OCEB White Paper on Business Rules, Decisions, and PRR

Version 1.1, December 2008

Paul Vincent, co-chair OMG PRR FTF
TIBCO Software

Abstract

The Object Management Group's work on standards for business rules has led to a production rules standard - PRR, for Production Rule Representation. This standard has been developed by a consortium of industry and academic interests under the auspices of the OMG Business Modeling and Integration Domain Task Force, which also covers business architecture and process modeling. PRR provides UML extensions for rule-based behaviors, in particular focusing on the production rules used commonly to provide BPM and SOA systems with automated decisions via BREs and BRMSs.

Introduction

Most students of Computer Science will have come across production rules during the study of the expert systems and knowledge-based systems that were popular in the 1980s and early 1990s. Expert systems such as Mycin and R1 resulted in a whole new genre of artificial intelligence tools that often represented expert knowledge, for example about diagnostic problems, as if-then rules that were executed by rules engines on an as-required basis. Much of this early work also progressed to knowledge representation research that is now realized through ontology languages and formal logic representations – a common area of study in today's AI research labs.

In order to efficiently represent the expert knowledge and practices required for these knowledge-based systems, research was carried out into rule execution algorithms and languages to allow for efficient execution. In particular, it was found that rule conditions could effectively be compiled into an efficient pattern matching mechanism called the Rete algorithm. In the 1990s, it was realized that expert and knowledge-based systems were effectively automating those business rules in organizations used to automate decisions. The result was a trend for software developers to embed these Business Rule Engines (BREs) to handle the business logic in their applications or custom processes. These BREs were mostly Rete-based and worked in a data-driven, forward chaining manner, whereby the action of some rule would set some data that resulted in some other rule being fired.

Rules and Business Rules

UML models have for a long time been constrained through constraint rules defined with OCL – Object Constraint Language. This is a technical modelling standard for specifying interrelationship constraints on UML classes. OCL is part of the UML specification.

However, business rules are defined using business terminology, so SBVR – the Semantics of Business Vocabulary and Rules standard – was developed. SBVR, as its name implies, defines a vocabulary for terms and facts as well as the rules that relate them. SBVR provides a formal mechanism for documenting all business rules relating to business policy in a formal business-domain language. However, SBVR does not concern itself with detailed behavioral rules for business operations, processes or decisions. It is envisaged that SBVR rules would guide or influence the design of operational processes and decisions, and these processes and decisions would then enforce or implement these policy rules.

The OMG MDA (Model Driven Architecture) defines various levels of models with the idea of automated or guided transformations between them. Therefore, a business or Computation-Independent Model (MDA CIM), such as a term and fact model defined in SBVR, can in theory be transformed to a UML Class model, and the associated SBVR policy rules mapped to associated UML OCL constraints, with the UML Classes and OCL being at the platform-independent model level (MDA PIM). Such mappings from SBVR are the subject of ongoing research.

The business rules managed in the usual BRMS (Business Rule Management System) are not generally SBVR type policy rules, but usually represent operational rules and decisions to drive automated business processes. Such rules are also at the level of the MDA CIM, but in vendor-specific representations and custom translations to the respective vendor-specific BRE languages.

BRE rule languages are typically representations of production rules, and represent behaviours such as methods, Action Languages, scripts, and activity diagrams. Such rules are vendor-specific and can be represented at the Platform-Specific Model level (MDA PSM).

Tying the MDA CIM-level concepts together is the OMG BMM (Business Motivation Model), which provides a structure for describing business means and ends, and associated strategies, tactics, policies and business rules to drive operational systems and decisions.

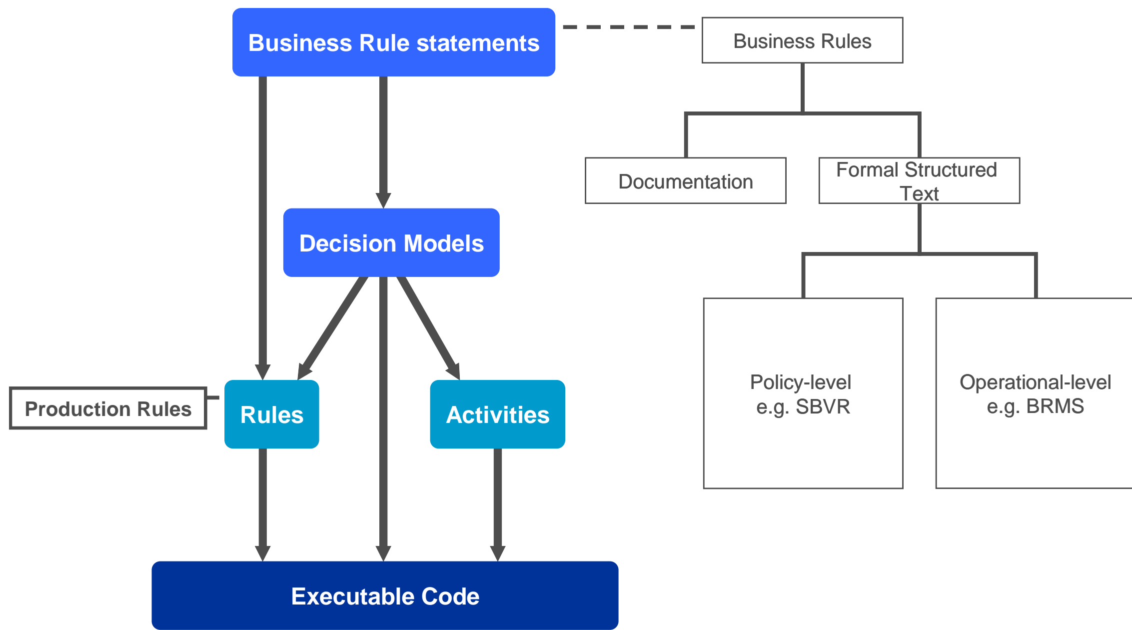

Figure 1 – The relationship between Business Rules and Production Rules

Production Rules

Production rules represent "if <condition list> then <action list>" statements. For common production rule engines used as BREs[1] these are defined declaratively, which means they can be defined in any order (as the rule engine will determine execution order). Usually such rule engines are also typically classed as inference engines, as they can infer new information. The rule conditions (or LHS - Left Hand Side) and actions (or RHS - Right Hand Side) are commonly expressed in terms of a *business object model*, equivalent to a UML class model, on which the rules are dependent. One consequence of this is that changes to the structure of the class / object model may require any dependent rules to be re-factored to suit.

Production rules are usually organized within *rulesets*. A ruleset may represent simply a structure for the convenience of managing the rules, or provide an execution context (for example with parameters that map onto business terms used in the rules) representing a program function or method.

Rulesets, or groups of rulesets, can be used as a *rule service* or *decision service*, either invoked directly from a calling application, or as a service such as a web service. Often this usage is associated with stateless operation of the rule engine – no information is retained in the rule engine between transactions.

For example, a "determineLoanValue" rule service could require several rulesets together with a business object model detailing the loanee and associated product, with the computations defined in the rules' actions.

Note that because it is often easier to design large rule systems as a sequence of independent rulesets to be executed in some order, rule engines sometimes extend the notion of rule execution with mechanisms to orchestrate rulesets – typically called "ruleflows".

Another approach is to deploy rulesets in a continuous, event-driven rule engine or agent for tasks such as CEP (Complex Event Processing). Other UML constructs such as state models might be used to provide context for rule execution. Modeling the state of entities over time, and the continuous processing of events, usually requires stateful operation of the rule engine so that information is retained in the rule engine between events.

---

[1] Although BRE (Business Rule Engine) is not formally defined, a good definition might be "a rule engine whose rules are described in terms of a business object model or business terms and facts".

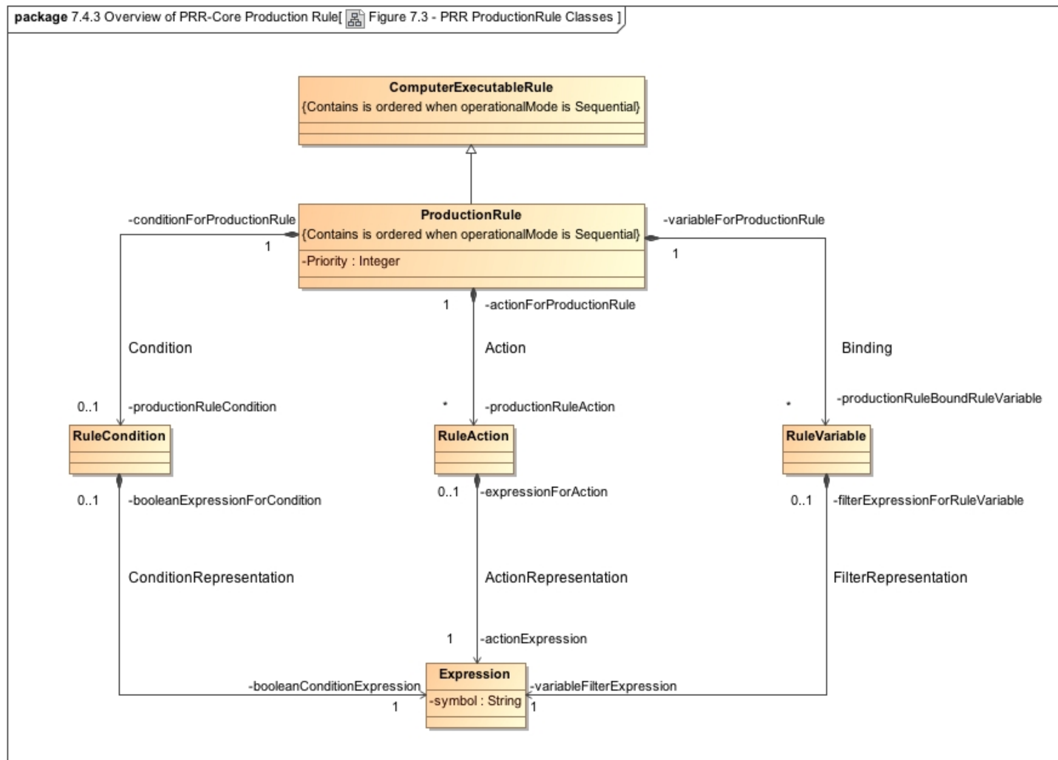

Figure 2 – PRR Definition of ProductionRule

Production Rules in Business Processes

For business processes represented in a BPMS (Business Process Management System), detailing decision logic within the process diagram often obfuscates the core business processes. Business processes can represent manual (workflow) or automated tasks, with the commonest form of process representation being BPMN (Business Process Modeling Notation).

Occasionally subprocess diagrams or graphs will be used to specify re-usable decision logic. In other cases, separate declarative rules may be preferred, and the process activity will delegate to a rule service or decision service made up of production rules. Often, this rule service will return a decision value for a BPMN gateway in order to influence business process execution or represent a business decision.

Less common but increasingly significant roles for rules in processes include the selection of business processes, where a rule executes a business process as an action, and process monitoring, where a rule is used to check the status of running processes.

Decision Models versus Production Rules

The most common format[2] for BPM users to represent business rules is the decision table. This provides a common set of condition and action statements, with the table providing different values representing different rules. Some systems map decision tables to a specific algorithm; others will map them to component production rules. Similar models are decision trees and decision graphs.

Note that decision models output from Predictive Analytics tools may or may not be usefully mapped to production rules. One example might be a segmentation model representing a decision tree segmenting customers for marketing offers, which maps to a decision tree and thence production rules. Alternatively a model type such as a neural net representing a face-recognition feature will not usefully map to production rules. Often such analytics tools generate models in a language called PMML (Predictive Model Markup Language).

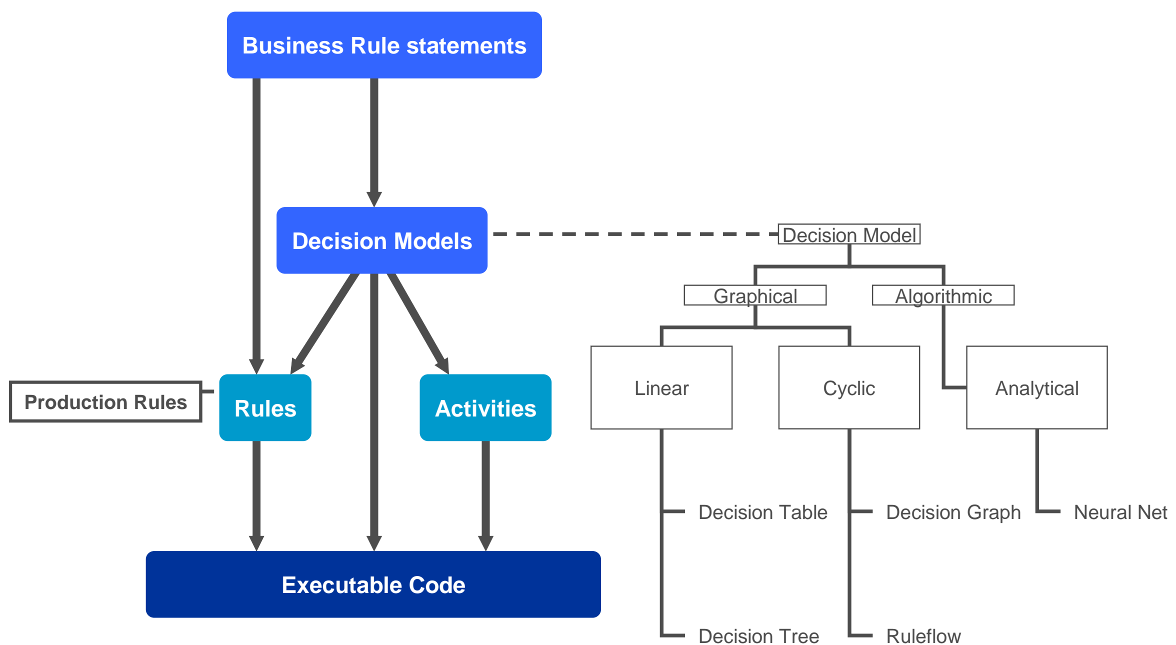

Figure 2 – The Relationship between Decision Models and Production Rules

[2] Only anecdotal evidence exists for this. It is likely to be a subject for future research.

Rationale for a Production Rule Representation

Most rule engine vendors define their own formats for Business Object Models and (production) rules. In addition UML modeling tools rarely support the notion of declarative rules outside of OCL class constraints. The UML Production Rule Representation (PRR) was defined to provide an MDA Platform Independent Model (PIM) for production rules that would standardize model-driven engineering when using common BREs.

PRR achieves 2 main goals:

- a basic Rule and Ruleset behavior that can be subclassed to other executable rule types as needed.
- a metamodel for the production rules used in BREs[3] that use the Rete type of approach. In addition it provides a non-Rete type to allow for simple rule definitions as used in many BPM systems' internal definition of rules.

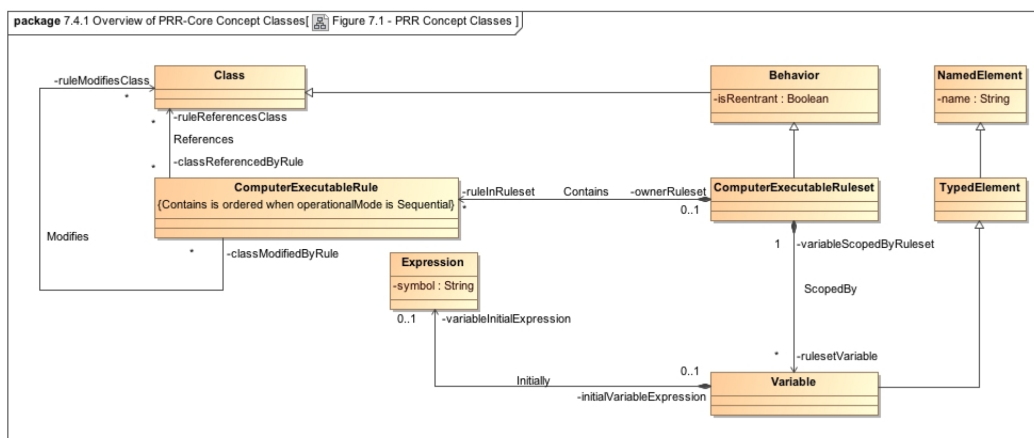

Figure 3 – PRR definition of ComputerExecutableRule and Ruleset

[3] Hereafter, the term BRE is used as a synonym for Rete-driven inferencing production rule system, as per common usage.

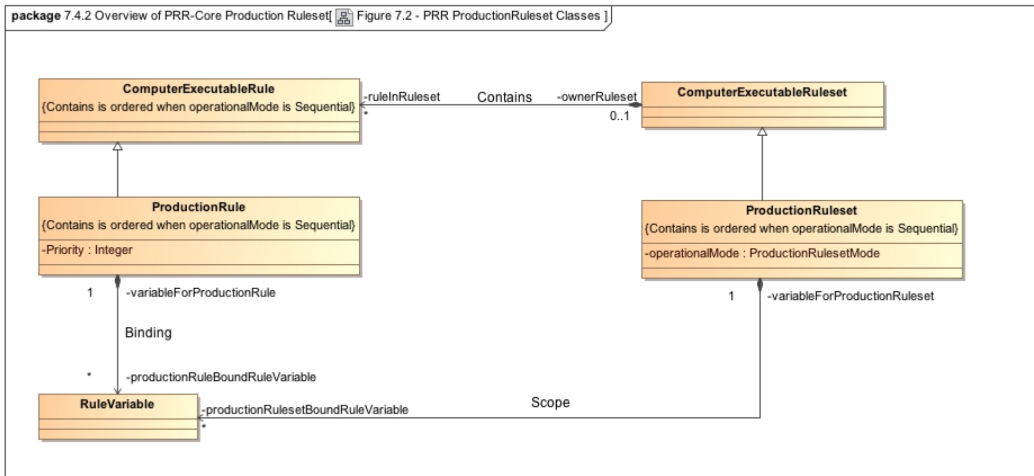

Figure 3 – ProductionRule vs ComputerExecutableRule

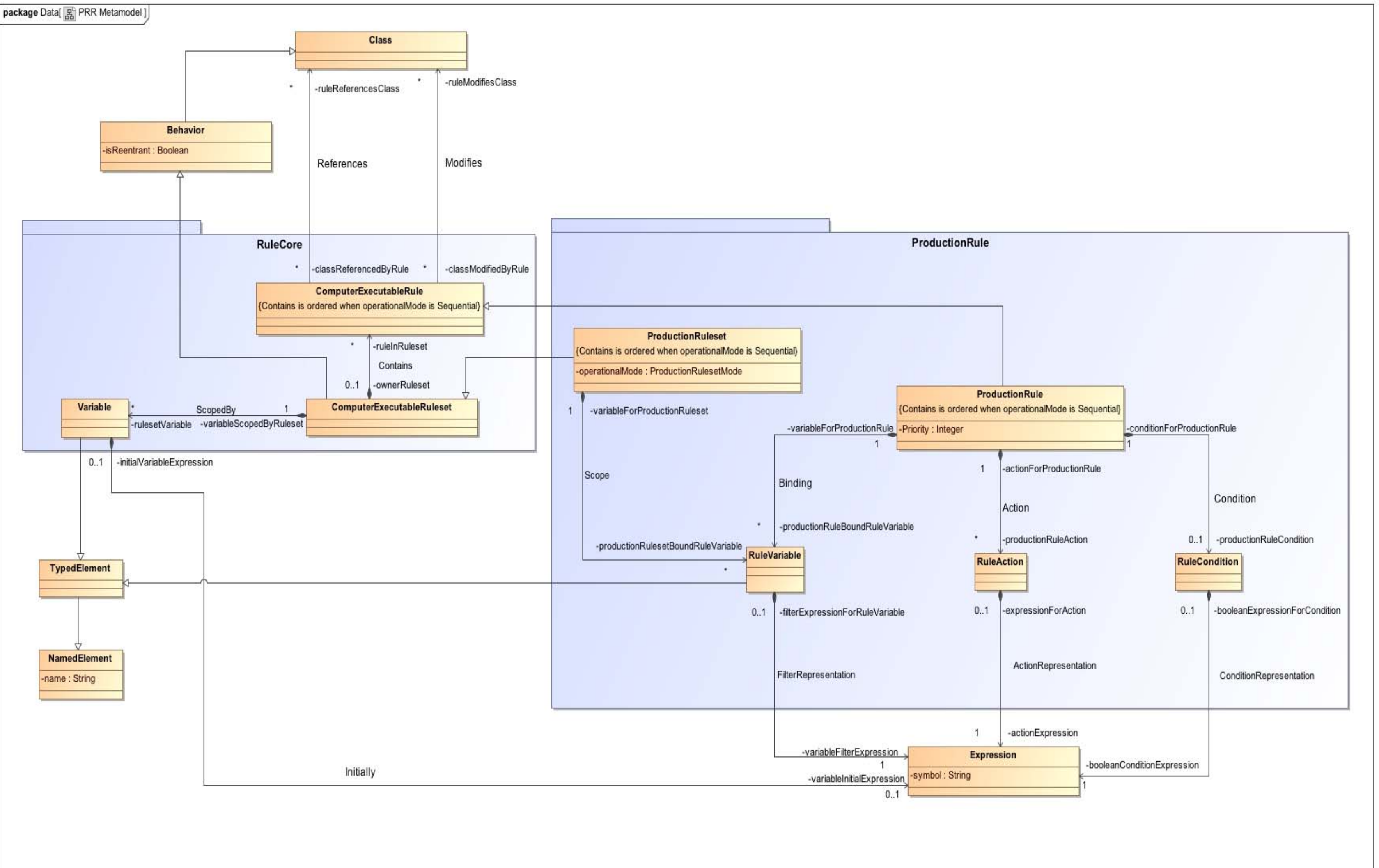
package Data[PRR Metamodel]
Class
-ruleReferencesClass
-ruleModifiesClass
References
Modifies
Behavior
-isReentrant : Boolean
RuleCore
-classReferencedByRule
-classModifiedByRule
ComputerExecutableRule
{Contains is ordered when operationalMode is Sequential}
-ruleInRuleset
Contains
-ownerRuleset
Variable
ScopedBy
-rulesetVariable
-variableScopedByRuleset
ComputerExecutableRuleset
-initialVariableExpression
TypedElement
NamedElement
-name : String
ProductionRule
ProductionRuleset
{Contains is ordered when operationalMode is Sequential}
-operationalMode : ProductionRulesetMode
-variableForProductionRuleset
Scope
-productionRulesetBoundRuleVariable
ProductionRule
{Contains is ordered when operationalMode is Sequential}
-Priority : Integer
-variableForProductionRule
-actionForProductionRule
-conditionForProductionRule
Binding
-productionRuleBoundRuleVariable
Action
-productionRuleAction
Condition
-productionRuleCondition
RuleVariable
RuleAction
RuleCondition
-filterExpressionForRuleVariable
-expressionForAction
-booleanExpressionForCondition
FilterRepresentation
ActionRepresentation
ConditionRepresentation
-actionExpression
-variableFilterExpression
-variableInitialExpression
Expression
-symbol : String
-booleanConditionExpression
Initially

Figure 5 – The PRR Metamodel

Limitations of PRR 1.0

Ideally, PRR would provide a common language for defining rules against UML classes and objects, and a common diagramming or rule entry format. The former is a particular limitation due to the fact that there is no standard script or concrete action language for use in UML – this is because UML is required to be open to all types of platform specific languages. For this reason, PRR itself does not provide a complete answer to a standard rule language for BPMSs.

A non-normative (i.e. not part of the official standard) expression language for conditions and actions – PRR OCL, based on OCL – is included in the specification as an example of a PRR-suitable expression language.

These limitations will likely be addressed in future versions of PRR.

How PRR Rules Work

PRR's main goal is to provide a common metamodel for the production rules used in BREs. The major BRE vendors all support a common modus operandi, whereby rules are defined[4] in terms of *rule variables*[5], *conditions* and *actions*.

To explain[6] the semantics of PRR rules we must first consider what happens to rules at runtime. Rules are generally defined in terms of classes (represented in the rule variables), and the condition statements are Boolean expressions that both filter instances of the rule variables and act as join statements between them. At runtime, the business object model is populated with data and events for the use of the rules – this is termed "working memory".

- **Declarative Rule definition**
  - Defined in terms of RuleVariables
  - Each tuple of RuleVariables
    + the instantiated rule condition
    + the instantiated rule action
    represents a "rule instance"
- **Scope / declaration**
  - Classes / Events relevant for the rule
- **Conditions**
  - Filters on declarations
  - Joins across declarations
- **Actions**
  - What to do
    for each tuple
    that satisfies the conditions…

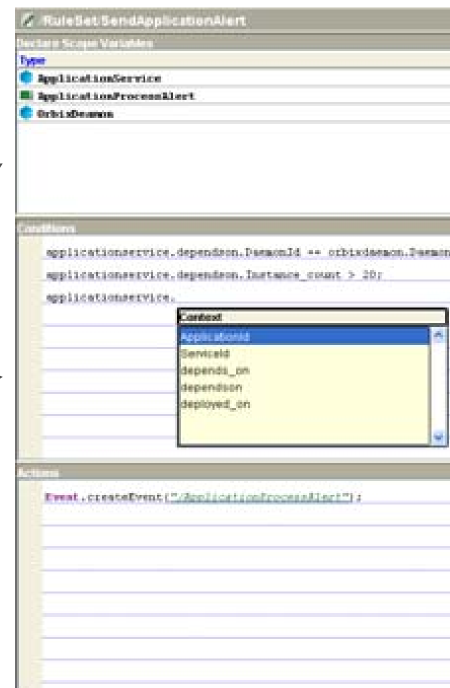

Figure 3 – A Production Rule Definition

During execution, the combination of rule variables in the rule definition can be considered as representing a tuple. Potentially, all possible combinations of rule variables, instantiated with any instances currently defined in working memory, are available for rule processing – meaning that 0 to many tuples may be valid. But

---

[4] For programmers familiar with the IF.. THEN.. construct in a conventional 3GL, the main obstacles to understanding production rules are the declarative definitions and the rule variables.

[5] RuleVariables are described differently by vendors, such as rule declarations (TIBCO), variables (ILOG) and patterns (Blaze).

[6] This description of rule engine semantics is not intended to be authoritative, but conceptual enough for users to understand. In particular we do not describe here rule scheduling (for execution), non-monotonic reasoning, or conflict resolution.

- when any rule variable has no instance in working memory, then no tuple can be defined and the rule cannot be considered for execution.
- when only 1 instance of each rule variable is in working memory, then only a single tuple is valid and only 1 rule firing is possible.

Rules will "fire" – i.e. the rule actions will be executed - if the condition expression holds for the rule and the rule is scheduled to execute (versus any other rules).

Typically, rule engines will execute all tuples for all rules until there are no more to be processed. Of course, some rule actions may additionally affect working memory such that new rules and new tuples become valid for processing, and other rules and tuples become invalid. Rule engines handle this behaviour automatically, adjusting the schedule of rule firings accordingly.

From the above, it can be seen that production rules for rule engines are not just program statements, but actually represent patterns that match against instances of classes at runtime – indeed, a single rule definition can easily execute against 100s or 1000s of objects when such objects are instances of its defined rule variables. Additionally, the use of efficient algorithms in rule engines ensures that their execution strategy is extremely efficient, especially when inferencing is required.

Note that for rule or decision services where:

- no inferencing is required in a ruleset,
- only 1 rule can fire,
- few if any rule variables have more than 1 instance at runtime

then the rules may be more efficiently executed as conventional procedural code, without the use of a rule engine. Some BREs allow, and most BPMSs only support, such a sequential execution mode. Therefore this model of behavior is also supported by PRR.

Use Cases for PRR

The main role for PRR, as envisaged at its conception, was to support UML modelers wanting to exploit rule engines for model-driven engineering, deploying decision rules to runtime components such as those provided by TIBCO, Ilog or Blaze rule engines.

The popularity of BPM systems, and the use of UML class models to model business entities in process modeling, means that PRR is also likely to play a future role in process management – for example, modeling decision activities.

Another area of rapid adoption is CEP, which combines events with data to identify patterns and take appropriate decisions and actions. Vendors such as TIBCO use a production rule engine for this purpose.

Summary

PRR provides a standard production rule metamodel to help marry model-driven engineering with a standard approach to representing production rules used in rule engines for business decisions. It is also a tentative first step in modeling behavioral rules in UML and providing the basis for standardized decision representations for use alongside BPMN.

* * *

Acknowledgements:
PRR Chairs: Christian de Sainte Marie (ILOG), Paul Vincent (TIBCO)
PRR Contributors have included the following organizations: TIBCO, ILOG, No Magic, Fair Isaac, Pegasystems, IBM, Sandpiper, LibRT, members of RuleML
Diagrams: produced by Robert Ong of No Magic with MagicDraw

Notes:
Some terms and names in this document may be subject to © and ™ of their respective owners.