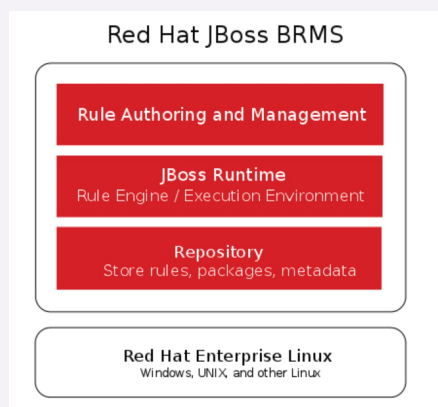


### Red Hat JBoss BRMS Components



Software packages include: Expert, Guvnor, Flow, Fusion, and Drools integrated development environment (IDE).

### Business rule structure

#### Declarative rules

when “condition” then “consequence”

**Facts** Domain model objects used by Drools to evaluate conditions and execute consequences

Can be loaded from a database

**Stated:** Provided to rules engine by the caller

**Inferred:** calculated based on the stated facts

**Does persist:** May be long lived

#### Working memory

Holds objects acted upon by business rules

Does not persist. Only lives while needed

### Business rule structure (cont)

Developer asserts 'facts' into 'working memory' and then tells engine to run rules. As the engine runs rules on the 'facts' in working memory the effect may result that the 'facts' change. The engine will run the rules again until no rules left to run.

Rules written in Drools are stored in .drl files.

### BRMS Rule Components

#### LHS conditional elements

A list of constraints on facts (LHS)

**Pattern:** A pattern is zero or more constraints with optional binding.

#### RHS actions/consequences

An action executed by this rule if facts with the list of constraints are found in the working memory (RHS).

### Methods of authoring rules

#### Business Central web interface

Technical rule editor - creates .drl file

Guided rule editor - creates .brl file

Decision table editor

DSL (Domain-Specific Language) natural language extensions

#### JBDS Drools perspective, rule resource wizard

Individual rule - creates .drl file

Rule package - creates .drl file

DSL (Domain-Specific Language) natural language extensions

#### Decision table

### Methods of authoring rules (cont)

Spreadsheet or .csv file imported into JBDS project, or JBoss BRMS repository

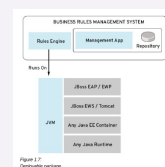
In Business Central web interface

### Bibliography

#### Red Hat Training

Authoring Rules for Red Hat JBoss BRMS - Student Workbook

### Package: Deployable



BRMS supports the deployment of business rules to Red Hat JBossvMiddleware platforms and non-JBoss runtime environments.

Because the engine is based on Java EE, it is portable to any supported Java EE runtime.

### A rule is a declarative statement of knowledge

**Remember:** Rules are not called directly, and do not call other rules directly. Rules operate in response to facts only and are fired by the rule engine.

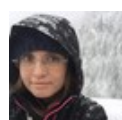
### Working memory includes an API

#### update(object, handle) or update(object)

Tell the engine that an object has changed

*However* if property change listeners are provided to the JavaBeans that are inserted into the engine, it is possible to avoid the need to call update when the object changes.

#### insert(new Something())



By Natalie Moore (NatalieMoore)

### Working memory includes an API (cont)

Place a new object into the working memory

#### retract(handle)

Removes an object from working memory (delete)

#### insertLogical(new Something())

Same as insert except that object retracted when no more facts to support truth of firing rule

### Pattern structure (LHS element)



There are three types of restrictions:

1. Single value restrictions.
2. Compound value restrictions.
3. Multi-restrictions.

When a restriction is applied to a field this becomes a constraint. When a constraint is applied to an object it becomes a pattern :)

### Constraints (LHS)

#### Might be:

single field

inline "eval"

constraint group (several field constraints tied together)

#### Connected using symbols:

' Comma

&& Represents **and**. All patterns match.

|| Represents **or**. Either pattern matches.

#### Operators are typically mathematical

== Equal to

> Greater than

< Less than

>= Greater than or equal to

<= Less than or equal to

### What Is A Rule?



Rules tell the system to look for certain patterns, and when found to perform certain actions.

"When" = Look for certain patterns

"Then" = Perform certain actions

### Forward chaining

#### What is forward chaining?

Can be described logically as repeated application of modus ponens (the when, then statements). Forward chaining is a popular implementation strategy for expert systems, business and production rule systems.

As a rule fires, it can change the objects in working memory.

Some rules should fire before others

Forward chaining is in the rule engines to decide which rules to fire in which order

Recursive: rules are executed when the LHS conditions are met. The actions may change facts, causing new rules to be fired.

#### Important to understand. Understanding forward chaining will help get the most out of the rule engine

### Modus Ponens

Latin name of a very old and common form of propositional logic

"P implies Q; P is asserted to be true, so therefore Q must be true."

If it is raining, I will meet you at the theater.  
It is raining.

Therefore, I will meet you at the theater.

### RHS actions/consequences

Consequence or action required

Actions to be executed by the rule.

Never use imperative or conditional code on RHS

#### Must be atomic

"when this, then do this," not "when this, maybe do this or this."

#### Typical actions

update Indicates that an object has changed and rules may need to be reconsidered.

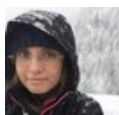
insert inserts a new object in working memory

insert-Logical inserts a new object, but the object is automatically retracted when there are no more facts to support the truth of the rule.

retract removes an object from working memory (delete)

Rules should be as declarative as possible. Think about whether it is a rule or code. Code belongs in the object model, utility classes and functions. Code should be kept out of rules as much as possible.

Java code should be limited to action statements (such as setting a value). Do not use if-else, for-while loops, or other Java logic.



By Natalie Moore  
(NatalieMoore)