The Abstract Syntax of RuleML – Towards a General Web Rule Language Framework

Gerd Wagner[1], Grigoris Antoniou[2], Said Tabet[3], and Harold Boley[3,4]
[1]*Dept. TechnologyManagement, Eindhoven University of Technology, The Netherlands*
[2]*Institue of Computer Science, FORTH, Greece*
[3]*Co-chair of the RuleML Initiative, http://www.ruleml.org*
[4]*National Research Council, Fredericton, NB, Canada*
G.Wagner@tm.tue.nl

Abstract

This paper discusses the approach taken by the Rule Markup Language (RuleML) Initiative towards a general Web rule language framework and relates it to the MDA and UML by the Object Management Group (OMG). It also presents the abstract syntax of RuleML 0.85 as a MOF/UML model and considers the possibility to integrate RuleML with OCL and Action Semantics.

1. Introduction

The Model Driven Architecture (MDA)[1] is a framework for software development defined by the OMG. MDA can be regarded as an evolutionary approach to bringing models as first-class citizens into the software engineering process. It is based on a fundamental distinction between three different modeling levels: the level of semi-formal (computation-independent) business domain modeling, the level of platform-independent logical design modeling, and the level of platform-specific implementation modeling.

Rule markup languages will be the vehicle for using rules on the Web and in other distributed systems. They allow deploying, executing, publishing and communicating rules on the Web. They are also converging towards a lingua franca for exchanging rules between different systems and tools.

In a narrow sense, a rule markup language is a concrete (XML-based) rule syntax for the Web. In a broader sense, it should have an abstract syntax as a common basis for defining various concrete sublanguages serving different purposes. The goal of RuleML is to permit reusability and interchange at a higher level, similar to the idea behind MDA. Instead of creating yet another rule language, RuleML has offered a family of modular sublanguages on top of a shared data model that can be leveraged by existing and future language instantiations ([2] [1]).

In the present paper we consider rules at three different abstraction levels:

1. At the business domain level, rules are statements that express (certain parts of) a business/domain policy (e.g., defining terms of the domain language or defining/constraining domain operations) in a declarative manner, typically using a natural language or a visual language. Examples are:

   (R1) "The driver of a rental car must be at least 25 years old"

   (R2) "A gold customer is a customer with more than $1Million on deposit"

   (R3) "An investment is exempt from tax on profit if the stocks have been bought more than a year ago"

   (R4) "When a share price drops by more than 5% and the investment is exempt from tax on profit, then sell it"

   R1 is an *integrity rule*, R2 and R3 are *derivation rules*, and R4 is a *reaction rule* (see below for explanations of these rule categories). These appear to be the major semantic categories of business rules. Actually, many business rules appear to be reaction rules, which specify policies for real-world business behavior.
2. At the platform-independent level, rules are formal statements, expressed in some formalism or computational paradigm, which can be directly mapped to executable statements of a software platform. Rule languages used at this level are SQL:1999, OCL 2.0, and ISO Prolog. Remarkably, SQL provides operational constructs for all three business rule categories mentioned above: *checks/assertions* operationalize a notion of constraint rules, *views* operationalize a notion of derivation rules, and *triggers* operationalize a notion of reaction rules.
3. At the platform-specific level, rules are statements in a specific executable language, such as Oracle 10g views, Jess 3.4, XSB 2.6 Prolog, or the Microsoft Outlook 6 Rule Wizard.

Generally, rules are self-contained knowledge units that involve some form of reasoning. They may, for instance, specify

- static or dynamic integrity constraints
- derivations (e.g. for defining derived concepts),
- reactions (for specifying the reactive behavior of a system in response to events)

[1]See http://www.omg.org/cgi-bin/doc?mda-guide.

Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI'04)
0-7695-2100-2/04 $ 20.00 IEEE

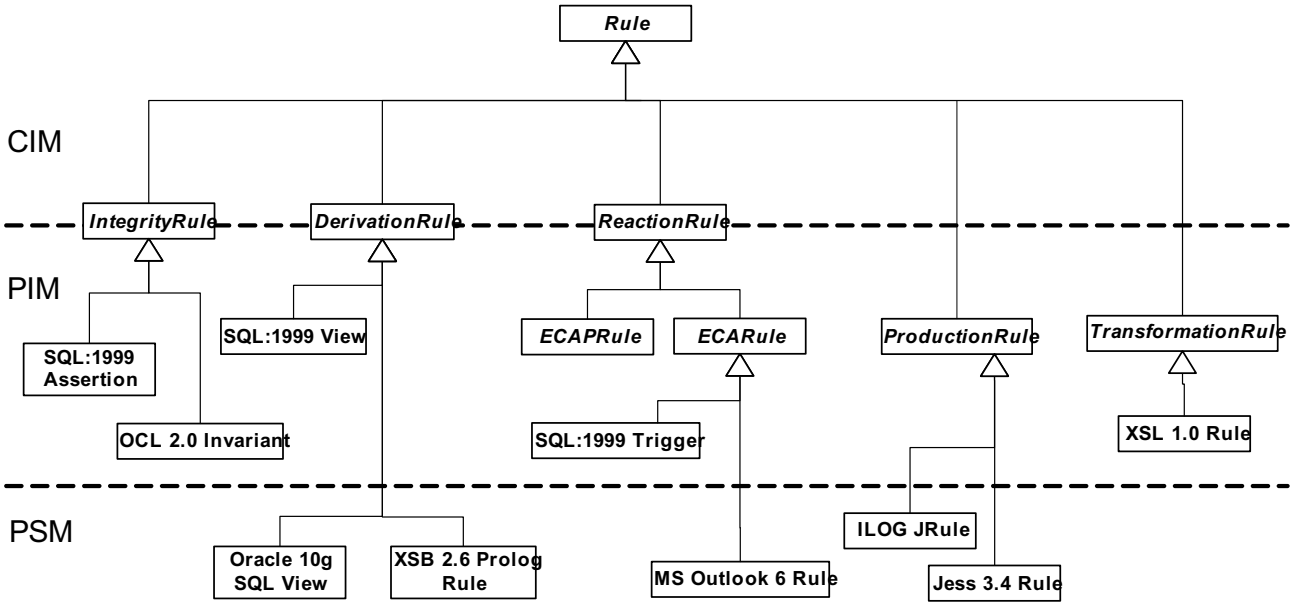

Figure 1: Rule concepts and rule expressions at different levels of abstraction.

Given the linguistic richness and the complex dynamics of business domains, it should be clear that any specific mathematical account of rules, such as classical logic Horn clauses, must be viewed as a limited descriptive theory that captures just a certain fragment of the entire conceptual space of rules, and not as the only definitive, normative account. Rather, we need a pluralistic approach to the heterogeneous conceptual space of rules. Therefore, in RuleML, a family of rule languages capturing the most important types of rules is being defined. While these languages come with a recommended formal semantics, some of their rule bases may be marked to have a variant acceptable semantics. This will accommodate various formalisms based on non-standard logics, supporting temporal, fuzzy, defeasible, and other forms of reasoning.

In this paper, we discuss the abstract syntax of the Rule Markup Language (RuleML). The abstract syntax of a language can be defined with the help of a MOF/UML model, as recommended by the OMG, or it can be defined with the help of a suitably general grammar definition language such as the EBNF formalism used in the definition of the abstract syntax of OWL and SWRL. Figure 4 shows the syntax of abstract derivation rules in the form of a MOF/UML model. The different 'parts' of RuleML rules and their relationships are defined in MOF without specifying any concrete symbols for their serialization. In particular, the abstract syntax specifies the major role parts of rules and their semantic categories. The concrete syntactic ordering is left undefined; e.g., there is no bias towards a prefix notation (such as in Jess) or an infix notation (such as in RDF).

2. Categorizing Rules

The main categories of rules considered in RuleML are derivation rules, integrity rules (constraints), reaction rules, production rules and transformation rules, as depicted in Figure 1. We consider the concepts of derivation rules, integrity constraints, and reaction rules to be meaningful both as (computation-independent) business rule categories and as (platform-independent) computational rule categories, whereas the concepts of production rules and transformation rules appear to be only meaningful as computational rule categories.

Notice that those categories whose name is in italics, such as DerivationRule, refer to an abstract concept of rule, while the others (with non-italicized names), such as "SQL:1999 View", refer to rule concepts of concrete languages such as SQL:1999.

The main link between the different types of rules is the notion of a LogicalFormula or of a LogicalSentence, one of which being used in all of them. Traditionally, logical formulas are expressed in a language based on a predicate logic signature. However, OCL is the language of choice for expressing logical formulas referring to the state of a system whose structure is defined by a UML class model.

3. Integrity Rules

Integrity rules, also known as (integrity) constraints, consist of a logical sentence (in some logical language such as predicate logic or temporal logic). They express assertions that must hold in all evolving states and state transition histories of the discrete dynamic system for which they are defined.

Rule R1 from the introduction is an example of a static constraint. An example of a dynamic constraint rule is: "The confirmation of a rental reservation must lead to an allocation of a car of the requested car group for the requested date prior to that date".

Well-known languages for expressing constraints are SQL and OCL. In logic programming, rules with empty heads (also called "denials") corresponding to the negation of the conjunction of all body atoms are sometimes used as constraints. It is an important issue for RuleML whether to add a direct notion of constraints in future versions

The enforcement of constraints can be implemented with the help of ECA rules whose event condition refers to state changes that would violate the constraint and whose action would be an alert or some kind of repair action. In a DBMS, the implementation of constraint rules by means of triggers is normally more efficient than their implementation by means of declarative assertions.

4. Derivation Rules

Derivation rules, in general, consist of one or more conditions and one or more conclusions[2], which are both roles played by expressions of the type LogicalFormula.

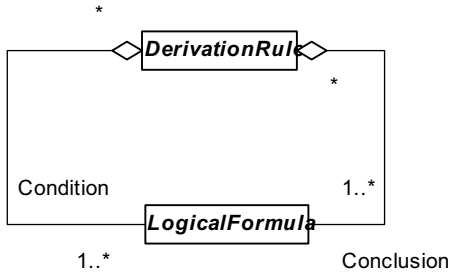

Figure 2: The general format of derivation rules.

For specific types of derivation rules, such as definite Horn clauses or normal logic programs, the types of condition and conclusion are specifically restricted. In RuleML 0.85, conditions are quantifier-free logical formulas with weak and strong negation, called *QF-Formula* in Figure 3. More precisely, they are quantifier-free predicate logic formulas with weak and strong negation, called *QF-PL-Formula* in Figure 4 (the QF-PL-Formula class specializes the abstract class *QF-Formula*, which admits also of other kinds of atoms such as OCL or OWL atoms, by restricting it to predicate logic atoms).

The distinction between weak and strong negation, although well-established in extended logic programs, may be not familiar to people trained in classical logic.

[2] Notice that we don't consider rules with no condition or no conclusion. These expressions are better not called "rules", but "facts" and "denial constraints".

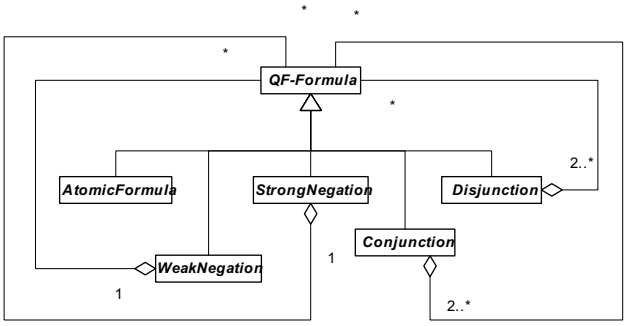

Figure 3: Quantifier-free formulas with weak and strong negation.

Intuitively speaking, weak negation captures the absence of positive information, while strong negation captures the presence of explicit negative information. Under the preferential model semantics of minimal/stable models, weak negation captures the computational concept of *negation-as-failure* (or *closed-world* negation).

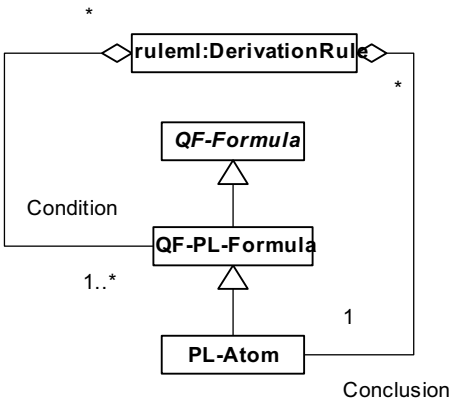

Figure 4: RuleML 0.85 derivation rules: there are one to many conditions, each being a quantifier-free predicate logic formula, and there is exactly one conclusion, being a predicate logic atom.

In RuleML 0.85, a derivation rule has exactly one conclusion, which takes the form of a predicate logic atom, called PL-Atom in Figure 4 and Figure 5.

The rules R2 and R3 from the introduction are examples of derivation rules.

5. Reaction Rules

Reaction rules are the second important type of rule in RuleML. Integrity and transformation rules have not received as much attention as derivation and reaction rules. Reaction rules are considered to be the most important type of business rule in [3]. They consist of a mandatory triggering event term, an optional condition, and a triggered action term or a post-condition (or both), which are roles of type EventTerm, LogicalFormula, ActionTerm, and LogicalFormula, respectively, as shown in Figure 6.

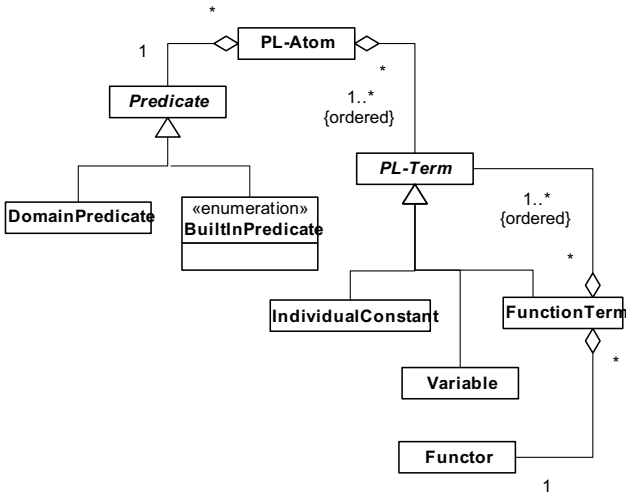

Figure 5: The abstract syntax of predicate logic atoms.

While the condition of a reaction rule is, exactly like the condition of a derivation rule, a quantifier-free formula, the post-condition is restricted to a conjunction of possibly negated atoms.

Action and event terms may be composite and specified in different ways. For instance, the UML Action Semantics could be used to specify triggered actions in a platform-independent manner.

There are basically two types of reaction rules: those that do not have a post-condition, which are the well-known *Event-Condition-Action (ECA)* rules, and those that do have a post-condition, which we call *ECAP rules*.

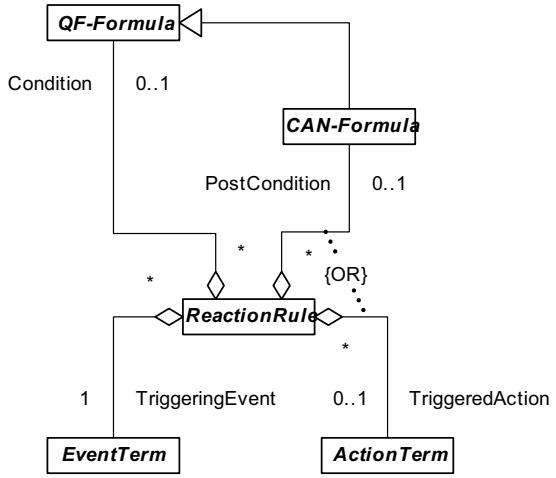

Figure 6: The general format of reaction rules.

The post-condition of a reaction rule is either an atomic formula, a negation of an atomic formula or a conjunction of these. This is called a CAN-Formula in Figure 7. Such a definite formula specifies an update in a declarative way.

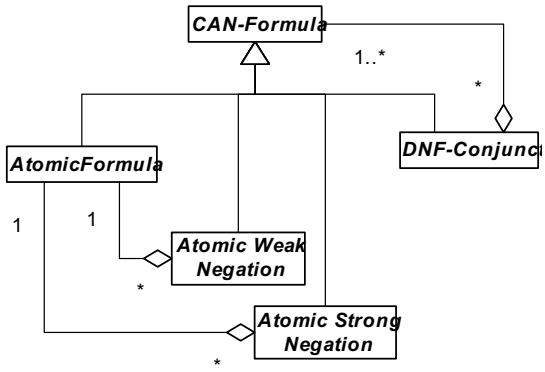

Figure 7: A CAN-Formula corresponds to a disjunctive normal form conjunct, that is, it is a conjunction of possibly (weakly or strongly) negated atomic formulas.

6. Production Rules

Production rules consist of a condition and an action term. They have become popular as a widely used technique to implement 'expert systems' in the 1980s. However, in contrast to (e.g. Prolog) derivation rules, the production rule paradigm lacks a precise theoretical foundation and does not have a formal semantics. This problem is partly due to the fact that the semantic categories of events and conditions in the left-hand-side, and of actions and effects in the right-hand-side of a rule, are mixed up.

7. Conclusion and Future work

As we are exploring the integration of RuleML with the business rules effort at OMG, we are investigating the use of OCL for expressing atomic sentences in RuleML conditions and the OMG Actions Semantics for expressing actions in RuleML production and reaction rules.

References

[1] Boley, H.: The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations, Invited Talk, INAP2001, Tokyo, Springer-Verlag, LNCS 2543, 5-22, 2003.

[2] Boley, H., Tabet, S., Wagner G.: Design Rationale of RuleML: A Markup Language for Semantic Web Rules. International Semantic Web Working Symposium (SWWS), June 2001, Stanford, USA.

[3] Taveter K., Wagner, G.: Agent-Oriented Enterprise Modeling Based on Business Rules. In Proc. of 20th Int. Conf. on Conceptual Modeling (ER2001), Springer-Verlag, LNCS 2224, pp. 527–540, 2001.