Rule history and versioning (part 3)

> this column is one 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 continue our discussion on how to deal with rule versioning and rule history in a declarative way. In our first column[1] on this topic we described the characteristics of the rule history and versioning problem, and we discussed an example and a simple solution. In our second column[2] on this topic we discussed an efficient and declarative solution that can be used when rules are versioned on a small number of dates (for example every year on the first of January). Today we will discuss a strategy for dealing with rule versions that is efficient and works if there are lots of different new versions of rules. Some rule engines have implemented a strategy like this as basic functionality in their environment.

Solving the rule versioning problem in an advanced way

For complex rule versioning it is recommended to see a rule as an object with attributes (meta information) and to create an object for each rule. This allows one to specify extra information about a rule in your object model, use inheritance to propagate certain properties of rules to a wider range of rules, and do selections of rules based on that information.

A Rule Class would have the following class attributes:

- StartDate

  date from which the rule is applicable
- EndDate

  date until which the rule is applicable
- PostRule

  Boolean attribute indicating whether the rule should be posted
- RuleStatement

  actual rule statement or reference to rule statement
- RuleHandle*

  Each rule that is posted to the inference engine receives a handle; by storing the information you can dynamically disable and enable rules without restarting the inference engine. (This functionality may not be supported by all rule engines)
- Tasks*

  information about the applicability of a rule in a certain functional context

*The Attributes marked with * are not necessary to solve the rule versioning and history problem. They are listed to give hints for further improvements of the performance of posting rules or depend on the functionality of the rules.*

The Rule Class should have a method “RuleStatement” in which the actual rule is stated or referenced.

For each rule in the application, a subclass of the RuleClass should be created. In this class the class attributes StartDate, EndDate, and RuleStatement get a value.

Given the above structure, the implementation of the task will have the following structure:

example code

```
task()start inference
rulesetmarkdate
getrules
forwardchain
end
start inference
postrules
forwardchain
goalmakeunknown(->premium)
backwardchain(->premium)
end
```

The first inferblock derives the markdate and decides which rules to post. The ruleSetMarkDate ruleset is the same as in our previous examples. The ruleset GetRules selects the rules that need to be posted with if-match rules (rules that consider multiple instances at a time) in the following way:

example code

```
getrules()
bind r to ruleclass
ifrule getapplicablerules
ifmatch r
where
(r.startdate > markdate or isunknown(->r.startdate))
and (markdate < r.enddate or isunknown(->r.enddate))
then
r.postrule = true
end
```

The second inferblock posts the necessary rules with the use of a forwardchain statement and the following rule in the ruleset PostRules:

example code

```
postrules()
bind r to ruleclass
ifrule getpostedrules
ifmatch r
where
r.postrule = true
then
r.rulestatement
end
```

After the forward chaining statement, the applicable rules are posted and the task inference can start, in this example with a backwardchain statement on premium.

Evaluation of solution

The third solution offers a lot of flexibility; like solution 2, the solution looks more efficient because in the final step only the applicable rules are posted. The performance gain of this solution will depend on:

- The number of rules subject to rule versioning;
- The complexity of the reasoning task in the applicable rules. If this complexity is higher, it is expected that the gain is greater.

The solution, though, will not guarantee to give the necessary performance win. However, the solution gives many more possibilities for increasing the performance when it is not good enough. The strength of this solution is that it is scalable. For example, the programmer can work with hash tables or database queries to select the rules that must be evaluated. This is the main difference with solution 2.

A disadvantage of this approach is that you cannot use verification technology to see if there are rules that express the same logic and have overlapping applicability periods. Verification will be more difficult, in general.

Recommendations

In short I recommend:

- Add conditions to a rule that specify versioning information when only a (small) subset of the rules is subject to versioning. (See [1].)
- Group rules in rulesets based on their applicability period when a large subset of the rules is subject to versioning and the periods in which rules change are usually the same for these rules. (See [2].)
- Create tables or classes that allow you to store meta information for each rule, and create a selection algorithm (eventually rules based) to select only the applicable rules. Use this solution when a very large subset of the rules is subject to versioning and the applicability period of individual rules is different for all these rules.

Learned something new? Let me know by sharing this post.

This article was originally published by BRCommunity (link).

[1] Silvie Spreeuwenberg, “Rule History and Versioning (Part 1),” *Business Rules Journal*, Vol. 8, No. 11 (Nov. 2007), URL: http://www.BRCommunity.com/a2007/b375.html

[2] Silvie Spreeuwenberg, “Rule History and Versioning (Part 2),” *Business Rules Journal*, Vol. 8, No. 12 (Dec. 2007), URL: http://www.BRCommunity.com/a2007/b382.html