## Rule history and versioning (part 1)

This column is the first in a series that will provide the reader with best practices on using or choosing a rules engine. The following topics will be discussed in future columns:
• magic values
• rule templates
• rule on/off
• exception handling
• what to do procedural, when to do rules
• local variables in rules
• arrays and chaining
• forward chaining over multiple instances
• backward or forward chaining?
The target audience for this series is typically the user of a rule engine, i.e., a programmer or someone with programming skills. All coding examples should be read as *pseudo-code* and should be easily translated to a specific target syntax for a rule engine that supports backward and forward chaining in an object-oriented environment.

Today we discuss the way you can deal with rule versioning and rule history in a declarative way. From now on the term 'rule versioning' will be used to denote rule versioning and history.

## Analysis of the rule versioning problem

The following statements describe the rule versioning problem:

- Rules are declarative statements that are used in one or more tasks to derive a certain value for an attribute.
- Some rules may only be applied in a certain period. This period can be described with a start-date and a stop-date. A stop-date is not obligatory. Rules with only a start-date are at least applicable from the start-date to the current date.
- The rules applicable on a certain date should be consistent.
- A case from the past can be re-assessed using the rules that were applicable in the period that the case deals with.
- The date which is used for retrieving the correct rules can differ based on:
    - *The task that is going to be performed.*
    - *The case (situation) that the rules are going to be applied on.*
    - *The event that induced the activation of the rules.*

Some rule engines have out of the box support for rule versioning. The tool may support one of the three methods that we will discuss in this series.  In that case it is still a good idea to be aware of alternative methods as discussed in this series.

## Solving the rule versioning problem in a declarative way

In this paragraph we make a proposal for solving the rule versioning problem in a declarative way. We will use the following abstract example rules to illustrate the ideas.

**example rules**

```
ifrule 1 – version 1
if age > 18
then
applicant.rejected = false
endifrule 1 – version 2
if age > 20
then
applicant.rejected = false
end
ifrule 2
if applicant.rejected = false
then
premium = 20%
end
```

Each rule that is subject to rule versioning should have an extra condition that specifies the period in which the rule is applicable. With the above example rule 1 (versions 1 and 2) we will get two variants:

**example code using a markdate in the condition of a rule**

```
ifrule 1 – version 1
if markdate >= 01-01-1999 and markdate < 01-01-2001
and age > 18
then
applicant.rejected = false
end
ifrule 1 — version 2
if markdate >= 01-01-2001
and age > 20
then
applicant.rejected = false
end
```

 

The extra condition uses a ‘markdate’ attribute (translation from a Dutch word). The value of markdate can be derived by other rules. For example:

**example rule deriving markdate**

```
ifrule markdate – normal case
if task = “calculate premium”
and event = “normal calculation”
then
markdate = currentdate
end
ifrule markdate – recalculation case
if task = “calculate premium”
and event = “re-calculation”
then
markdate = period.startdate
end
```

The inference engine will automatically use the right rules on the right moment when we backward chain on the attribute ‘premium’. The next code example can be an example of triggering the rules:

**example code to trigger rules**

```
method:  example_triggering_rulesstart inference
postrules( example rule 1 — version 1,
example rule 1 — version 2,
example rule 2,
example rule markdate — normal case,
example rule markdate — recalculation case)
backwardchain(premium)
end
```

Given the following case:

- Task = calculate premium
- Event = re-calculation
- Current.period.startdate = 02-11-2002
- Current.age = 22

The inference engine will follow this reasoning path:

- Evaluate rule “Example rule 2”,  applicant rejected = unknown

 

- Evaluate rule “Example rule 1 — version 1”, markdate = unknown
- Evaluate rule ” Example rule markdate — recalculation case”, markdate = 02-11-2002
- Evaluate rule “Example rule 1 — version 1”, markdate = unknown, rule fails
- Evaluate rule “Example rule 1 — version 2”, applicant rejected = false
- Evaluate rule “Example rule 2”,  premium = 20%

The backward chaining process will automatically apply the rules that are applicable to the case. You can do the same in a forward chaining mode although it will be less efficient.

The rules for deriving ‘markdate’ can contain complex business logic. If they are subject to rule versioning as well, a new Markdate should be introduced:

**example code markdate-ruleversioning**

```
ifrule markdate – normal case – version 1
if markdate-ruleversioning < 09-09-99
and task = “calculate premium”
and event = “normal calculation”
then
markdate = currentdate
endifrule markdate — normal case — version 2
if markdate-ruleversioning >= 09-09-99
and task = “calculate premium”
and event = “normal calculation”
then
markdate = current.period.enddate
end
```

## Evaluation of the solution

The proposed solution has the advantages of declarative programming in general:

- The rules about rule versioning are explicitly stored.
- Versions of rules can easily be changed.
- Versioning strategy of rules can easily be changed.

With the use of verification techniques overlapping applicability periods between different versions of the same rule can be easily detected. This is illustrated in the next example:

 

**example verification on the markdate**

```
ifrule 1 – version 1
if markdate >= 01-01-1999 and markdate < 01-01-2001 and
age > 18
then
applicant.rejected = false
endifrule 1 — version 2
if markdate > 31-12-2000
and age > 20
then
applicant.rejected = false
end
```

These two rules can be detected as containing a redundancy for attribute ‘markdate’.

**ifrule 1 — version 1**
if markdate >= 01-01-1999 and markdate < 01-01-2001 and age > 18
then
applicant.rejected = false
end

**ifrule 1 — version 2**
if markdate >01-01-2001
and age > 20
applicant.rejected = false
end

In this situation it can be detected that there is no rule applicable on 01-01-2001.

The disadvantage of the proposed solution is that all versions of the rules need to be posted to the inference engine. When there are a lot of rules this might be inefficient and result in decreased performance of the system. In the next column a solution to this problem is proposed.

*This article was originally published by BRCommunity (link).*