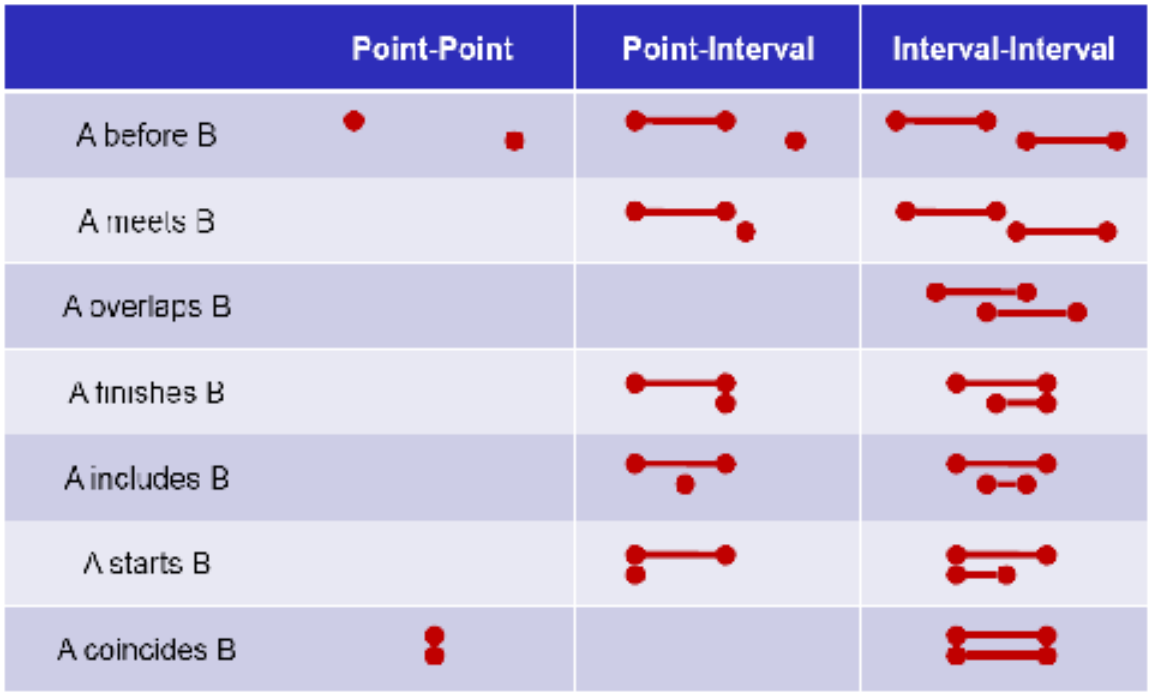
- **No notion of time**
- No requirement on event ordering
- Since they are based on the concept of “now” it is not possible to use sliding windows
- Not possible to determine when events can no longer match, so the application must **explicitly retract events** when they are no longer necessary

Stream Mode

- Events in each stream must be **time-ordered**
- The engine will force synchronization between streams through the use of the session clock
- **Sliding Window** support
- Automatic Event Lifecycle Management
- Automatic Rule Delaying when using Negative Patterns

Events as Facts in Time

Temporal relationships between events




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

Drools hidden gems:

Property Reactivity
Backward chaining
&
(Conditional) Named Consequences

Introducing the loop problem

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

Introducing the loop problem

```
rule "Salary award for min 2 years service"
no-loop
when
    e : Employee( lengthOfService > 2 )
then
    modify( e ) { setSalary( e.getSalary() * 1.05
) };
end
```

Introducing the loop problem

```
rule "Salary award for min 2 years service"
no-loop
when
    e : Employee( lengthOfService > 2 )
then
    modify( e ) { setSalary( e.getSalary() * 1.05
) };
end
```

```
rule "Salary award for team leaders" no-loop
when
    e : Employee( role == Role.TEAM_LEADER )
then
    modify( e ) { setSalary( e.getSalary() * 1.05
) };
```

Introducing the loop problem

```
rule "Salary award for min 2 years service"
when
     e : Employee( lengthOfService > 2 )
     not SalaryMin2Years( employee == e )
then
    modify( e ) { setSalary( e.getSalary() * 1.05
) };
    insert ( new SalaryMin2Years(e) );
end


rule "Salary award for team leaders"
when
    e : Employee( role == Role.TEAM_LEADER )
    not SalaryTeamLeader( employee == e )
then
    modify( e ) { setSalary( e.getSalary() * 1.05
) };
```

FRUSTRATION

sometimes a punch to the face is needed to get someones attention

Adding property reactvity

- Annotate the class:
  - Java:

```
@PropertyReactive
public class Employee {
    int salary;
    int lengthOfService;
    // … getters/setters
}
```

  - DRL:

```
declare Employee
    @PropertyReactive

    salary : int
    lengthOfService :
int
}
```

Problem solved!

```
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( role == Role.TEAM_LEADER )
then
    modify( e ) { setSalary( e.getSalary() * 1.05
) };
end
```

(un)Watching specific properties

```
rule "Salary award for min 2 years service"
when
    e : Employee( salary < 1000,
                  lengthOfService > 2 ) @Watch( !
salary )
then
    modify( e ) { setSalary( e.getSalary() *
1.05 ) };
end
```

Forward & Backward chaining

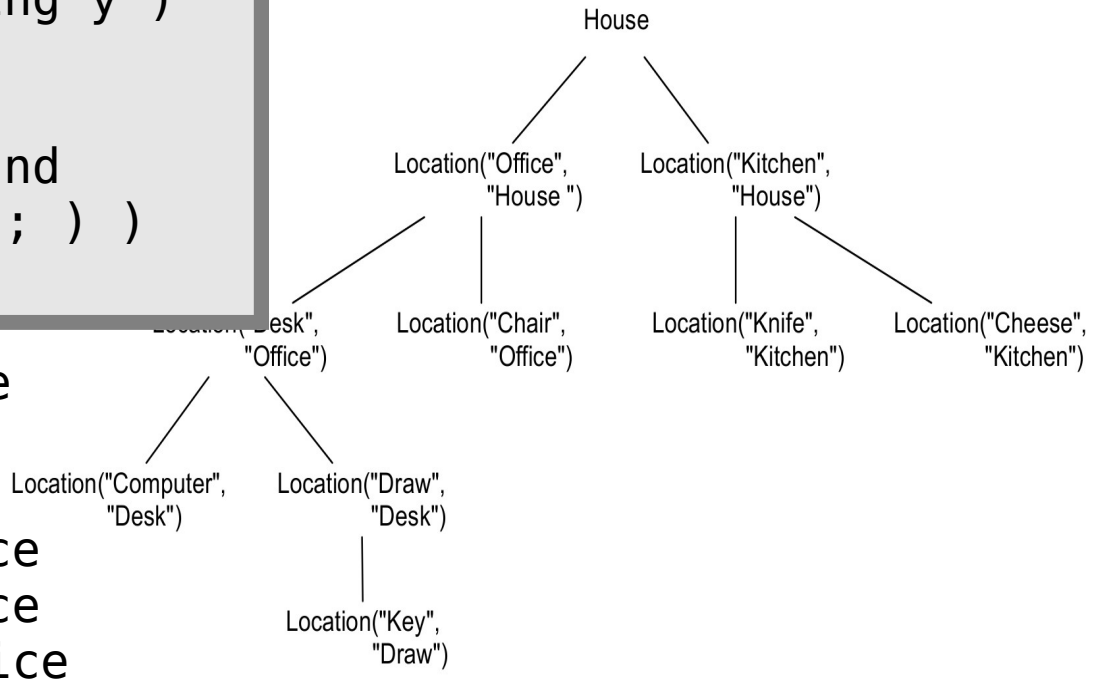
- Forward Chaining **starts with facts**/data and trigger actions or output conclusions
- Backward Chaining **starts with goals** and search how to satisfy them (e.g. Prolog)
- Drools is a **Hybrid Chaining Systems** meaning that it allows to mix these 2 strategies
- Backward-Chaining is often referred to as derivation queries and then Drools implements with the **query** construct
- A query is **a simple way to search the working memory** for facts that match the stated conditions
- A query is just **a rule with no consequence**. It collects all the results and returns them to the caller

Backward chaining example

```
rule “Print all things contained in the Office”
when
    isContainedIn(thing, "Office"; )
then
    System.out.println( "thing " + thing +
                        " is in the Office" );
end
                        String y )
    Location( x, y; )
or
    ( Location( z, y; ) and
      isContainedIn( x, z; ) )
end
```

In Var (buond)

Out Var (unbuond)




```
thing Key is in the Office
thing Computer is in the
Office
thing Draw is in the Office
thing Desk is in the Office
thing Chair is in the Office
```

Why one consequence is not enough?

Sometimes the constraint of having one single consequence for each rule can be somewhat limiting and leads to verbose and difficult to be maintained repetitions

```
rule "Give 10% discount to customers older
than 60"
when
    $customer : Customer( age > 60 )
then
    modify($customer)
{ setDiscount( 0.1 ) };
end

rule "Give free parking to customers older
than 60"
when
    $customer : Customer( age > 60 )
    $car : Car ( owner == $customer )
then
```

Named consequences

```
rule "Give 10% discount and free parking to
customers older than 60"
when
    $customer : Customer( age > 60 )
    do[giveDiscount]
    $car : Car ( owner == $customer )
then
    modify($car) { setFreeParking( true ) };
then[giveDiscount]
    modify($customer) { addDiscount( 0.1 ) };
end
```

When the pattern matching evaluation reaches this point activate the named consequence and continue evaluation

Give 10% discount to a customer older than 60 regardless if he owns a car or not

Conditional named consequences

```
rule "Give free parking and 10% discount to over 60
        Golden customer and 5% to Silver or

when
    $customer : Customer( age > 60 )
    if ( type == "Golden" ) do[giveDiscount10]
    else if ( type == "Silver" ) break[giveDiscount5]
    $car : Car ( owner == $customer )

then
    modify($car) { setFreeParking( true ) };
then[giveDiscount10]
    modify($customer) { setDiscount( 0.1 ) }
then[giveDiscount5]
    modify($customer) { setDiscount( 0.0
end
```

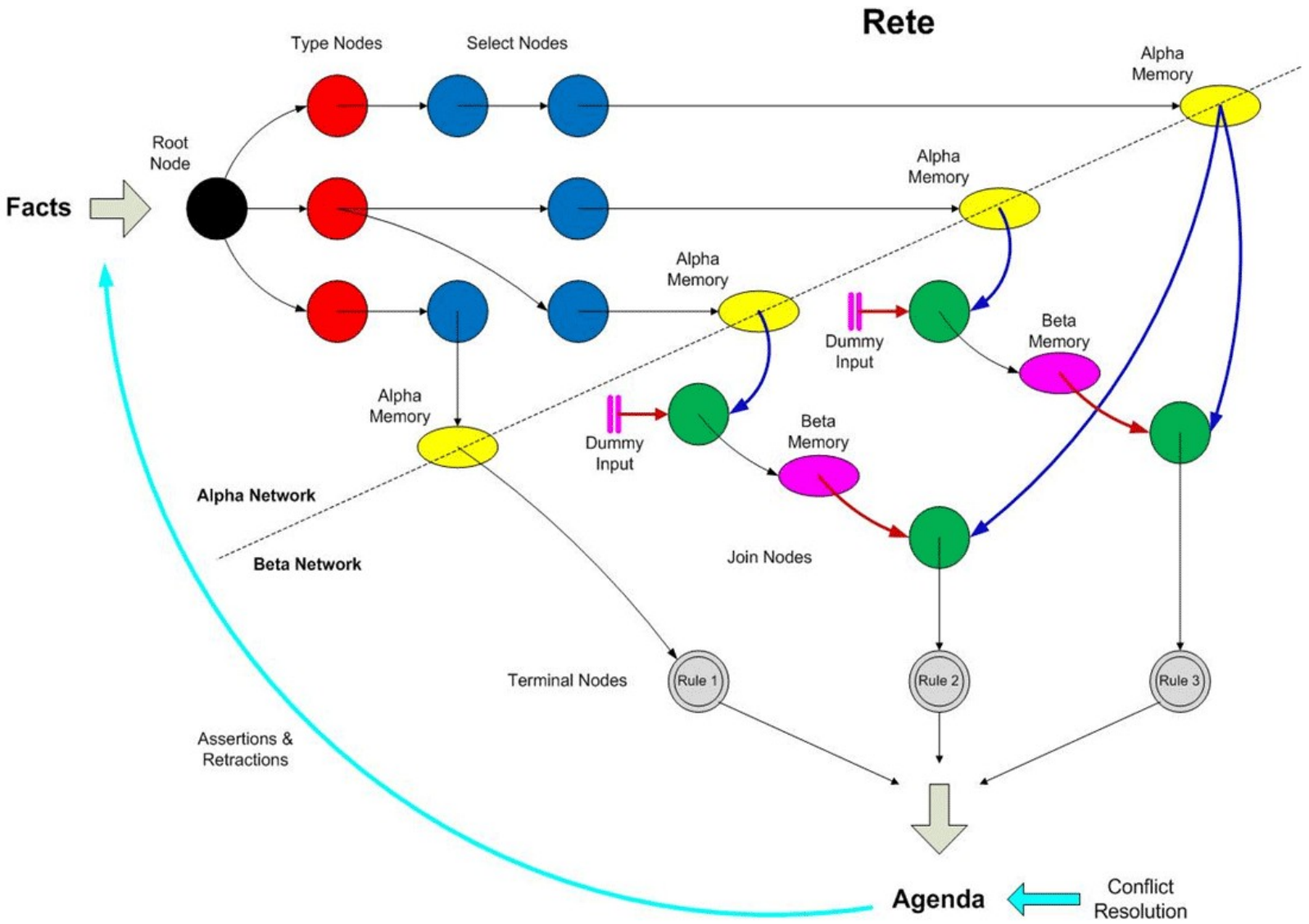
If the condition evaluates to true activate the named consequence and continue

Else If this other condition is met activate the named consequence and but block any further evaluation

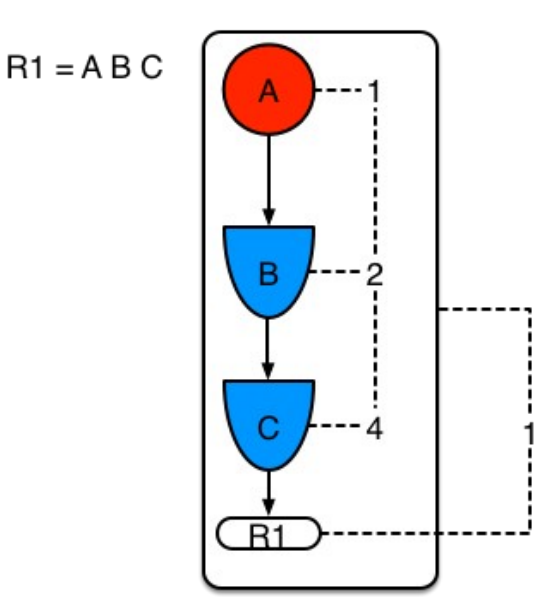
Innovations in Drools 6

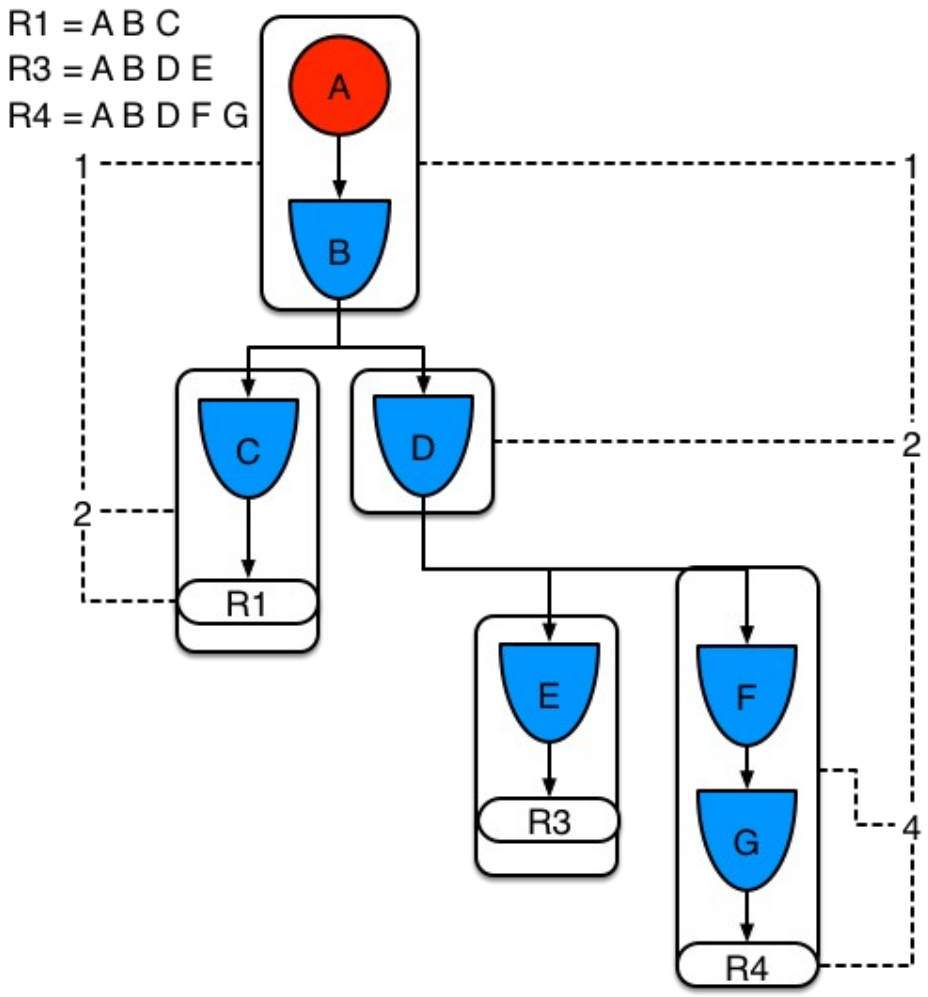
- A brand new engine: from ReteOO to **Phreak**
- From tuple based to set based propagation
- A git based repository …
- … combined with a maven based deployment model
- A simplified and mostly declarative API

From ReteOO ...

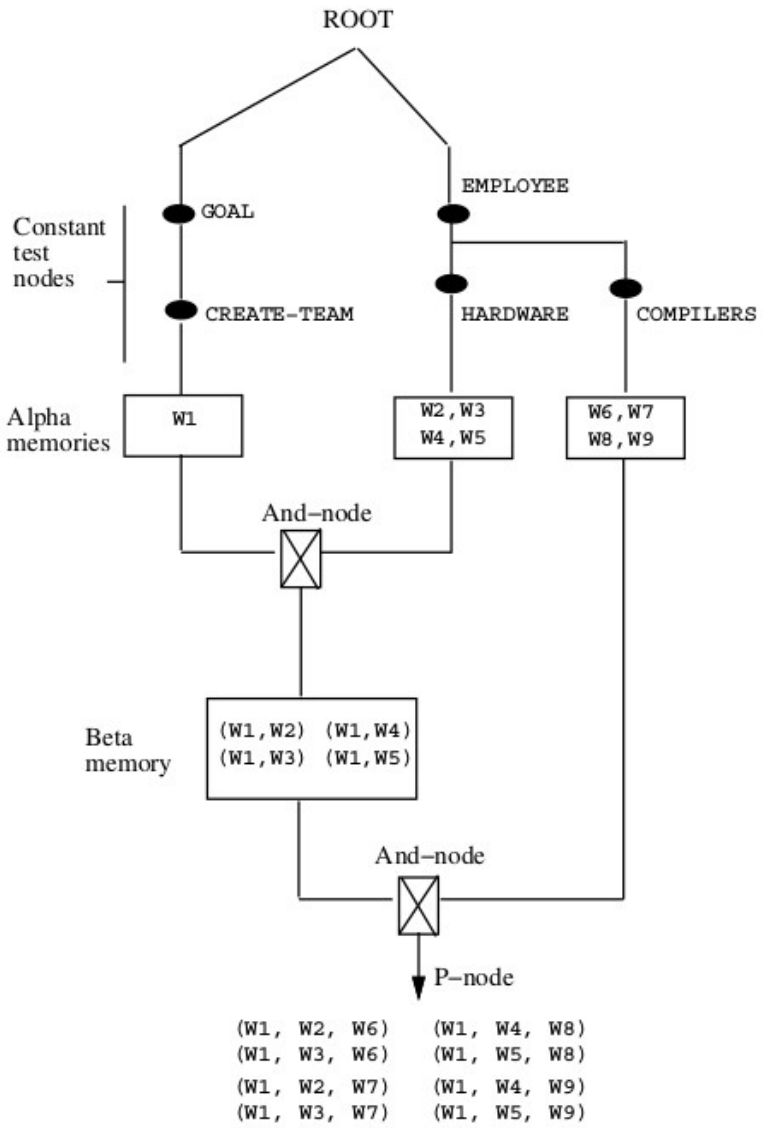

Rete
Type Nodes
Select Nodes
Alpha Memory
Root Node
Facts
Alpha Memory
Alpha Memory
Dummy Input
Beta Memory
Alpha Memory
Dummy Input
Beta Memory
Alpha Network
Beta Network
Join Nodes
Terminal Nodes
Rule 1
Rule 2
Rule 3
Assertions & Retractions
Agenda
Conflict Resolution

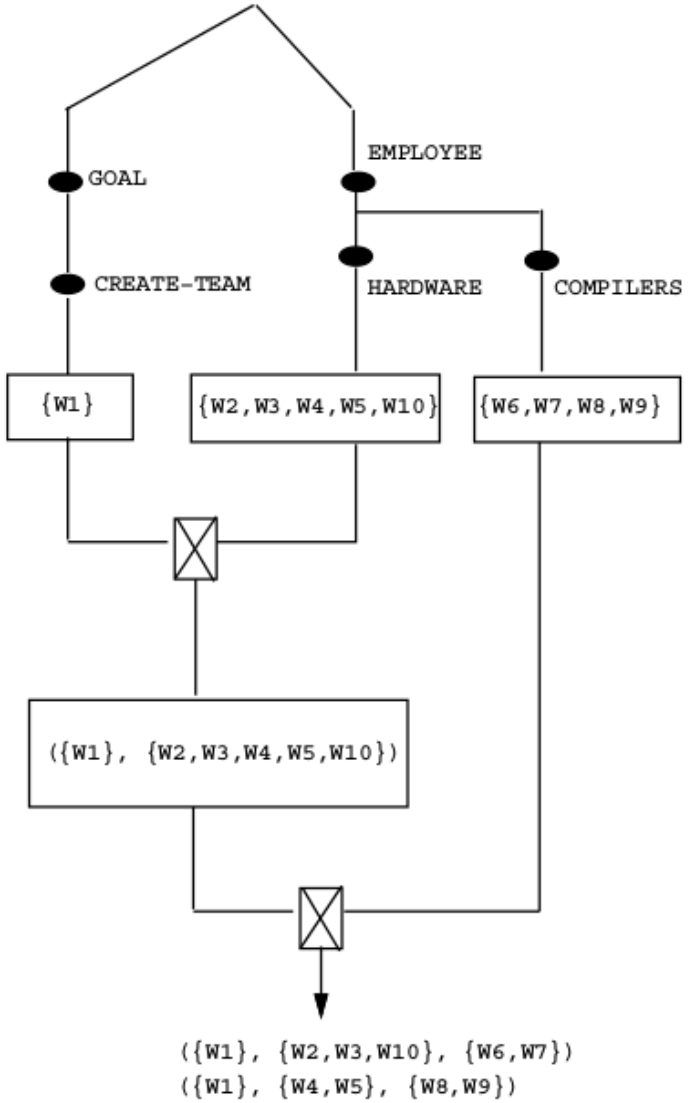
… to Phreak


R1 = A B C
A
B
C
R1
1
2
4
1


R1 = A B C
R3 = A B D E
R4 = A B D F G
A
B
C
D
R1
E
R3
F
G
R4
1
1
2
2
4

From tuple based to set based propagation





Advantages

- Preserves all ReteOO optimizations combining them with pros of other well known algorithms like Leaps, Collection Oriented Match, L/R Unlinking …
- On average 20% faster then ReteOO (and up to 400% faster on specific use cases)
- Reduced memory footprint
- More forgiving in presence of badly written rules

Keep innovating

Extending an Object-Oriented RETE Network with Fine-Grained Reactivity to Property Modifications

Mark Proctor[1,2], Mario Fusco[2], and Davide Sottara[3]

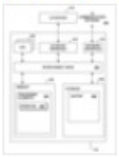
Dept. of Electrical & Electronic Engineering, Imperial College London, Londor
m.proctor13@imperial.ac.uk
JBoss, a Division of Red Hat Inc.
mfusco@redhat.com
Biomedical Informatics Dept., Arizona State University, Scottsdale (AZ)
davide.sottara@asu.edu

Building a Hybrid Reactive Rule Engine for Relational and Graph Reasoning

Mario Fusco[1(✉)], Davide Sottara[2(✉)], István Ráth[3], and Mark Proctor[1,4]

[1] A Division of Red Hat Inc., JBoss, Milan, Italy
mfusco@redhat.com
http://www.jboss.org
[2] Department of Biomedical Informatics, Arizona State University, Tempe, AZ, USA
davide.sottara@asu.edu
[3] Department of Measurement and Information Systems,
Budapest University of Technology and Economics, Budapest, Hungary
rath@mit.bme.hu
[4] Department of Electrical and Electronic Engineering,
Imperial College London, London, UK
m.proctor13@imperial.ac.uk

Compile-time grouping of tuples in a streaming application



www.google.it/patents/US20140095506
App. - Filed 21 Feb 2013 - Published 3 Apr 2014 - Michael J. Branson - International Business Machines Corporation
... Feb 17, 2011, **Mark Proctor**, Pattern behavior support in a **rule** engine ... 2014, **Red Hat**, Inc. Systems and Methods for Efficient Just-In-Time Compilation ...
Overview - Related - Discuss

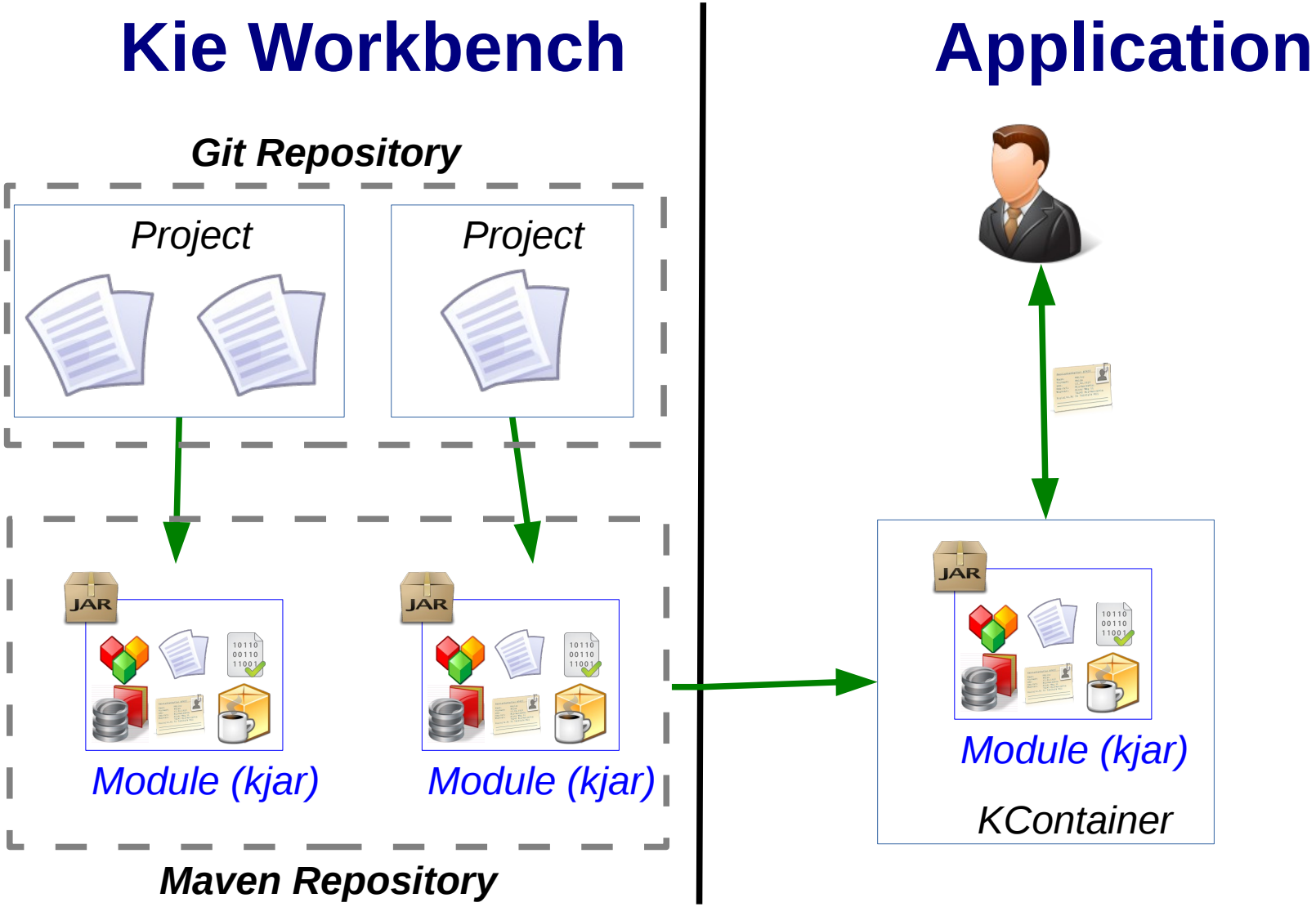
Property reactive modifications in a rete network

www.google.it/patents/US20140201124
App. - Filed 11 Jan 2013 - Published 17 Jul 2014 - Mar Proctor - Red Hat, Inc.
A processing device executing a Rete **rule** engine mo a particular property of an object ... Inventors, **Mark Pr** Mario Fusco. Original Assignee, **Red Hat** ...
Overview - Related - Discuss

Lazily enabled truth maintenance in rule engines

www.google.it/patents/US8538905
Grant - Filed 2 Dec 2010 - Issued 17 Sep 2013 - Mark Proctor - Red Hat, Inc.
Some embodiments of a method to lazily enable truth maintenance in a **rule** engine have been presented. ... 2007, Dec 4, 2008, **Mark Proctor**, Method and apparatus to define a ruleflow ... Owner name: **RED HAT**, INC., NORTH CAROLINA ...
Overview - Related - Discuss

A git/maven based workbench

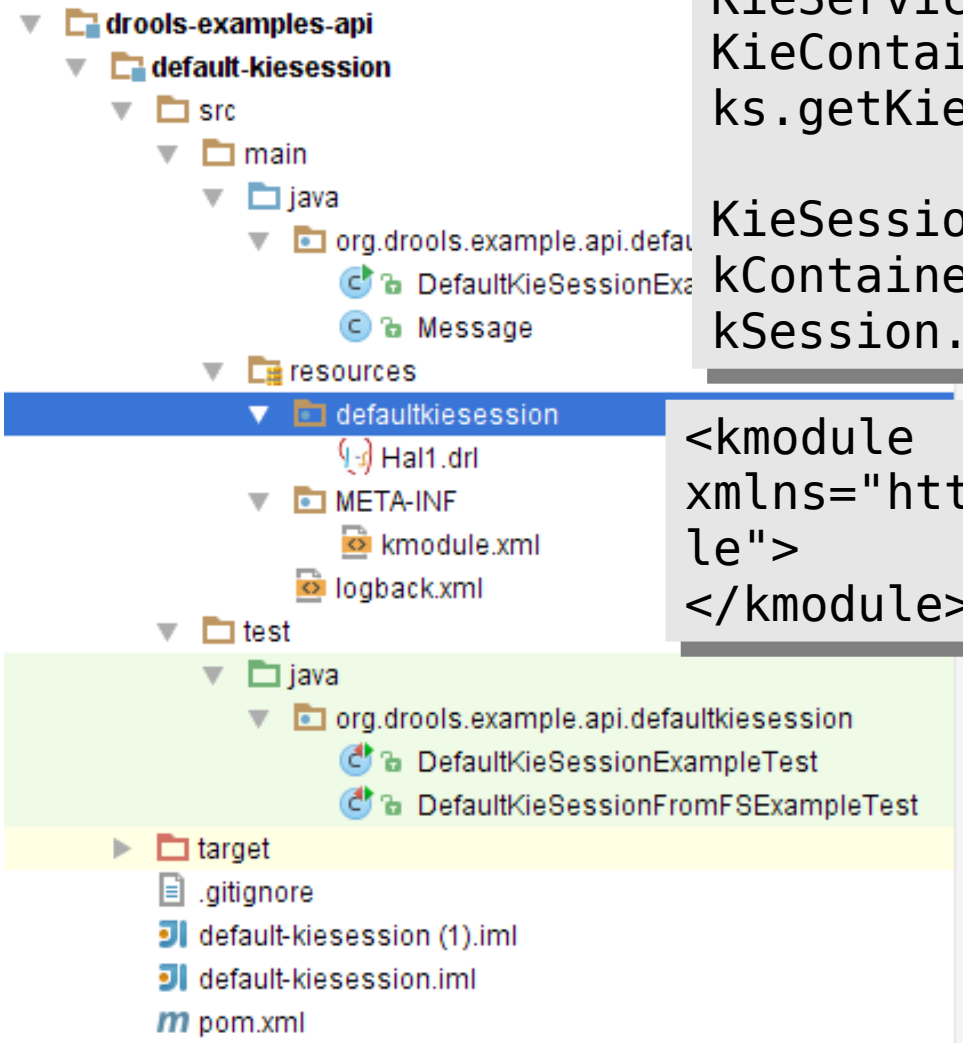


Defining Kbases and KSessions

```
<kmodule
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://jboss.org/kie/6.0.0/kmodule">

    <kbase name="ServerKB"
packages="org.myproject.example.server,

org.myproject.example.server.model"
            eventProcessingMode="stream"
equalsBehavior="identity">
        <ksession name="ServerKS" default="true" />
    </kbase>

    <kbase name="ClientKB"
packages="org.myproject.example.client">
        <ksession name="StatefulClientKS" type="stateful"/>
        <ksession name="StatelessClientKS"
```

```
KieContainer kc =
KieServices.Factory.get().getKieClasspathContainer();
KieSession serverKsession = kc.newKieSession( "ServerKS" );
KieSession clientKsession =
kc.newKieSession( "StatelessClientKS" );
```

Meaningful defaults



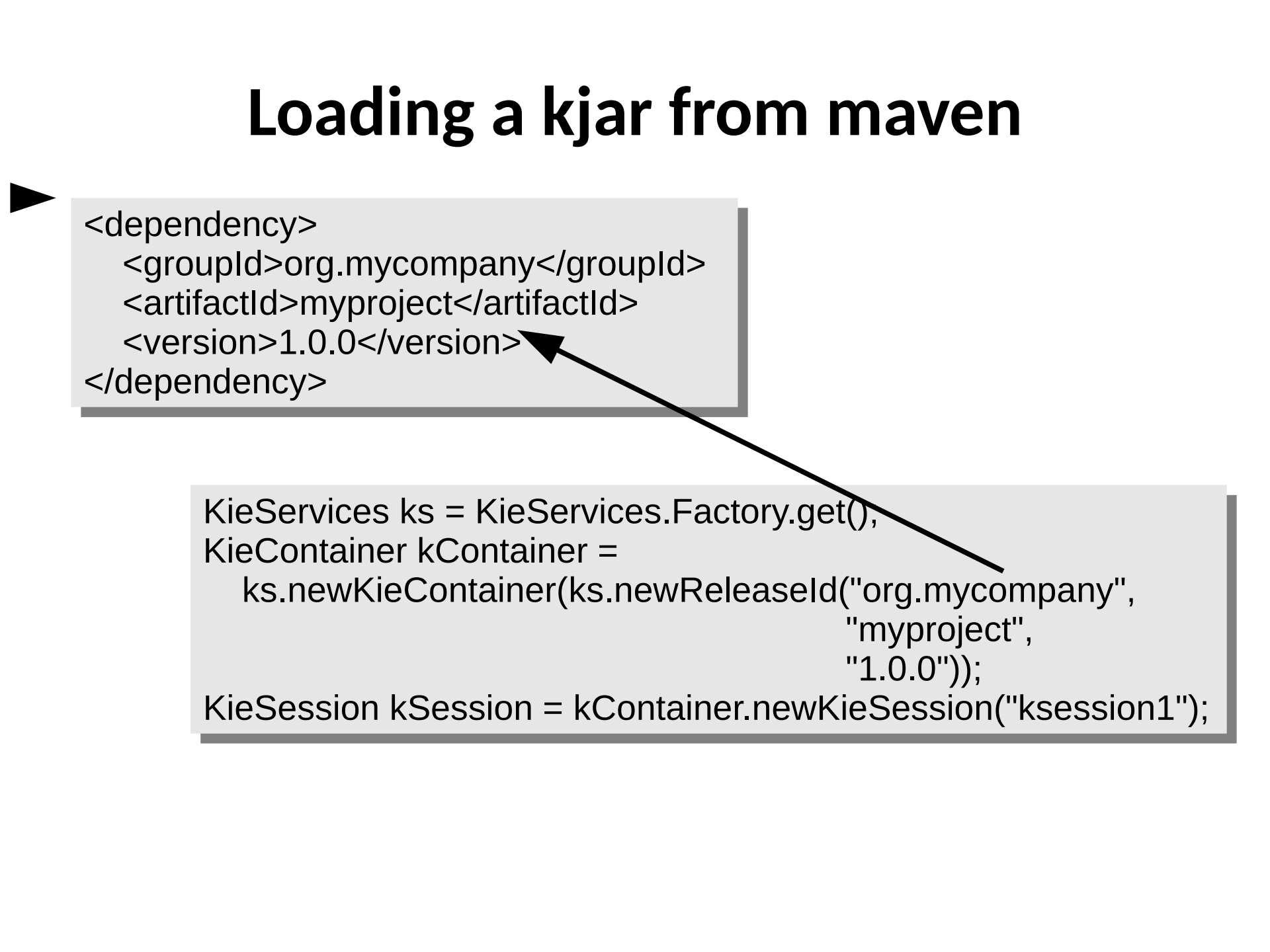

```
KieServices ks =
KieServices.Factory.get();
KieContainer kContainer =
ks.getKieClasspathContainer();

KieSession kSession =
kContainer.newKieSession();
kSession.fireAllRules();
```

```
<kmodule
xmlns="http://jboss.org/kie/6.0.0/kmodu
le">
</kmodule>
```

Loading a kjar from maven

```
<dependency>
    <groupId>org.mycompany</groupId>
    <artifactId>myproject</artifactId>
    <version>1.0.0</version>
</dependency>
```

```
KieServices ks = KieServices.Factory.get();
KieContainer kContainer =
    ks.newKieContainer(ks.newReleaseId("org.mycompany",
                                       "myproject",
                                       "1.0.0"));
KieSession kSession = kContainer.newKieSession("ksession1");
```



Future Directions

From DRL ...

```
rule "When there is a fire turn on the sprinkler"
when
   Fire($room : room)
   $sprinkler : Sprinkler( room == $room, on == false )
then
   modify( $sprinkler ) { setOn( true ) };
   System.out.println( "Turn on the sprinkler for room "
+
                       $room.getName() );
end

rule "When the fire is gone turn off the sprinkler"
when
   $room : Room( )
   $sprinkler : Sprinkler( room == $room, on == true )
   not Fire( room == $room )
then
   modify( $sprinkler ) { setOn( false ) };
   System.out.println( "Turn off the sprinkler for room
" +
```

… to Java 8 DSL …

```
Variable<Sprinkler> sprinkler = any(Sprinkler.class);
Variable<Fire> fire = any(Fire.class);

Rule r1 = rule("When there is a fire turn on the sprinkler")
.when( input(fire), input(sprinkler),
       expr(sprinkler, s -> !s.isOn()),
       expr(sprinkler, fire, (s, f) ->
s.getRoom().equals(f.getRoom()))
).then( on(sprinkler).execute(s -> {
        System.out.println("Turn on the sprinkler for room " +
                           s.getRoom().getName());
        s.setOn(true);
    }).update(sprinkler)
);

Rule r2 = rule("When the fire is gone turn off the sprinkler")
.when( input(fire), input(sprinkler),
       expr(sprinkler, Sprinkler::isOn),
       not(fire, sprinkler, (f, s) ->
f.getRoom().equals(s.getRoom()))
).then( on(sprinkler).execute(s -> {
```

… to POJO rules

```
public static class WhenThereIsAFireTurnOnTheSprinkler {
    Variable<Fire> fire = any(Fire.class);
    Variable<Sprinkler> sprinkler = any(Sprinkler.class);

    Consequence when = when(
       input(fire),
       input(sprinkler),
       expr(sprinkler, s -> !s.isOn()),
       expr(sprinkler, fire, (s, f) ->
s.getRoom().equals(f.getRoom()))
    );

    public void then(Drools drools, Sprinkler sprinkler) {
        System.out.println("Turn on the sprinkler for room " +
                           sprinkler.getRoom().getName());
        sprinkler.setOn(true);
        drools.update(sprinkler);
    }
}
```

Defining a Reactive Stream

```
public class TempServer {
    public static Observable<TempInfo> getFeed(String town) {
        return Observable.create(subscriber ->
            Observable.interval(1, TimeUnit.SECONDS)
                      .subscribe(i -> {
                          if (i > 5) subscriber.onCompleted();
                          try {

subscriber.onNext(TempInfo.fetch(town));
                          } catch (Exception e) {
                              subscriber.onError(e);
                          }
            }));
    }

    public static Observable<TempInfo> getFeeds(String... towns)
{
        return Observable.merge(Arrays.stream(towns)
                                      .map(TempServer::getFeed)
                                      .collect(toList()));
```

Reactive Streams integration

```
Variable<TempInfo> temp = any( TempInfo.class );
Variable<Person> person = any( Person.class );

Rule r1 = rule("low temp")
        .when(
                subscribe(temp, "tempFeed"),
                expr(temp, t -> t.getTemp() < 0),
                input(person, "persons"),
                expr(person, temp, (p, t) ->

p.getTown().equals(t.getTown()))
              )
        .then(on(person, temp)
                        .execute((p, t) -> System.out.println(
                                p.getName() + " is freezing, in " +
p.getTown()
                                + " - temp is " + t.getTemp())));

 DataStore persons = storeOf(new Person("Mark", 37,
"London"),
 "Toronto"),
                                new Person("Edson", 35,
                                new Person("Edson", 35,
 "Toronto"),
                                new Person("Mario", 40,
 "Milano"));
 bindDataSource(ksession, "persons", persons);
```

Introducing OOPath

```
  rule R when
      $student : Student( $plan: plan )
      $exam : Exam( course == "Big Data" ) from
  $plan.exams
      $grade : Grade( ) from $exam.grades
  then /* RHS */ end
```

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

March 3rd, 2016

10% Discount code

CERN@VDZ

Thanks … Questions?

Q

A

Mario Fusco
Red Hat – Senior Software Engineer
mario.fusco@gmail.com
twitter: @mariofusco