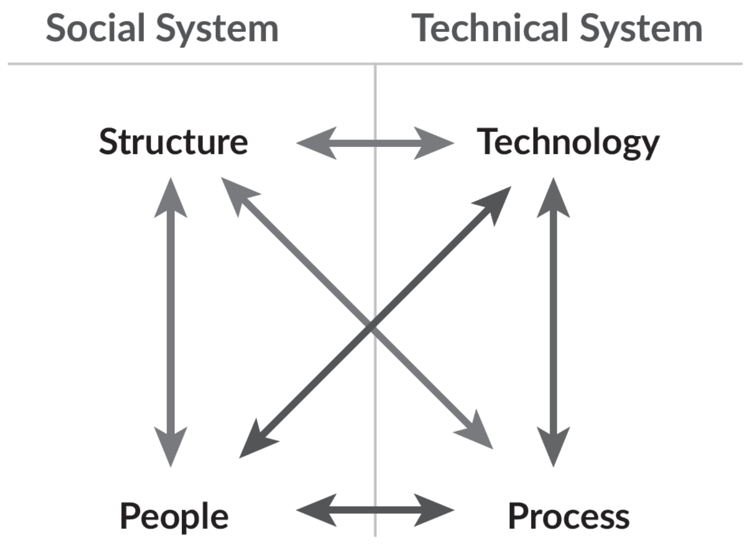
Low level Models – Business rules

Low level models

- Structure
- People
- Process
  - Process
  - Data
  - Business rules
- Technology



Concepts

- Business rule
- Policy
- Business logic

Business rule

- Statement that constrains some aspect of the business
  - Can only be true or false
  - Can apply to people, processes, IS

A student may register for a section of a course only if he/she has successfully completed the prerequisites for that course

A preferred customer qualifies for a 10 percent discount, unless he has an overdue account balance

Business rules

- Every business has business rules
  - More or less formalized
  - More or less automated
- Often, the business rule is visible to the end user / customer
- Often business rules apply to more than one business process
  - Ex. Only living persons can use services of a bank

Business logic

- Business rule encoded in computer language
  - In a 3 tiers architecture a business rule typically is encoded in the application tier
  - But can also be encoded in the presentation layer or data layer (db triggers)

Policy

- General direction
- Is implemented through several business rules

Only valid missions are allowed for POLITO employees.

Policy – business rules

Policy:
Only valid missions are allowed for POLITO employees.

BR:
1 The mission is accepted only if the available budget is higher than the presumed cost.

2 The daily expenses for meals cannot be higher than 80€.

3 For flights business class is forbidden.

A BR should be

Declarative	The statement of a policy, not how the policy is enforced.
Precise	The rule must have only one interpretation.
Atomic	A business rule marks one statement, not many.
Consistent	A business rule must be internally and externally (to other rules) consistent.
Expressible	A business rule must be able to be stated in natural language.
Distinct	Business rules mustn't be redundant.
Business-oriented	A business rule is stated in terms business people can understand.

BR - implementation

- Manual
- Automated
  - Embedded in computer programs
  - In declarative form, executed by an engine

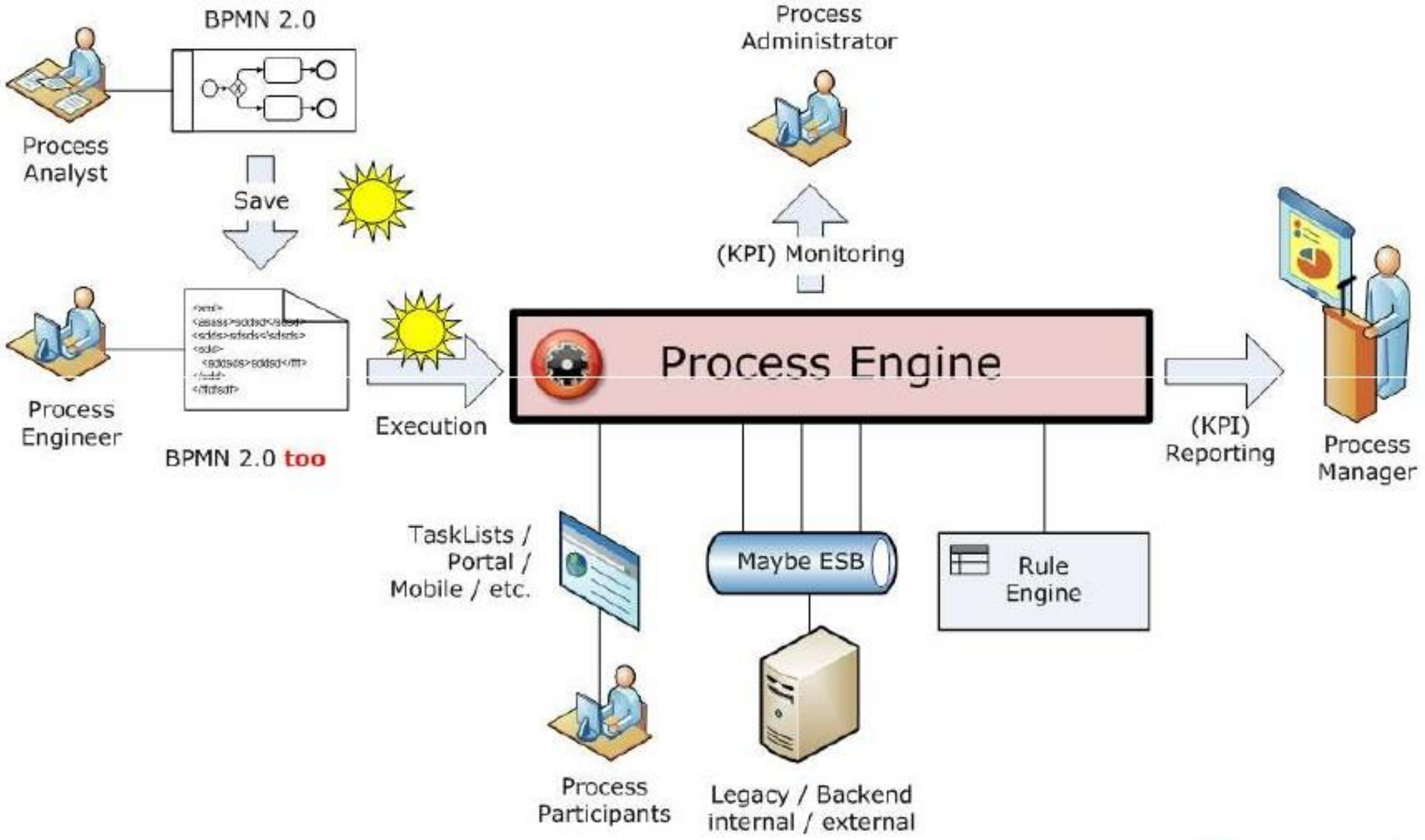
Declarative BR

- Advantages
  - Force formal definition of rules
  - Separate business rules and business processes (separation of concerns)
    - Different, most suitable tools
    - Independent evolution over time (BR can change without recompiling / redeploying programs)
    - Localization of rules (repository of rules)
    - Business people can write /check BRs
    - Business rule written once, enforced over all processes

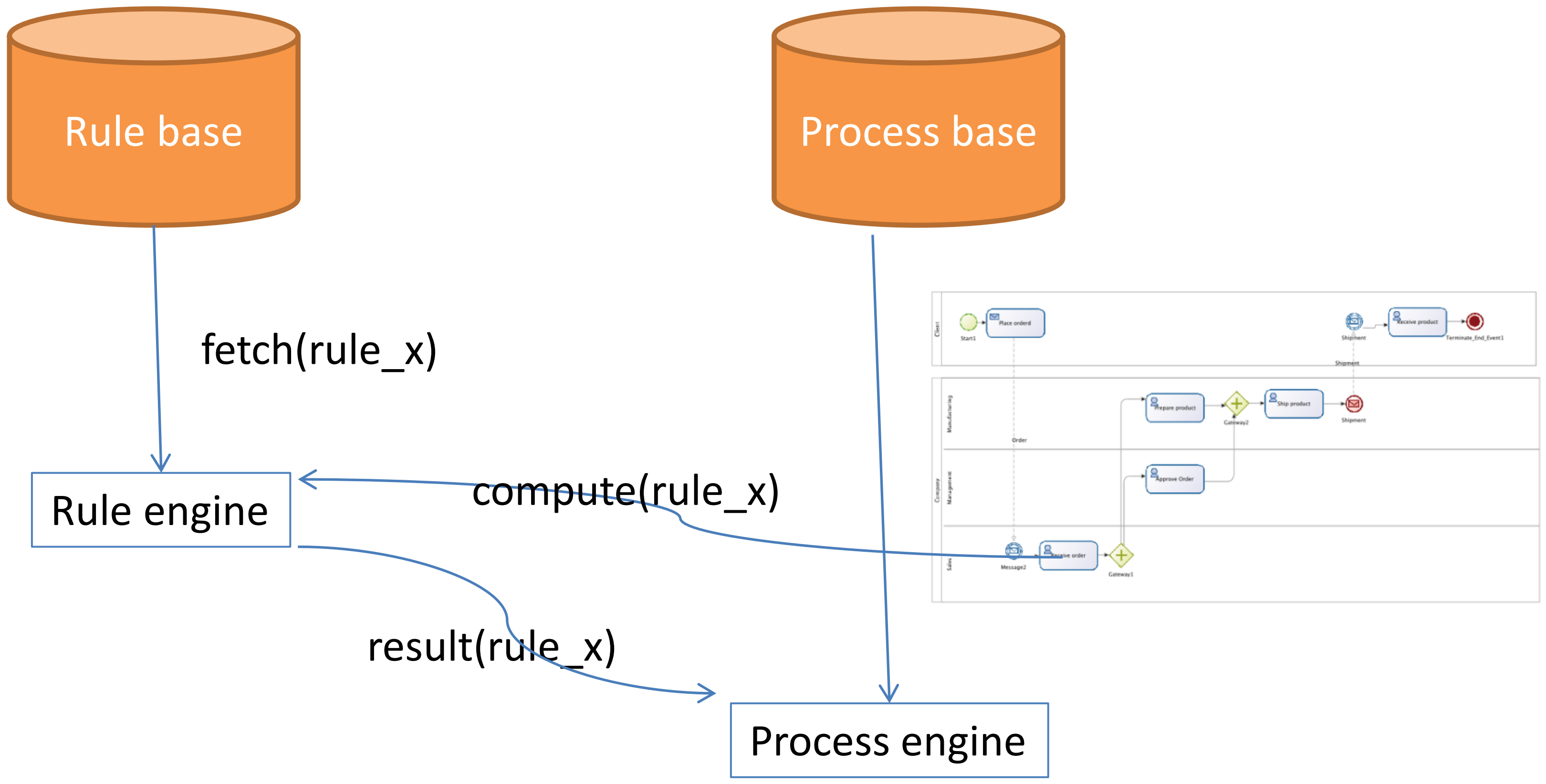
BR: decisions

Processes: activity flow, task assignment, deadlines and escalation, exception handling

Desired Architecture with BPMN 2.0



camunda
the business process company

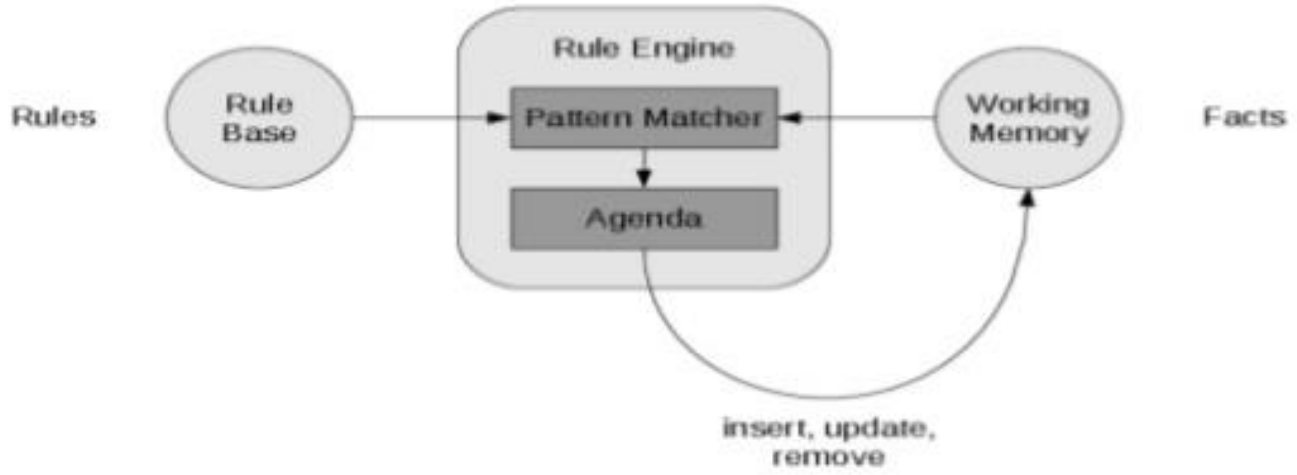

Rule base
Process base
fetch(rule_x)
Rule engine
compute(rule_x)
result(rule_x)
Process engine
Client
Start1
Place orderd
Shipment
Receive product
Terminate_End_Event1
Shipment
Company
Manufacturing
Prepare product
Gateway2
Ship product
Shipment
Order
Management
Approve Order
Sales
Message2
Receive order
Gateway1

Declarative rules and BRMS

Business Rules Management Systems

A **Business Rules Management System (BRMS)** is a software system that can be used to define, deploy, execute and monitor business rules.

The **Business Rules Engine** is the part of the BRMS in charge of executing the business rules.



BRMS

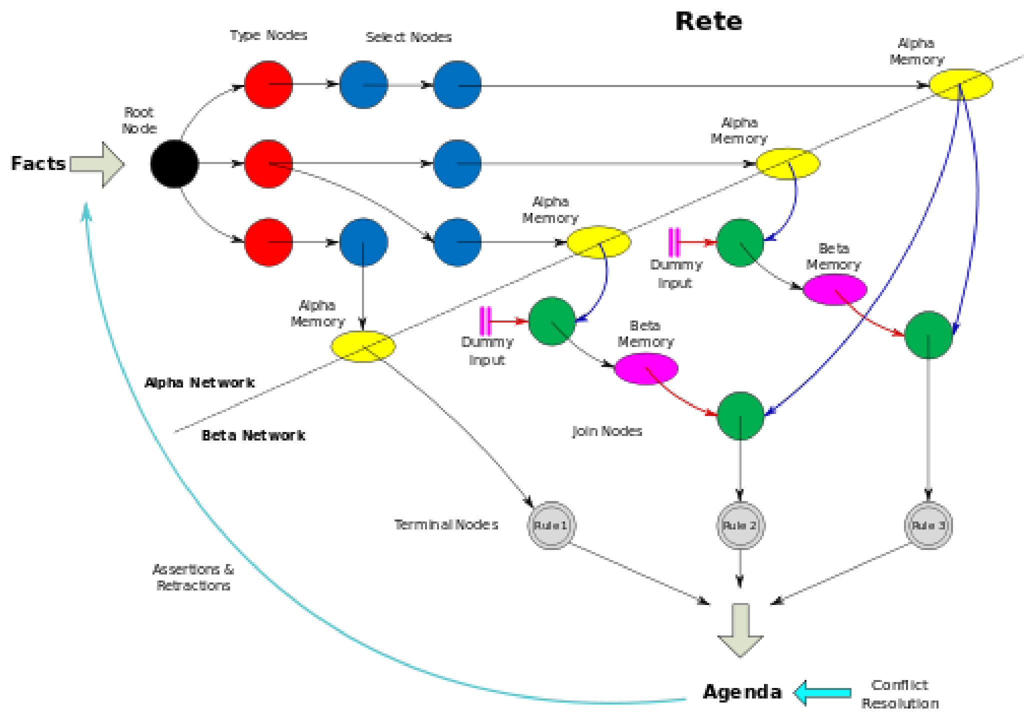
- Drools, inside Jboss (Red Hat)
- Camunda DMN, (Camunda)
- Oracle BR engine, inside Oracle DB suite (Oracle)
- Operational decision manager (IBM)

JBoss Drools

Drools is a suite of products that allow to manage Business Rules.

-**Guvnor**	Business Rule Management System;
-**Expert**	Declarative, rule based, coding environment;
- **Flow**	Workflow (or business) process management;
- **Fusion**	Event Processing engine;
- **Planner**	Automated planning.

The Rete Algorithm



Execution of rules in Drools is governed by the Rete algorithm, that describes how the Rules are processed to generate a discrimination network, to filter data as it propagates through the network.

In this implementation, that is way more efficient than typical formal chaining, rules are represented using directed acyclic graphs.

Drools Rule Language

A Drools Rule has two parts. When the **Condition** is verified, the **Action** is carried out.

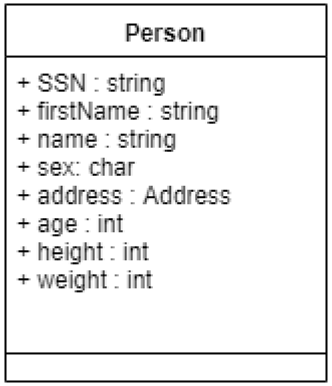
The then clause can also update, insert and delete elements in the working memory.

```
rule “rule-name”
when
     <Condition: query in an Object query
     language>                                LHS
then
     <Action: any Java code>                  RHS
end
```

Drools Rule Language: Patterns

Conditional element are composed by one or more **patterns,** which are typically composed with "and" (that is thus implicit).

A pattern matches against a fact of the given type.



`Person()` *matches all Person objects in the working memory*

Parenthesis define **constraints** for patterns:

`Person(age == 100)` *matches all Person objects having age = 100*

Drools Rule Language: Constraints

Constraints are essentially Java espressions, with some particular characteristics.

Property access is obtained through Java getters defined in classes; if default getters (getProperty() for property) are defined, properties can be accessed directly by their name.

```
Person( age == 50 ) // same as: Person( getAge() == 50 )
```

Any Java expression that returns a *boolean* can be used as a constraint inside the parentheses o a pattern. Java expression can be mixed with property accesses.

```
Person(Math.round( weight / ( height * height)) < 25.0 )
```

Drools Rule Language: Binding variables

For referring to the matched object, a **pattern binding variable** can be used:

```
rule ...
when
      $p: Person()
then
      system.out.println( “Person” + $p );
end
```

Drools Rule Language: Binding variables

A variable can be bound also to a property of an object, so that it can be referenced by other constraints.

The following example filters out two persons with the same address.

```
rule ...
when
      $p: Person( $address : address )
      $p2: Person( address == $address )
then
      system.out.println( "Person" + $p + $p2 );
end
```

Drools Rule Language: Operators

< <= > >=
These operators can be used on properties with natural ordering (also strings, dates).

```
Person( firstName < $otherFirstName )
```

matches / not matches
matches a field against any Java Regular Expression. Not matches returns true if the string does not match.

```
Person( name matches "(Rossi)?\\S*Verdi" )
```

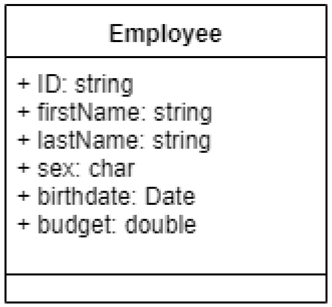
Drools Rule Language: Operators

contains / not contains
To check if a field that is a Java Collection contains a given value. Not contains returns true if the Collection does *not* contain the value.

```
EnrolledEmployees( employeesList contains “Rossi” )
```

memberOf / not memberOf
To check whether a field is a member of a Java Collection. Not memberOf returns true if the Collection does *not* contain the field.

```
Person( name memberOf $employeesList )
```



Drools Rule Language: Operators

in / not in

To compare a field with more than one possible values to match.

```
Person( name in ( “Rossi”, “Verdi”, “Bianchi” ) )
```

str

Used to check the length of a string field, and if the string starts or ends with a certain value.

```
Person( name str[startsWith] “R” )
Person( name str[endsWith] “I” )
Person( name str[length] 5 )
```

Drools Rule Language: Conditional Elements

and

To group Conditional Elements in logical conjunction. It is the default conditional elements. The following writings are equivalent.

```
Person( age > 60 ) and Person ( sex == “m” )
```

```
Person( age > 60 )
Person( sex == “m” )
```

or

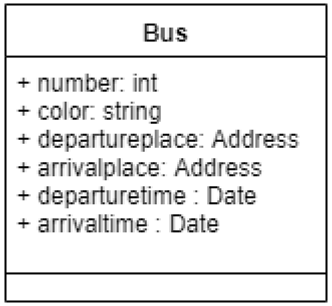
Used to group other Conditional Elements in logical disjunction.

```
Person( sex == "f", age > 60 ) or Person( sex == "m", age
> 65 )
```

Drools Rule Language: Conditional Elements

exists

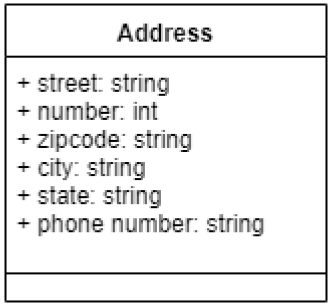
At least one matches the following query. It is different from the Pattern, which is like "for each one of...". If exists is used, the rule will only activate at most once, regardless of how much data there is in working memory that matches the condition.



```
exists ( Bus(color == "red", number == 42) )
```

from

Allows to specify an arbitrary source for data to be matched (not in Working Memory).

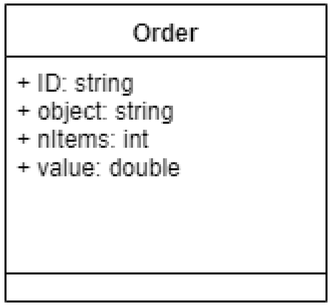


```
$p : Person( )
$a : Address( zipcode == "23920W") from $p.address
```

Drools Rule Language: Conditional Elements

accumulate

Accumulate iterates a rule over a collection of objects, on which it can execute custom operations, or a set of default accumulate functions: average, min, max, count, sum, collectList, collectset.

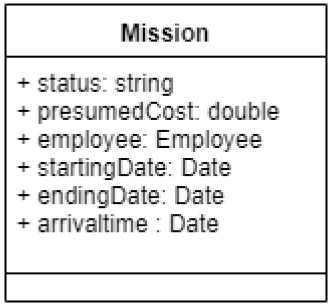


```
when
        $order : Order()
        $total : Number( doubleValue > 100 )
                from accumulate( OrderItem( order == $order, $value : value ),
                sum( $value ) )
then
        # apply discount to $order end
```

Drools Rule Language: Example

The mission is accepted only if the available budget for the employee is higher than the presumed cost.

```
rule "maxCost"
when
      $e : Employee( $b : budget )
      Mission( employee = "$e" and presumedCost > $b )
then
      m.reject();
end
```



Drools Rule Language: Example

Accept without checking all missions with presumed cost that is less than €100.

```
rule “minCost”
when
      $m : Mission(status == “pending” &&
      presumedCost < 100 )
then
      m.accept();
end
```

Drools Rule Language: Example

A mission is not accepted if the employee has already another accepted mission in that period.

```
rule “refuse mission”
when
      $m : Mission( $e : employee, $sd :
            startingDate, $ed : endingDate )
      exists ( Mission( employee == $e &&
            ( startingDate > $sd && startingDate <
            $ed ) || ( endingDate > $sd &&
            endingDate < $ed ) ) )
then
      $m.reject()
end
```

Domain-specific Languages (DSL)

DSL are written in natural language statements, and hence can be edited also by domain experts (as business analysts).

Rules in DSL are then translated into DLR rules.

[when] – cost is less than 5000

Decision Tables

Decision tables are a compact way of representing conditional logic. They are expressed with spreadsheets (.xls) or .csv files.

Each row in the spreadsheet is a rule, and they are used to generate DRL rules automatically.

Decision tables are well suited for business level rules.

Decision Tables: Example

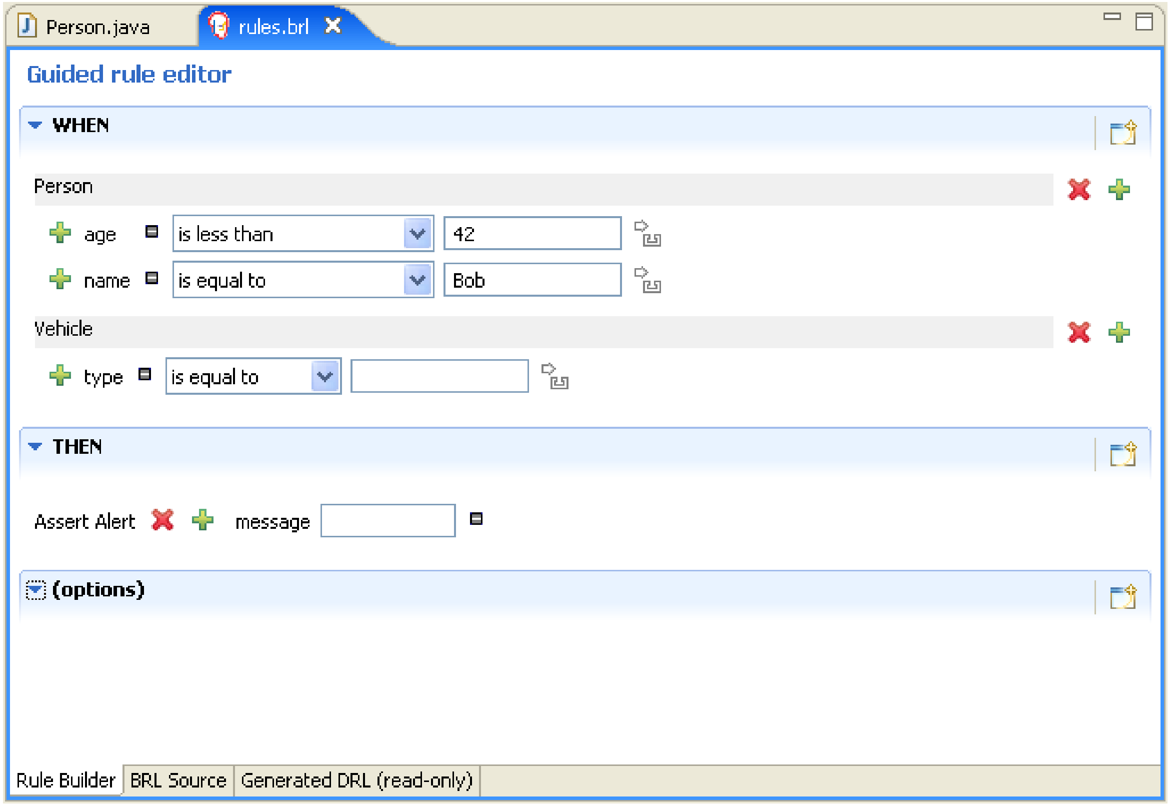
	Condition	Action	Action
	m: Mission		

	value $param	m.setMessage($param);
		update(m);
			m.setStatus($param);
			update(m);

Accepted Mission	<= 5000	Mission Accepted	Mission.ACCEPTED
Rejected Mission	> 5000	Mission out of budget	Mission.REJECTED

Guided Editor

Rule IDE (in Eclipse) for step-by-step definition of rules.


Person.java | rules.brl

Guided rule editor

WHEN

Person

age | is less than | 42

name | is equal to | Bob

Vehicle

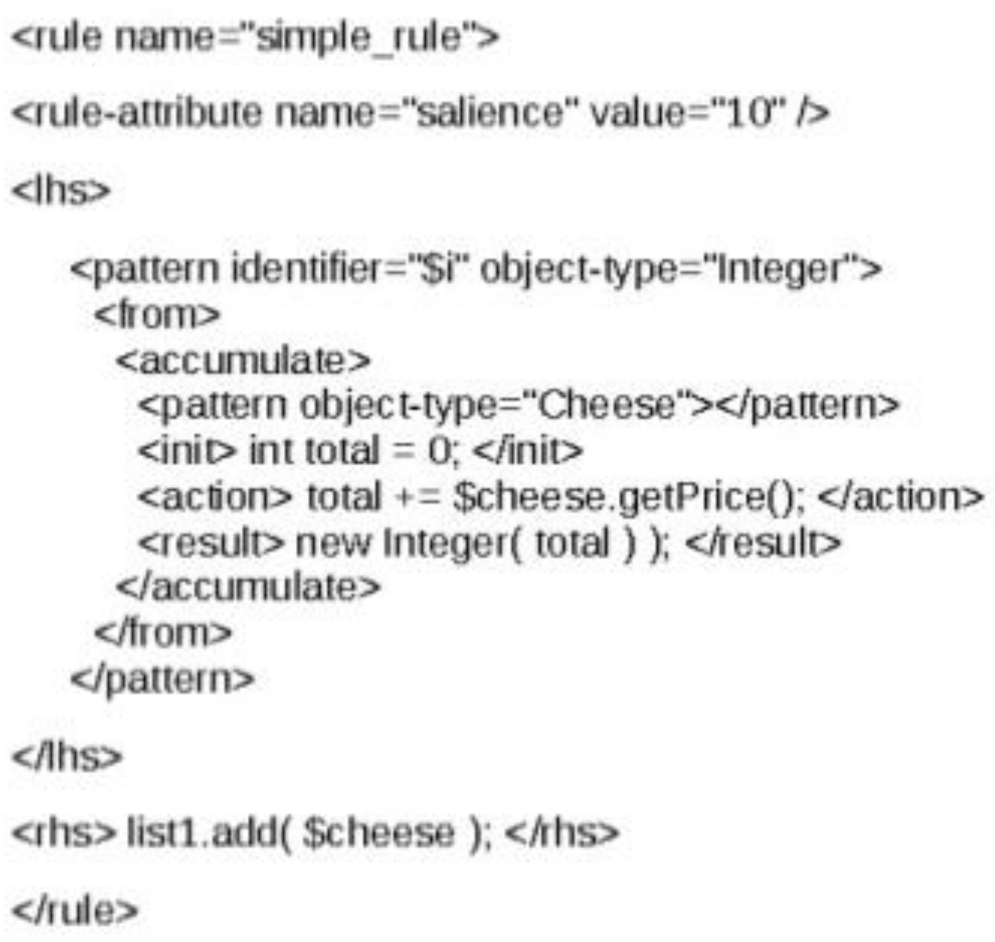
type | is equal to |

THEN

Assert Alert | message |

(options)

XML



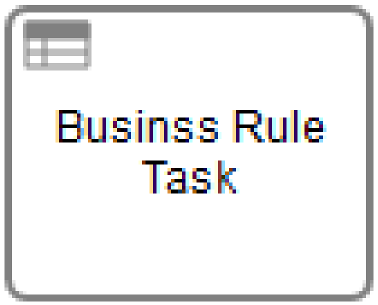
```
<rule name="simple_rule">

<rule-attribute name="salience" value="10" />

<lhs>

    <pattern identifier="$i" object-type="Integer">
      <from>
        <accumulate>
          <pattern object-type="Cheese"></pattern>
          <init> int total = 0; </init>
          <action> total += $cheese.getPrice(); </action>
          <result> new Integer( total ) ); </result>
        </accumulate>
      </from>
    </pattern>

</lhs>

<rhs> list1.add( $cheese ); </rhs>

</rule>
```

BUSINESS RULES IN CAMUNDA

SOftEng
http://softeng.polito.it

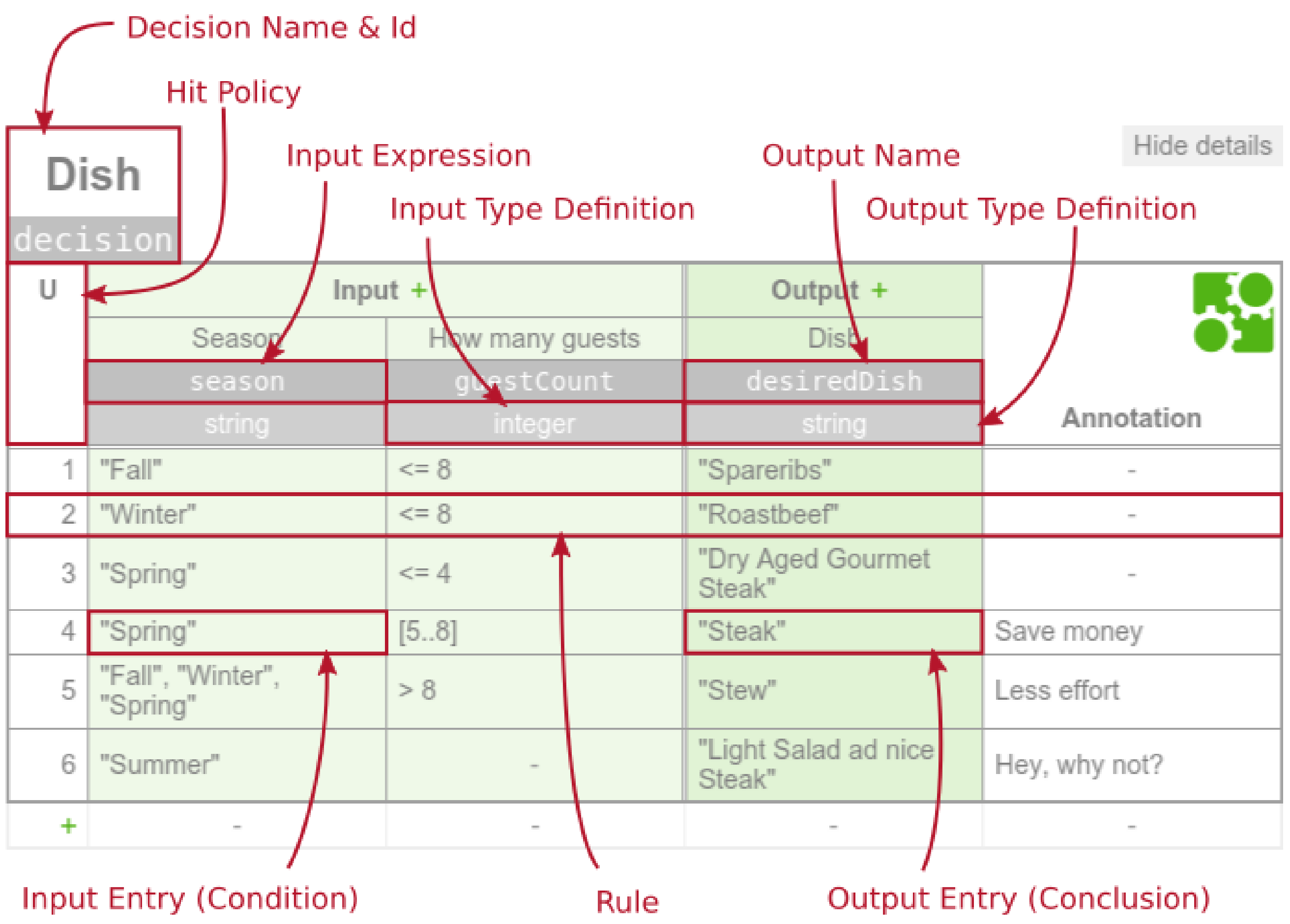
Business Rules in Camunda

In Camunda BPM Business Rules are executed synchronously by a Business Rule Task.



Custom external Rule engines can be used writing the relative Java Code as in a normal service Task.

Camunda: Decision Tables

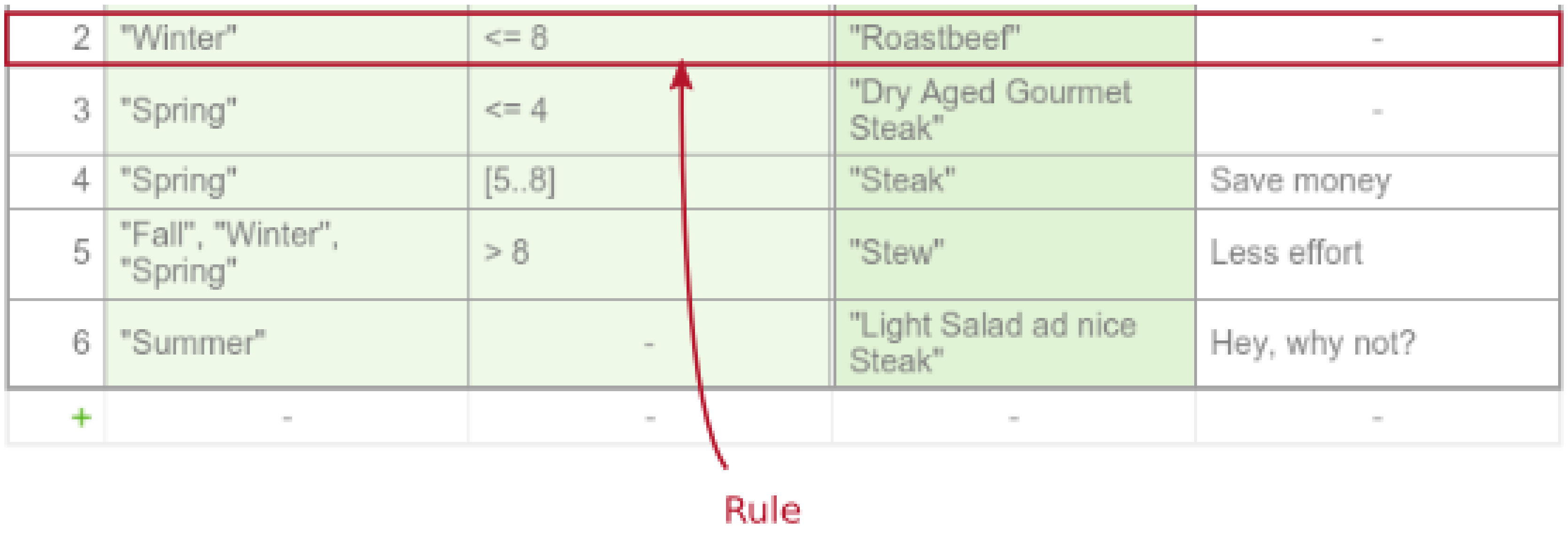


Dish

decision

U	Input +		Output +	
	Season	How many guests	Dish	
	season	guestCount	desiredDish	
	string	integer	string	Annotation
1	"Fall"	<= 8	"Spareribs"	-
2	"Winter"	<= 8	"Roastbeef"	-
3	"Spring"	<= 4	"Dry Aged Gourmet Steak"	-
4	"Spring"	[5..8]	"Steak"	Save money
5	"Fall", "Winter", "Spring"	> 8	"Stew"	Less effort
6	"Summer"	-	"Light Salad ad nice Steak"	Hey, why not?
+	-	-	-	-

Camunda: Decision Tables



2	"Winter"	<= 8	"Roastbeef"	-
3	"Spring"	<= 4	"Dry Aged Gourmet Steak"	-
4	"Spring"	[5..8]	"Steak"	Save money
5	"Fall", "Winter", "Spring"	> 8	"Stew"	Less effort
6	"Summer"	-	"Light Salad ad nice Steak"	Hey, why not?
+	-	-	-	-

A decision table can have one or more rules. Each rule contains input and output entries. The input entries are the condition and the output entries the conclusion of the rule.

Camunda: DMN

Decision Tables in Camunda are represented and editable through DecisionTable XML elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/DMN/20151101/dmn.xsd" id="definitions" name="definitions"
namespace="http://camunda.org/schema/1.0/dmn">
  <decision id="dish" name="Dish">
    <decisionTable id="decisionTable">
      <!-- ... -->
      <rule id="rule2-950612891-2">
        <inputEntry id="inputEntry21">
          <text>"Winter"</text>
        </inputEntry>
        <inputEntry id="inputEntry22">
          <text><![CDATA[<= 8]]></text>
        </inputEntry>
        <outputEntry id="outputEntry2">
          <text>"Roastbeef"</text>
        </outputEntry>
      </rule>
      <!-- ... -->
    </decisionTable>
  </decision>
</definitions>
```

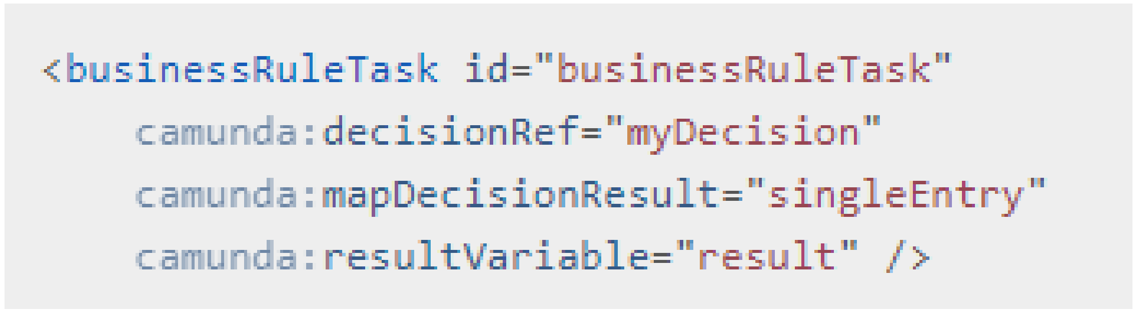
Business Rules in Camunda

In the definition of the Business Rule Task, three elements have to be specified.

- **decisionRef:** the input variable for the table;

- **mapDecisionResult:** the mapper to use for the table search;

- **resultVariable**: the variable in which the output result taken from the table is stored.



```
<businessRuleTask id="businessRuleTask"
    camunda:decisionRef="myDecision"
    camunda:mapDecisionResult="singleEntry"
    camunda:resultVariable="result" />
```