

OMG Production Rule Representation - Context and Current Status

Authors:

Said Tabet of RuleML: stabet@comcast.net
Gert Wagner of RuleML: G.R.Wagner@tm.tue.nl
Silvie Spreuwenberg of LibRT: silvie@librt.com
Paul Vincent of Fair Isaac Blaze Advisor: paulvincent@fairisaac.com
Gonzagues Jacques of ILOG JRules: gjacques@ilog.fr
Christian de Sainte Marie of ILOG: cma@ilog.fr
Jon Pellant of Pega Systems: jon.pellant@pega.com
Jim Frank of IBM: joachim_frank@us.ibm.com
Jacques Durand of Fujitsu: mmjdurand@us.fujitsu.com

Related OMG RFP: br/2003-09-03

Contents:

1.0	<i>Introduction</i>	2
2.0	<i>Architectural Context</i>	3
3.0	<i>PRR: Metamodel Issues</i>	5
4.0	<i>Current work</i>	7
5.0	<i>Conclusion</i>	8
6.0	<i>Appendix</i>	9

1.0 Introduction

1.1 Background: the Production Rule Representation RFP

The OMG Production Rule Representation (PRR) was developed from a RFP (Request For Proposals) developed in 2002-2003 to address the need for a representation of production rules in UML models (ie business rule modeling as part of a modeling process). This RFP was the 2nd business rule-related standard proposal from the “Business Enterprise Integration” task force of the OMG, which has taken over the charter of the “Business Rules Working Group” that was set up within OMG in 2002. The first RFP concerned the business semantics of business rules.

The PRR RFP, defined by some of the market-leading rule engine vendors such as Computer Associates, Fair Isaac and ILOG, solicited proposals for:

- An OMG MOF2-compliant¹ metamodel with precise dynamic semantics to represent production rules. This metamodel is to support a language that can be used with UML models to explicitly represent production rules as visible, separate and primary model elements in UML models.
- An XMI XML Schema Description (xsd) for production rules, based on the proposed metamodel, in order to support the exchange of production rules between modeling tools used for rule development.
- An example of a syntax that is compliant with the proposed metamodel for expressing production rules in UML models. This syntax will be considered non-normative.

1.2 Goals of the Production Rule Representation Standard

The Production Rule Representation, sponsored by a wide consortium of production rule interests, is to be proposed as a new OMG standard with the goals of:

- accelerating adoption of production rule components in everyday software systems
- improving the modeling of production rules, especially with respect to UML

¹ MOF: Meta-Object Facility

- allowing interoperability across different vendors providing production rule implementations.

2.0 Architectural Context

2.1 Relationship to the OMG Model-Driven Architecture

2.1.1 MDA

The Model-Driven Architecture (MDA) defines a model-driven approach to software development. An MDA specification consists of a definitive platform-independent base “UML model” (PIM), plus one or more platform-specific models (PSM) and interface definition sets, each describing how the base model is implemented on a different “platform”. The MDA also allows for an optional Business Model known as a CIM, or computation-independent model, that can be used as guidance to specify the PIM – an example of this is the Business Semantics for Business Rules OMG proposal. It is expected that elements in the Business Model / CIM will be mappable, through a standard transformation, to UML model elements such as the PRR at the PIM level, in conformance with the principles of the MDA.

2.1.2 Class of Platform

The target implementation platform for the PRR is the forward chaining rule engine of the types “Procedural Rule” and “Inference” Engines, hereafter described as “production rule engines”. The execution semantics are respectively referred to as "sequential rule processing" and "inferencing". The PRR defines a PIM for the production rule engine class of platform that can be subsequently transformed to a vendor-specific model (PSM) executable by a vendor-specific forward chaining rule engine.

Only production rules executed by a production rule engine are considered by the PRR.

RFP comment:

Note that this specification is a change over the RFP, which specified inference rule engines only (not procedural rule engines), and both forward and backward chaining rules. This change is in order to accommodate the industry requirements of the many vendors that do not use Inferencing technologies, and only a few use backward chaining techniques.

2.1.3 MDA layers

The PRR assumes the following usage of the MDA:

- The Business Model (or CIM): non-ambiguous representation of business policies, procedures, constraints as business rules in natural language and independent of assumptions regarding the platform on which an information system will be delivered.
- PIM: representation of production rules in UML targeted to the production rule engine class-of-platform that is independent of a vendor specific engine. The scope of the PRR is limited to this layer.
- PSM: representation of production rules in vendor-proprietary form executable by vendor-specific production rule engine.

Production rule engine vendors will be able to provide a mapping from the PRR PIM to the PSM specific to their products, depending on whether procedural or Inferencing rules are specified and whether they support those types. The means to implement the PSM models is provided by such production rule engine products.

The Business Model (CIM) layer – representation of business rules – is addressed by a separate RFP requesting business rule semantics for business users, OMG document br/03-06-03, Business Semantics of Business Rules RFP.

Future OMG RFPs would be expected to address other types of rule representation.

2.2 Differences with existing UML Languages

Some business rules have simple counterparts within a UML model, such as simple data constraints. For example, the business rule that “orders must have at least one line item” would typically be represented as a multiplicity constraint on an association. Other rules easily translate into constraint expressions. For example, the rules that “a birthday must be earlier than the current date” or that “an account can never have a negative balance” are easily expressed as invariants using OCL. On the other hand, another class of business rules does not seem to admit such simple representation by the available mechanisms of UML. Business rules that are expressed at the program level by production rules constitute a very large portion of this latter class. This is because it is not immediately clear how to represent (model) production rules in UML.

The two UML mechanisms for defining constraints and behaviour are the Object Constraint Language (OCL) and action semantics (AS). However, neither of these provides an “out-of-the-box” solution for representing production rules.

3.0 PRR: Metamodel Issues

3.1 Introduction

The OMG uses the concept of a metamodel to provide a standard representation. The MetaObject Facility (MOF) provides a UML-based mechanism for specifying metamodels. The PRR metamodel features:

- A definition of forward chaining production rules for Rete-based inference and procedural processing.
- A definition for rule condition and action expressions, that can also be replaced by alternative representations for vendor-specific useage.
- A definition of rulesets, that are collection of rules that are defined for their class of platform (procedural or inference rule engine).

As other rule types (such as backward chaining deduction rules, event-condition-action rules, and so forth) would expect to be considered in future extensions to this standard. To this end, the PRR is designed to be extensible.

3.2 Production Rules

From the RFP, a production rule 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 transition system formalism).

There are 2 types of production rule that need to be modeled for the PRR, based on whether the production rule is:

- executed in a forward chaining rule engine, such as Rete-based rule engines, in an order-independent manner.
- executed in a procedural rule engine, in an order-dependent sequential manner.

3.2.1 Forward-chaining Inference rules

A forward chaining inference rule is a production rule defined **without** respect to execution ordering, as execution ordering is under the control of the inference engine.

The production rule is typically¹ represented as:

¹ If.. then.. rules are sometimes represented as when... then... rules by some vendors.

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

The condition-list defines the rule conditions, which in turn define:

- bindings that define data tuples from the provided context defined by a local object model, and referenced in the rule actions; and
- condition expressions that constrain the data tuples to some subset of the current context.

The action-list defines the:

- rule actions, which are expressions that define behavior and thence change the state of the system. These expressions are defined in terms of the bindings, and executed only for those tuples that satisfy the rule condition constraints at the time of rule processing.

The execution semantic for the processing of a forward chaining inference rule, for example, can be stated as:

- bindings represent the common data for the LHS and RHS of the rule; these are akin to variable declarations but may also represent a Class or Collection as well as a single object
- the condition returns a list of tuples (each containing an instance of each binding) that match the expressions specified in the condition, or represents a truth value if it is a logical expression
- the (list of) action statements are then processed against each member of the list of tuples.

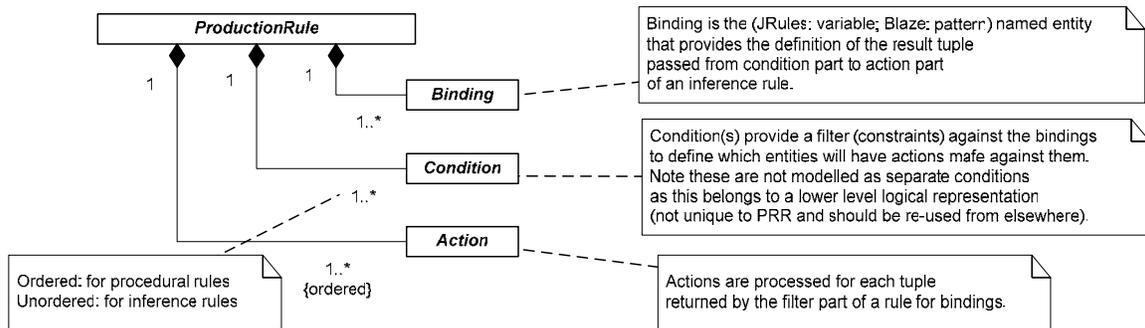


Figure 1: Abstract production rule (Revision 3)

3.2.2 Forward-chaining sequential production rules

A forward chaining procedural production rule is defined **with** respect to execution ordering, as rules are executed per some predefined order. Such rules can also be specified in terms of bindings, conditions and actions, as for declarative inference rules. The abstract production rule in Figure 2 applies also to the sequential production rule case, with the slight difference on the specification of ordering of rule conditions too.

3.3 Organization of Rules

3.3.1 Rulesets

Rulesets provide for organizing rules into structures. Although rule membership of a ruleset is exclusive, rulesets can be aggregated.

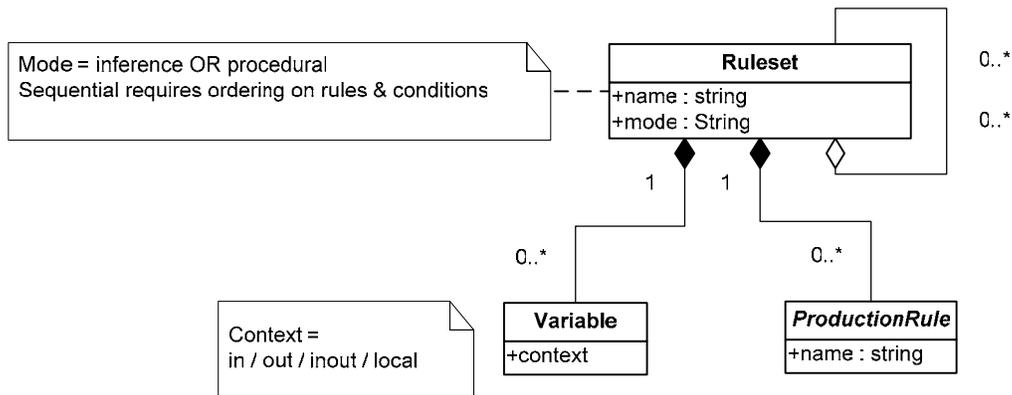


Figure 2: Ruleset Model (Revision 2)

4.0 Current work

Tasks remaining include:

- conclusion of a default rule condition and action expression language, ideally based on an OMG standard such as OCL2 or action semantics
- specification of the XMI schema
- specification of an example UML diagramming notation for business rule representation in UML diagrams.

Examples of rules and rule mappings are included in appendix 1, below.

A revised submission for the PRR is due in April, with a completed initial submission planned for the Summer of 2005.

5.0 Conclusion

Although a rule language for interoperability across the web is not in the scope of the PRR meta-model specification, we see compatibility with any future standard language for exchanging rules on the Web as an absolute requirement.

The PRR specification team aims to work with any forthcoming XML-based rule interchange standard (RuleML being currently the main proponent) to build a single unified Production Rule Representation standard that can satisfy the needs of rule modeling, XML-based rule interchange and commercial use by business rule engines in industry.

6.0 Appendix

6.1 Example Rule Mappings

Examples of different representations are used to indicate metamodel compliance and issues for the PRR. The following represents a subset of the examples collected as part of the PRR team's work on preparing a metamodel that fits existing real-world use of production rules across industry, per the vendors and organizations involved.

6.2 Rule Definitions

This shows examples of rule information without conditions and actions (see below). For the most part, the only rule definition aspect the PRR is concerned with is the rule name. However, common Production Rule representations are investigated below for completeness.

Case #	Description	Format	Example	Comments
1.1	Standard rule definition statement.	Blaze SRL	rule bestDiscount is if ... then ...	
		JRules	rule cheapPurchases { when ... then ... }	
		RBML	<pre><rbml:Rule id="cheapPurchases" name="cheapPurchase"> <rbml:Scope idref="Order"/> <rbml:RuleDef> <rbml:Premise> ... </rbml:Premise> <rbml:Action> ... </rbml:Action> </rbml:RuleDef> <rbml:Binding id="b_order" name="order"> <rbml:Class idref="Order"/> </rbml:Binding> </rbml:Rule></pre>	
		IBM	<pre>... <rule name="simpleDiscountRule"> <clause ...> ... </clause></pre>	

			<pre> <block name="cheapPurchases" gatingCases="lowValue"> ... </block> ... </rule> ... </pre>	
		RuleML		

6.3 Rule Conditions

The following table provides sample conditions in a variety of production rule representations, and possible PRR representations.

Case #	Description	Format	Example	Comments
2.1a	For orders under \$100.00 a discount of 3% should be applied, for orders over \$100.00 but under \$1000.00 the discount should be 6%, for orders worth \$1000.00 or more the discount should be 10%. Assume order is a variable (local or global) or ruleset parameter	Blaze SRL	If order < 100 then... If order >= 100 and order < 1000 then... If order >= 1000 then...	
		JRules	<pre> when { IlrContext() from ?context; evaluate(orderValue <100); } then ... when { IlrContext() from ?context; evaluate(orderValue >=100 && orderValue <1000); } then ... when { IlrContext() from ?context; evaluate(orderValue >=1000); } then ... </pre>	
		RBML	<pre> ... <rbml:Premise> <rbml:Condition id="Rule1.Condition1"> <rbml:BinaryLogicalExpression operator="lt"> <rbml:LeftHand> <rbml:AttributeReference> <rbml:Qualifier> </pre>	

W3C Workshop on Rule Languages for Interoperability

			<pre> <rbml:Binding idref="b_order"/> </rbml:Qualifier> <rbml:Attribute idref="orderValue"/> </rbml:AttributeReference> </rbml:LeftHand> <rbml:RightHand> <rbml:Literal freeformat="100"/> </rbml:RightHand> </rbml:BinaryLogicalExpression> </rbml:Condition> </rbml:Premise> ... <rbml:Premise> <rbml:Premise> <rbml:Condition id="Rule2.Condition1"> <rbml:BinaryLogicalExpression operator="ge"> <rbml:LeftHand> <rbml:AttributeReference> <rbml:Qualifier> <rbml:Binding idref="b_order"/> </rbml:Qualifier> <rbml:Attribute idref="orderValue"/> </rbml:AttributeReference> </rbml:LeftHand> <rbml:RightHand> <rbml:Literal freeformat="100"/> </rbml:RightHand> </rbml:BinaryLogicalExpression> </rbml:Condition> <rbml:BinaryLogicalOperator value="and"> <rbml:Condition id="Rule2.Condition2"> <rbml:BinaryLogicalExpression operator="lt"> <rbml:LeftHand> <rbml:AttributeReference> <rbml:Qualifier> <rbml:Binding idref="b_order"/> </rbml:Qualifier> <rbml:Attribute idref="orderValue"/> </rbml:AttributeReference> </rbml:LeftHand> <rbml:RightHand> <rbml:Literal freeformat="1000"/> </rbml:RightHand> </rbml:BinaryLogicalExpression> </rbml:Condition> </rbml:BinaryLogicalOperator> </rbml:Premise> </rbml:Premise> ... </pre>	
--	--	--	--	--

			<pre> <rbml:Premise> <rbml:Condition id="Rule1.Condition1"> <rbml:BinaryLogicalExpression operator="ge"> <rbml:LeftHand> <rbml:AttributeReference> <rbml:Qualifier> <rbml:Binding idref="b_order"/> </rbml:Qualifier> <rbml:Attribute idref="orderValue"/> </rbml:AttributeReference> </rbml:LeftHand> <rbml:RightHand> <rbml:Literal freeformat="1000"/> </rbml:RightHand> </rbml:BinaryLogicalExpression> </rbml:Condition> </rbml:Premise> ... </pre>	
		IBM	<pre> <clause caseVariable="orderValue"> <if condition="orderValue < 100" case="lowValue"/> <if condition="orderValue < 1000" case="mediumValue"/> <otherwise case="highValue"> </clause> </pre>	
		RuleML		

6.4 Rule Actions

The following table provides sample actions in a variety of production rule representations.

Case #	Description	Format	Example	Comments
3.1	For orders under \$100.00 a discount of 3% should be applied, for orders over \$100.00 but under \$1000.00 the discount	Blaze SRL	<pre> if ... then { discount = 3 } if ... then { discount = 6 } </pre>	

W3C Workshop on Rule Languages for Interoperability

<p>should be 6%, for orders worth \$1000.00 or more the discount should be 10%.</p> <p>Assume discount is a variable (local or global) or ruleset returned value</p>		<pre>if ... then { discount = 10 }</pre>
	JRules	<pre>when {...} then { discount = 3.0; updateContext(); } when {...} then { discount = 6.0; updateContext(); } when {...} then { discount = 10.0; updateContext(); }</pre>
	RBML	<pre>... <rbml:Action> <rbml:AttributeValueAssignment> <rbml:AttributeReference> <rbml:Attribute idref="discount"/> </rbml:AttributeReference> <rbml:TypedExpression> <rbml:Literal freeformat="3"/> </rbml:TypedExpression> </rbml:AttributeValueAssignment> </rbml:Action> ... <rbml:Action> <rbml:AttributeValueAssignment> <rbml:AttributeReference> <rbml:Attribute idref="discount"/> </rbml:AttributeReference> <rbml:TypedExpression> <rbml:Literal freeformat="6"/> </rbml:TypedExpression> </rbml:AttributeValueAssignment> </rbml:Action> ... <rbml:Action> <rbml:AttributeValueAssignment> <rbml:AttributeReference> <rbml:Attribute idref="discount"/> </rbml:AttributeReference> <rbml:TypedExpression> <rbml:Literal freeformat="10"/> </rbml:TypedExpression> </rbml:AttributeValueAssignment> </rbml:Action> ...</pre>
IBM	<pre>... <block name="cheapPurchases" gatingCases="lowValue"></pre>	

		<pre> <assign target="discount" value="3.0"/> </block> <block name="averagePurchases" gatingCases="mediumValue"> <assign target="discount" value="6.0"/> </block> <block name="expensivePurchases" gatingCases="highValue"> <assign target="discount" value="10.0"/> </block> ... </pre>	
		RuleML	

6.5 Rule Variables and Patterns

The following table provides sample definitions of in-rule variable and bindings used in conditions and/or actions, within the context of an individual rule. These are provided in a variety of production rule representations.

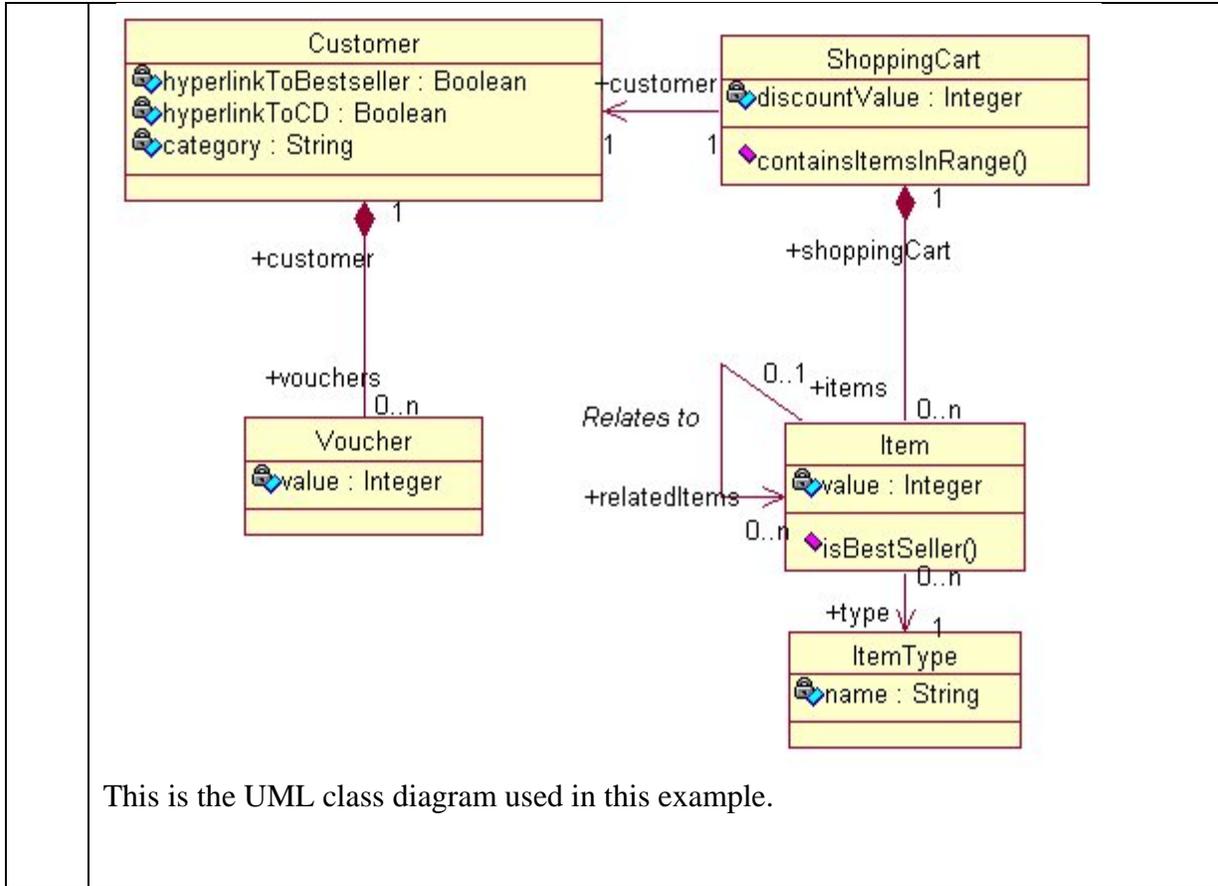
Case #	Description	Format	Example	Comments
4.1	Same rule as 1.1 For orders under \$100.00 a discount of 3% should be applied, for orders over \$100.00 but under \$1000.00 the discount should be 6%, for orders worth \$1000.00 or	Blaze SRL	<pre> order is any Orders such that type = "Goods". ... Rule ... if order.amount < 100 then... Rule ... if order.amount >= 100 and order.amount < 1000 then... Rule ... if order.amount >= 1000 then... </pre>	
		JRules		
		RBML		
		IBM	<pre> <parameter name="orderValue" type="Real" direction="in" /> <parameter name="discount" type="Real" direction="return" /> <block> <ruleGroup> <rule name="simpleDiscountRule"> <clause caseVariable="orderValue"> <if case="lowValue"> <condition>orderValue <lt; 100</condition> </if> <if case="mediumValue"> <condition>orderValue <lt; 1000</condition> </if> <else case="highValue" /> </clause> </pre>	

W3C Workshop on Rule Languages for Interoperability

	<p>more the discount should be 10%.</p> <p>Assume order is a pattern across a class Orders, with some constraint.</p> <p>Specify means of defining pattern.</p>		<pre> <block name="cheapPurchases" gatingCases="lowValue"> <assign target="discount">3.0</assign> </block> <block name="averagePurchases" gatingCases="mediumValue"> <assign target="discount">6.0</assign> </block> <block name="expensivePurchases" gatingCases="highValue"> <assign target="discount">10.0</assign> </block> </rule> </ruleGroup> </block> </pre>	
		RuleML		
		OCL	<p>Context Orders inv: Orders.allInstances() -> forAll(o1 o1.type == "Goods" and o1.amount < 100 implies (...)</p> <p>Context Orders inv: Orders.allInstances() -> forAll(o1 o1.type == "Goods" and o1.amount >= 100 and o1.amount < 1000 implies (...)</p> <p>Context Orders inv: Orders.allInstances() -> forAll(o1 o1.type == "Goods" and o1.amount >= 1000 implies (...)</p>	
4.5	<p>Shopping cart rule</p> <p>If the shopping cart contains between 2 and 4 items and either the purchase value is greater than \$100 and the</p>	Blaze SRL	<pre> anyShoppingCart is any ShoppingCart. rule discount is if anyShoppingCart.items.count is between 2 and 4 and ((anyShoppingCart.items.value.sum > 100 and anyShoppingCart.customer.category = "Gold") or (anyShoppingCart.items.value.sum > 200 and anyShoppingCart.customer.category = "Silver")) then { anyShoppingCart.discountValue = shoppingCart.discountValue + 15); } </pre>	

W3C Workshop on Rule Languages for Interoperability

<p>customer category is gold or the purchase value is greater than \$200 and the customer category is Silver then apply a 15% discount on the shopping cart value.</p>	JRules	<pre> rule discount { when { ?customer: Customer(); ?shoppingCart: ShoppingCart(customer == ?customer); evaluate((?shoppingCart.containsItemsInRange(2, 4)) && (((((?shoppingCart.getValue() > 100d) && (?customer.category equals "Gold")) ((?shoppingCart.getValue() > 200d) && (?customer.category equals "Silver"))))))); } then { modify ?shoppingCart { shoppingCart.discountValue = shoppingCart.discountValue + 15f); } } </pre>
	RBML	
	IBM	
	RuleML	
	OCL	



6.6 Ruleset Definitions

This shows examples of ruleset information without its contained rules. These are provided in a variety of production rule representations.

Case #	Description	Format	Example	Comments
5.1	Ruleset definition for example in 1.1, specifying parameters to be used in rule conditions and the output returned.	Blaze SRL	<pre> ruleset calculateDiscount for { order: an integer} returning an integer is { discount is an integer. ... if discount is known then {return discount} } // rulesets can only return 1 value (or object) // the rules may be written to return a value instead of the rule returning discount. </pre>	

W3C Workshop on Rule Languages for Interoperability

		JRules	<pre>ruleset simpleDiscountRuleset { in double orderValue; out double discount; }</pre>
		RBML	
		IBM	<pre><ruleSet name="simpleDiscountRuleSet"> <parameter name="orderValue" type="Real" direction="in"/> <parameter name="discount" type="Real" direction="return"/> ... </ruleSet></pre>
		RuleML	