

### 3. Production Rule Representation

#### 3.1 Introduction to PRR-Core and PRR-OCL

The following MOF2 compliant metamodel defines the PRR. It features:

- A definition of production rules for forward chaining inference and procedural processing.
- A non-normative definition for an interchangeable expression language (PRR OCL) for rule condition and action expressions, so they can be replaced by alternative representations for vendor-specific usage or in other standards.
- A definition of rulesets as collections of rules for a particular class of platform (procedural or inference rule engine).

The metamodel is composed of:

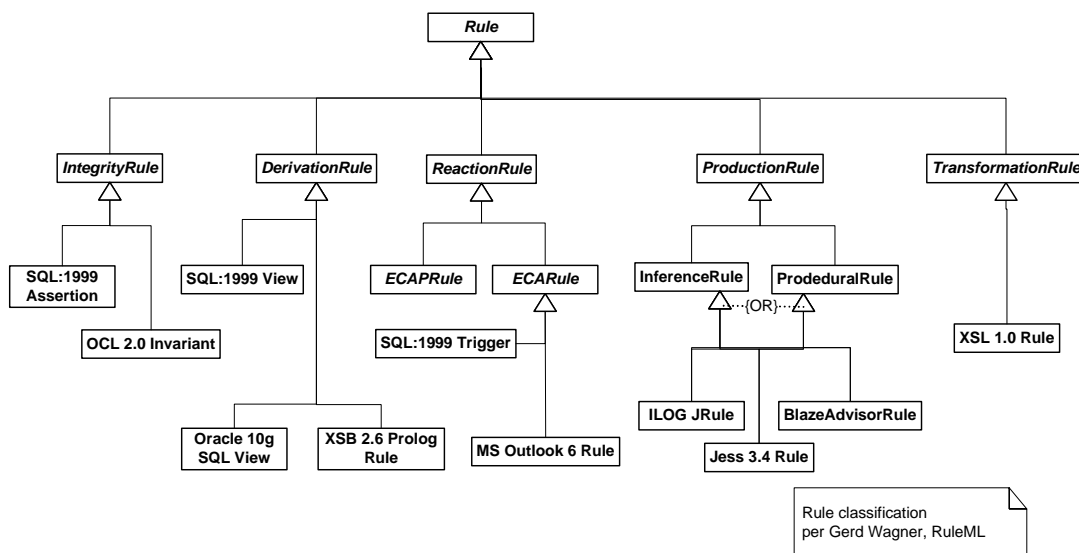
- a core structure referred to as PRR Core
- an non-normative abstract OCL-based syntax for PRR expressions, defined as an extended PRR Core metamodel referred to as PRR OCL.

Future extensions of PRR may address:

- rule metamodels for other classes of rules, such as Event-Condition-Action (ECA), backward chaining, and constraints
- rule representations that are specific to graphical notations, such as decision tables and decision trees
- representations of sequences of rulesets within larger decisions
- transformations between PRR and other MDA models such as SBVR.

Other concrete syntaxes may be applied to PRR Core in future. To this end, the PRR is designed to be extensible.

Production Rules fit into the following rule classification scheme (supplied by the RuleML Initiative), although they are a subclass of Computer Executable Rule rather than Rule to avoid confusion with other uses of “Rule” as a metamodel class.



## 3.2 Production Rules

### 3.2.1 Production Rule definition

A *production rule*<sup>1</sup> is a **statement of programming logic** that specifies the execution of one or more actions in the case that its conditions are satisfied. Production rules therefore have an operational semantic (formalizing state changes, e.g., on the basis of a state transition system formalism).

The effect of executing production rules may depend on the ordering of the rules, irrespective of whether such ordering is defined by the rule execution mechanism or the ordered representation of the rules.

The production rule is typically<sup>2</sup> represented as:

if [condition] then [action-list]

Some implementations extend this definition to include an “else” construct as follows:

if [condition] then [action-list] else [alternative-action-list]

although this form is not considered for PRR; all rules that contain an “else” statement can be reduced to the first form without an “else”, and the semantics for interpreting when “else” actions are executed may be complex in some Inferencing schemes. Note that this implies that a conversion from a PSM to a PIM might be complete but not reversible. Rules with “else” statements in a PSM would result in multiple PIM rules which could not then be translated back into the original rules. The new rules would be functionally equivalent, however.

### 3.2.2 Production Ruleset definition

The container for production rules is the *production ruleset*. The production ruleset provides

- a means of collecting rules related to some business process or activity as a functional unit,
- a runtime unit of execution in a rule engine together with the interface for rule invocation.

From an architecture and framework perspective, a ruleset is a Behavior in UML terms,

The rules in a ruleset operate on a set of objects, called the "data source" in this document. The objects are provided by the ruleset's:

- parameters
- context at invocation time.

The changed values at the end of execution represent the result or "output" of a ruleset invocation.

### 3.2.3 Rule Variable definition

The condition and action lists contain expressions (Boolean for condition) that refer to 2 different types of variables (which we term as standard variables and rule variables).

At definition time:

---

<sup>1</sup> From the [RFP].

<sup>2</sup> If.. then.. rules are sometimes represented as when... then... rules by some vendors.

## Submission to Production Rule Representation

- a *standard variable* has a type and an optional initial expression. In some systems, there may also be a constraint applied to the variable, but the latter is outside the scope of PRR. Standard variables are defined at the ruleset level.
- a *rule variable* has a type and a domain specified optionally by a filter applied to a data source. With no filter, its domain defaults to all objects conforming to its type that are within scope / in the data source. Rule variables may be defined at the rule level, or at the ruleset level; in the latter case the rule variable definitions are available to all rules.

### 3.2.4 Semantics of Rule Variables

At runtime:

- standard variables are bound to a single value (that could itself be a collection) within their domain. The value may be assigned by an initial expression, or assigned or reassigned in a rule action.
- rule variables are associated with the set of values within their domain specified by their type and filter. Each combination of values associated with each of the rule variables for a given rule is a tuple called a *binding*. It binds each rule variable to a value (object or collection) in the data source. These bindings are execution concepts: they are not modeled explicitly but are the result of referencing rule variables in rule definitions.

This means that a production rule is considered for instantiating against ALL the rule variable values. The use of rule variables means that the definition of a production rule is in fact:

for [rule variables] if [condition] then [action-list]

Note that there is an implied product of rule variables when multiple rule variables are defined e.g.:

for [rule variable 1] for [rule variable 2] if [condition] then [action-list]

### 3.2.5 Semantics of Production Rules

The operational semantics of production rules in general for forward chaining rules (via a rule engine) are as follows:

- Match*: the rules are instantiated based on the definition of the rule conditions and the current state of the data source
- Conflict resolution*: select rule instances to be executed, per strategy
- Act*: change state of data source, by executing the selected rule instances' actions

However, where rule engines are not used and a simpler sequential processing of rules takes place, there is no conflict resolution and a simpler strategy for executing rules.

#### 3.2.5.1 Operational Semantics for Forward-chaining production rules

A forward chaining production ruleset is defined **without** consideration of the explicit ordering of the rules; execution ordering is under the control of the inference engine that maintains a stateful representation of rule bindings.

## Submission to Production Rule Representation

The operational semantics of forward-chaining production rules extend the general semantics as follows:

1. *Match*: bind the rule variables based on the state of the data source, and then instantiate rules using the resulting bindings and the rule conditions. A rule instance consists of a binding and the rule whose condition it satisfies. All rule instances are considered for further processing
2. *Conflict resolution*: the rule instance for execution is selected by some means such as a rule priority, if one has been specified
3. *Act*: the action list for the selected rule instance is executed in some order

This sequence is repeated for each rule instance until no further rules can be matched, or an explicit end state is reached through an action.

It is important to note that:

- In the case where more than one binding satisfies the condition, there is one separate rule instance per binding.
- An action may modify the data source, which can affect current as well as subsequent bindings and condition matches. For example, an existing rule instance may be removed because the match is no longer valid or an additional rule instance may be added due to a newly valid match.

One popular algorithm for implementing such a forward chaining production rules is the Rete algorithm [RETE]<sup>1</sup>.

### 3.2.5.2 Operational Semantics for Sequential production rules

A sequential production rule is a production rule defined **without** re-evaluation of rule ordering during execution.

The operational semantics of sequential production rules extends the general semantics by separating the match into bind and evaluate steps, where the bind step is once-only step, as follows:

1. *Bind*: bind the rule variables based on the state of the data source at invocation time, and instantiate rules using the bindings.
2. *Evaluate*: evaluate the rule conditions based on the current state of the data source. Each instance is treated as a separate rule. If the condition evaluates to false then the rule instance is not considered.
3. *Act*: execute the action list of the current rule instance

This sequence 2-3 is repeated for one rule instance at a time until all the rules are processed, or an explicit end state is reached through an action.

It is important to note that:

---

<sup>1</sup> Charles Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", *Artificial Intelligence*, 19, pp 17-37, 1982

### *Submission to Production Rule Representation*

- The processing order is defined per rule, not per rule instance. It is specific to the engine what is the ordering of the rule instances.
- The instances to be executed are defined on the initial state of the data source. Side effects from the execution of one instance will not affect the eligibility of other instances for execution. Side effects **may** affect the whether specific conditions of those rules are satisfied.
- Rule execution order is determined by the specified sequence of the rules in the ruleset.

### 3.3 Overview of PRR-Core

PRR Core is a set of classes that allow for production rules and rulesets to be defined in a purely platform independent way without having to specify OCL to represent conditions and actions. As such all conditions and actions are “opaque” and simply strings. While this limits the ability to transform rules from one production environment (PSM) to another, it would allow for sharing of rules between all tools that understand the basic structure of production rules.

### 3.4 PRR-Core Metamodels

This section specifies the PRR-Core Metamodel.

#### 3.4.1 Overview of PRR-Core concept classes

The following is a partial model showing the concepts of general rules and rulesets for future extension to other rule types.

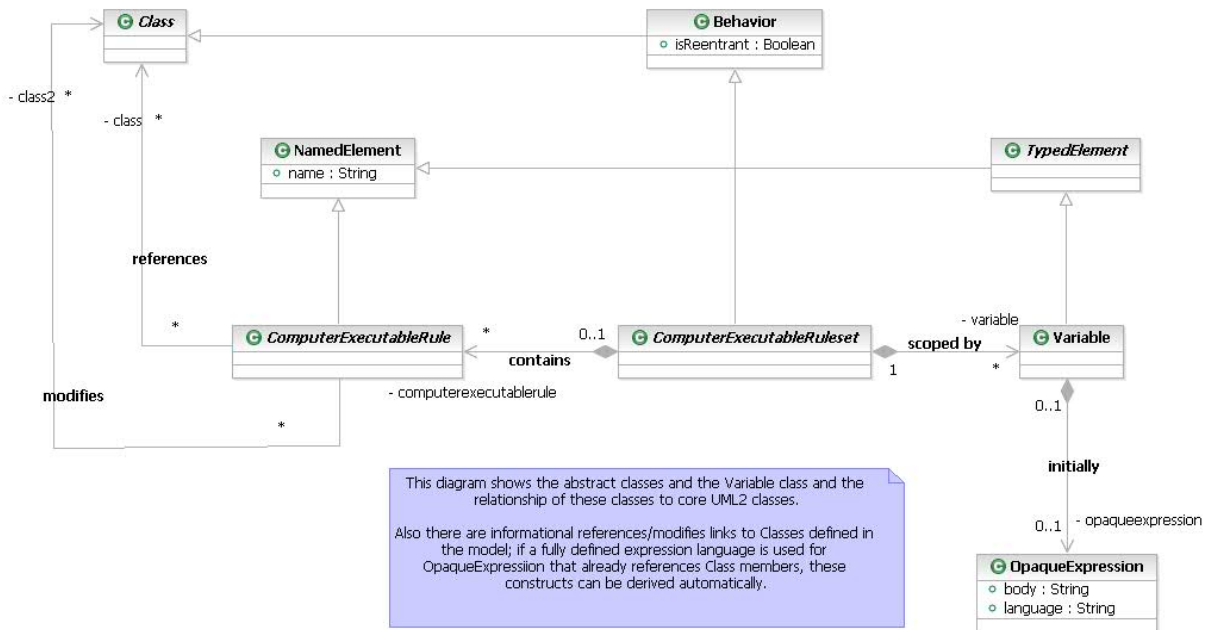


Figure 1: PRR Concept Classes

### 3.4.2 Overview of PRR-Core Production Ruleset

The following is a partial model showing the ProductionRuleset class and its relationship to other model elements.

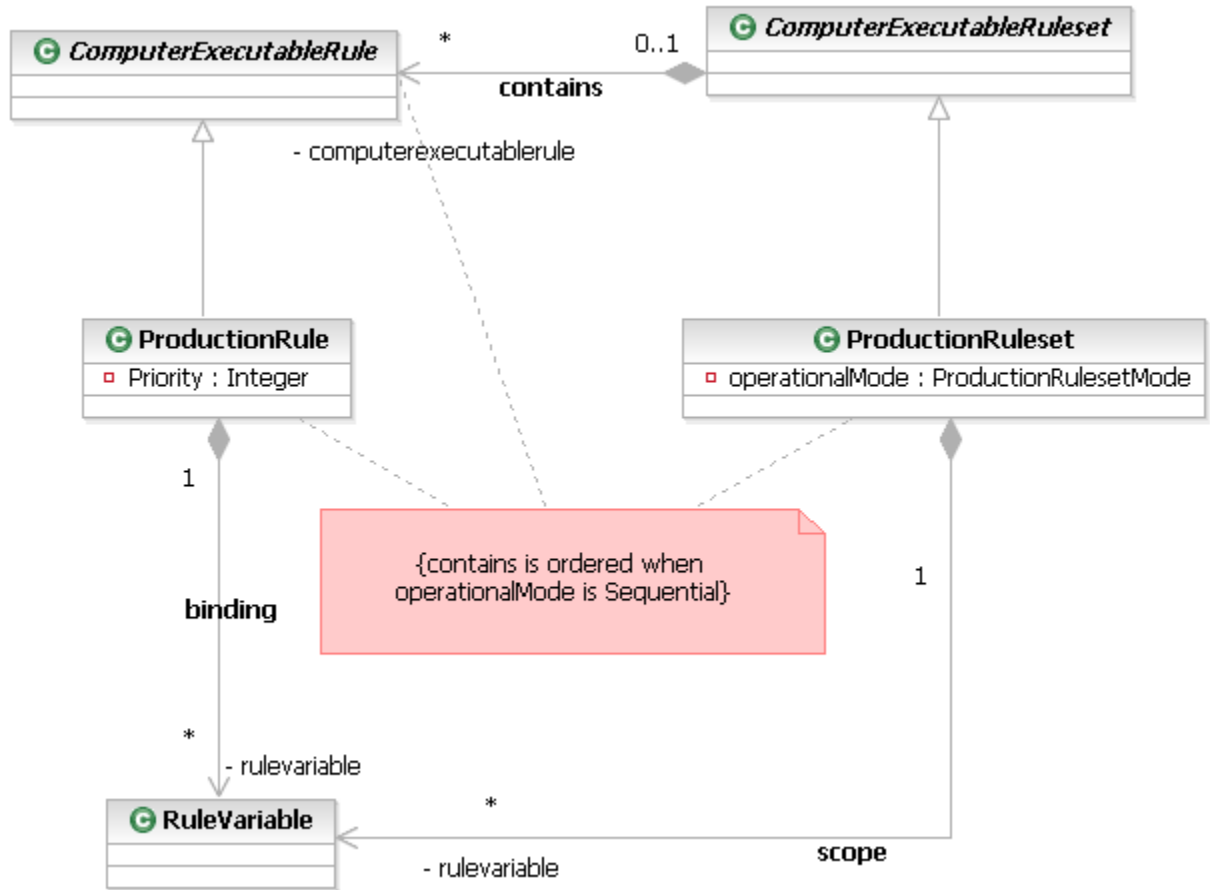


Figure 2: PRR ProductionRuleset Classes

### 3.4.3 Overview of PRR-Core Production Rule

The following is a partial model showing the ProductionRule class and its relationship to other model elements.

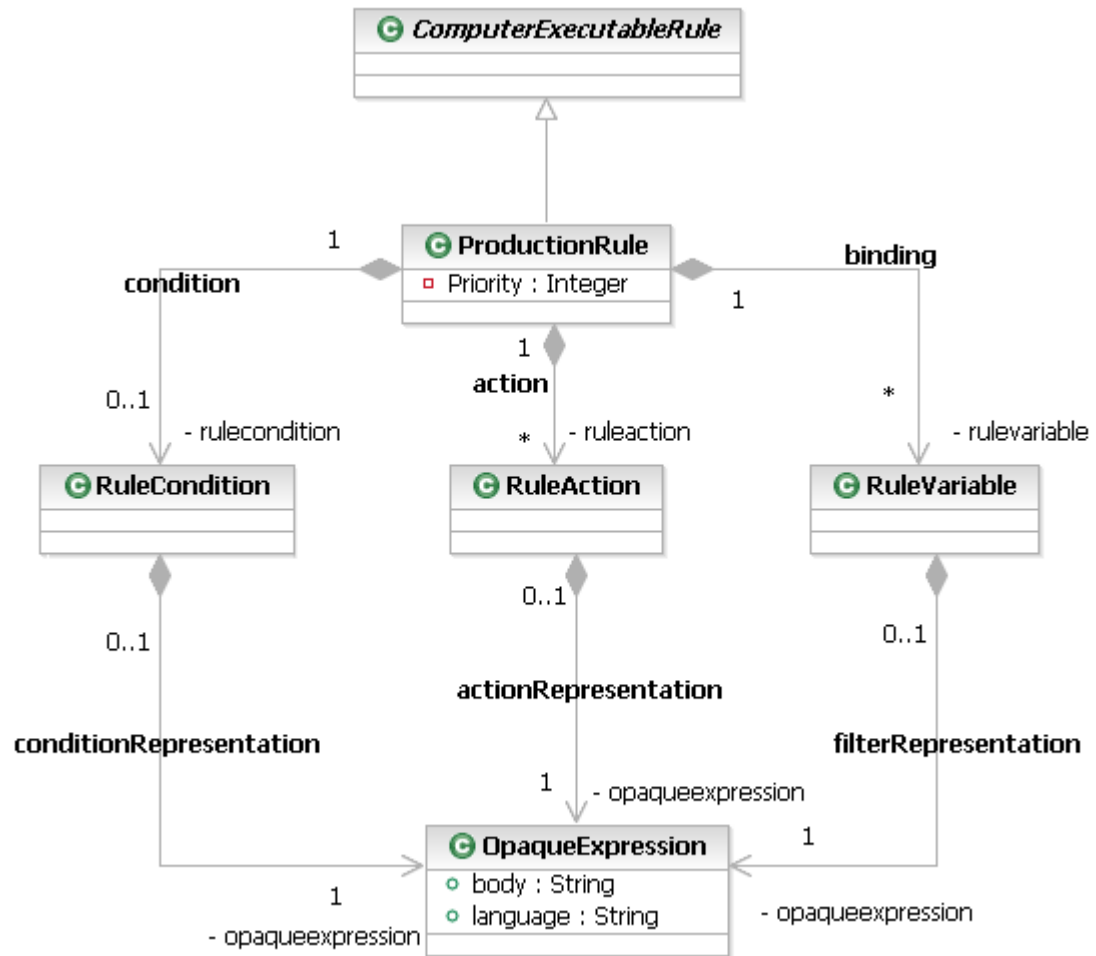


Figure 3: PRR ProductionRule Classes

### 3.4.4 Overview of PRR-Core RuleVariable

The following is a partial model showing the RuleVariable class and its relationship to other model elements.

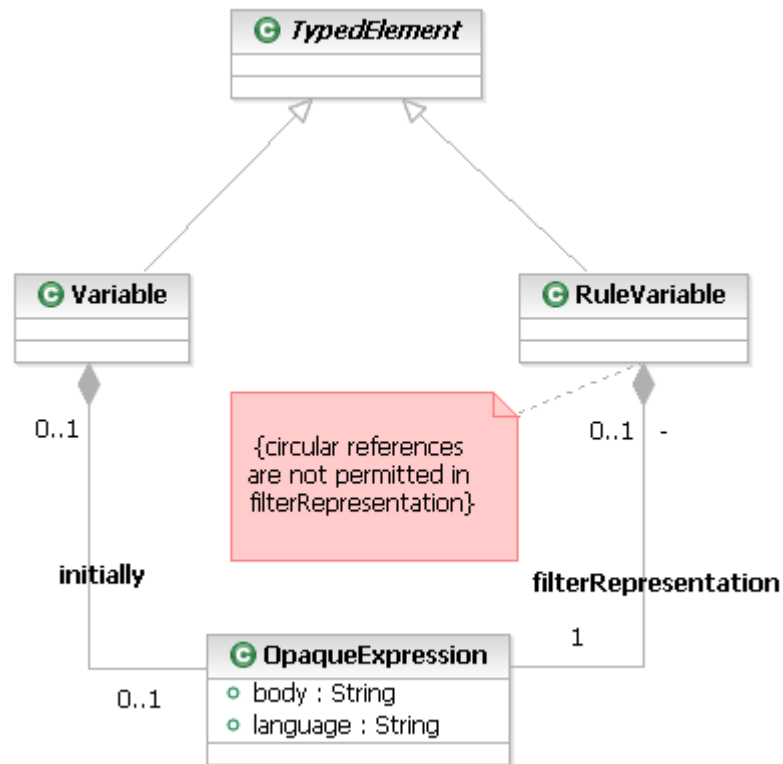


Figure 4: PRR RuleVariable Classes

### 3.4.5 Classes used in PRR Core

The following classes are used in these models. Each is defined below the table

Name	Description
RuleCore::ComputerExecutableRule	The computer executable rule represents a conditional piece of programmatic logic, including production rules. Future OMG standards may address other computer executable rule types such as event-condition-action rules, which would be derived from this class.
RuleCore::ComputerExecutableRuleset	The computer executable ruleset is a container for computer executable rules, and provides an execution context. In addition, a computer executable ruleset defines the interface for rule invocation, and the unit of execution in a rule engine; it is a Behavior in UML terms (or a service implementation in SOA terminology).
ProductionRule::RuleVariable	A RuleVariable defines a domain to be used in rule execution. The range of values that a rule variable can take may be further constrained by a filter expression...

## Submission to Production Rule Representation

RuleCore::Variable	The variable represents a programming construct to hold values for use in executing a ruleset. The values must conform to the variable's type.
ProductionRule::ProductionRule	A ProductionRule is a statement of programming logic that specifies the execution of one or more actions in the case that its conditions are satisfied. See section below.
ProductionRule::ProductionRuleset	The ProductionRuleset represents a ruleset for production rules
ProductionRule::RuleCondition	The condition represents a Boolean expression that is matched against available data to determine whether a ProductionRule can be instantiated
ProductionRule::RuleAction	The action association defines an ordered list of actions.
UML2::Kernel::NamedElement	See UML2
UML2::Kernel::TypedElement	See UML2
UML2::BasicBehaviors::Behavior	See UML2

### 3.4.6 Computer Executable Rule

The rule represents a conditional piece of programmatic logic, including production rules. Future OMG standards may address other rule types such as event-condition-action rules, which would be derived from this class. A Computer Executable Rule is a Named Element.

For additional information in PRR Core, associations to Classes used are provided. In PRR OCL and other variants with expression languages that refer to Class within OpaqueExpression, this is redundant.

#### 3.4.6.1 Attributes

None

#### 3.4.6.2 Associations

partOf:ComputerExecutableRuleset[0..1]

A rule may be part of a ruleset

modifies Class[\*]

The classes modified by the rule, e.g. within the OpaqueExpression for associated RuleAction. PRR Core only.

referenced class[\*]

The classes referred to by the rule, e.g. within the OpaqueExpression for associated RuleCondition. PRR Core only.

#### 3.4.6.3 Constraints

None

#### 3.4.6.4 Semantics

The semantics for **computer executable rules** are determined by their subtypes such as **production rules**.

### 3.4.7 Computer Executable Ruleset

The ruleset is a container for rules, and provides an execution context for rule execution. In addition, a ruleset defines the interface for rule invocation, and the unit of execution in a rule engine; it is a Behavior in UML terms (or a service implementation in SOA terminology) and so a Named Element.

#### 3.4.7.1 Attributes

None

#### 3.4.7.2 Associations

contains:ComputerExecutableRule[\*]

The rules contained in the ruleset.

scopedBy:Variable[\*]

The variables defined in the ruleset

#### 3.4.7.3 Constraints

None

#### 3.4.7.4 Semantics

The semantics for computer executable rulesets are determined by their subtypes such as production rulesets.

### 3.4.8 Variable

The variable represents a programming construct to hold values for use in executing a rule. The values must conform to the variable's type.

#### 3.4.8.1 Attributes

None

#### 3.4.8.2 Associations

initially:OpaqueExpression[0..1]

An optional expression specifying an initialization on the variable.

Scoped by:ComputerExecutableRuleset[0..1]

The variable may be part of the scope of a ruleset

#### 3.4.8.3 Constraints

None

#### 3.4.8.4 Semantics

The variable represents a typed element that is used in rule expressions as a substitute for an explicit object reference.

### 3.4.9 ProductionRuleset

The ProductionRuleset represents a ruleset for production rules.

#### 3.4.9.1 Attributes

operationalMode:enumeration{ProductionRulesetMode}

The operational semantics of the ruleset are described in its operationalMode attribute. The domain is open, but each model consumer (rule engine) will only understand a limited set of operational modes: this specification of PRR defines the semantics of rulesets with operation modes "Sequential" or "Forward Chaining".

#### 3.4.9.2 Associations

scope:RuleVariable[\*]

The list of RuleVariables that define the bindings in rule instantiation for all ProductionRule instances associated with the ProductionRuleset.

#### 3.4.9.3 Constraints

A ProductionRuleset may only contain ProductionRules

#### 3.4.9.4 Semantics

The ProductionRuleset defines the operational semantics of the production rules it contains via the operationalMode attribute. Generally rule execution cycle is defined in 3 stages, and is repeated until some state is met:

1. Match: identify eligible rules
2. Conflict resolution: rule selection per strategy
3. Act: change state per rule definition

The eligible rules are identified during the match step by binding their rule variables and checking their conditions' OpaqueExpression against specified data. All the instances of eligible rules, obtained by substituting the rule variables with the values within their domain, are considered for further processing. See section 3.2.5 "Semantics of Production Rules" on page 19 for further detail.

### 3.4.10 ProductionRule

A ProductionRule is a statement of programming logic that specifies the execution of one or more actions in the case that its conditions are satisfied.

The execution of a production rule will depend on the type of rule engine and the other rules in the ruleset in which it is contained.

The production rule is represented as:

for [rule variables] if [condition] then [action-list]

#### 3.4.10.1 Attributes

priority:integer

An optional attribute specifying the priority of a rule for use in determining the sequence of execution of Production Rules in Production Rulesets. Rules with higher priority values have higher priority than those with lower priority values.

## Submission to Production Rule Representation

### 3.4.10.2 Associations

condition:RuleCondition[0..1]

The rule condition that is required to be satisfied for the rule to be triggered.

action:RuleAction[\*]

The ordered list of actions that are executed when the rule is fired.

binding:RuleVariable[\*]

The list of RuleVariables that define the bindings in rule instantiation.

### 3.4.10.3 Constraints

There must be at least one RuleVariable or one RuleCondition specified.

### 3.4.10.4 Semantics

The operational semantics of production rules is defined in relation to the execution of the containing ruleset:

1. Given a set of objects assigned to its RuleVariables, the condition specifies whether the rule is eligible for execution / can be instantiated.
2. An instantiated rule can be chosen for execution (criteria being conflict resolution, strategy for execution sequencing, etc.), and if so, its actions are executed in order.

## 3.4.11 RuleCondition

The condition<sup>1</sup> represents a Boolean expression that is matched against available data to determine whether a ProductionRule can be instantiated. A tuple of RuleVariable values, known as a binding, defines a ProductionRule instance provided that with the binding the rule condition is satisfied. ProductionRule instances may be executed, subject to the operational mode of the containing ruleset. The condition filters the bindings that satisfy its expression, and then these values are used in the rule actions.

### 3.4.11.1 Attributes

None

### 3.4.11.2 Associations

conditionRepresentation:OpaqueExpression[1]

The expression specifying the rule condition.

### 3.4.11.3 Constraints

The OpaqueExpression evaluates to a Boolean result.

### 3.4.11.4 Semantics

The condition is used in the match step in the ProductionRuleset semantics, and gates the instantiation of the rules and the execution of the actions.

---

<sup>1</sup> Note that production rules are popularly defined in terms of multiple conditions (eg a set of Boolean expressions that include ANDs and ORs to create a single logical expression). For the purposes of PRR, we define that a condition in a ProductionRule is a single Boolean expression.

### 3.4.12 RuleAction

The action association defines an ordered list of actions. These actions may affect objects within the domain of a ruleset invocation (data source) or some external invocation.

#### 3.4.12.1 Attributes

None

#### 3.4.12.2 Associations

actionRepresentation: OpaqueExpression[\*]

The expression used to specify an action.

#### 3.4.12.3 Constraints

The actions form an ordered list.

#### 3.4.12.4 Semantics

When a rule is executed, the list of actions is executed in sequential order.

### 3.4.13 RuleVariable

The RuleVariable defines a domain to be used in rule execution. If nothing else is specified, its domain is the contents of the data source conforming to this type. Oftentimes, however, it is necessary to further restrict the domain of a rule variable (for example, if the data source contains different sets of objects with the same type, such as applicant: Person [\*], landlord: Person [\*], tenant: Person [\*], a rule variable with type Person would likely be restricted to one of these sets). The range of values that a rule variable can take may be further constrained by a filter expression.

#### 3.4.13.1 Attributes

None

#### 3.4.13.2 Associations

filterExpression: OpaqueExpression [\*] The expression used to specify a collection and/or filter for the domain represented by the RuleVariable

#### 3.4.13.3 Constraints

The filter expression for a Rule Variable must not create circular references through references to other Rule Variables.

#### 3.4.13.4 Semantics

At runtime, RuleVariables are used to specify the bindings that define applicable rule instances for specified values from the data source.