Design Rationale of RuleML: A Markup Language for Semantic Web Rules

Harold Boley[1], Said Tabet[2] and Gerd Wagner[3]
1. DFKI GmbH, Erwin-Schrödinger-Straße D-67663 Kaiserslautern Germany
2. Nisus, Inc. 180, Turnpike Road Westboro, MA 01581 USA
3. Faculty of Technology Management Eindhoven University of Technology P.O. Box 513 5600 MB Eindhoven The Netherlands
boley@dfki.de, stabet@nisusinc.com, G.Wagner@tm.tue.nl

Abstract. This paper lays out the design rationale of RuleML, a rule markup language for the Semantic Web. We give an overview of the RuleML Initiative as a Web ontology effort. Subsequently, the modular syntax and semantics of RuleML and the current RuleML 0.8 DTDs are presented (focusing on the Datalog and URI sublanguages). Then we discuss negation handling, priorities/evidences, as well as agents and RuleML. We next proceed to RuleML implementations via XSLT and rule engines. In our conclusions, we continue to explore the bigger picture of ontologies and discuss some requirements for a future RuleML. An appendix shows our Semantic Web scenario in the insurance industry.

1. Introduction

Rules have traditionally been used in theoretical computer science, compiler technology, databases, logic programming, and AI. The Semantic Web tries to represent information in the World Wide Web such that it can be used by machines not just for display purposes, but for automation, integration, and reuse across applications; it has recently advanced to a W3C Activity. *Rule Markup* for the Semantic Web has been a hot topic since rules were identified as one of its Design Issues.

However, Semantic Web rules have been less systematically studied than the corresponding ontology (actually, taxonomy) markup. The Rule Markup Initiative tries to fill the gap by exploring rule systems (e.g., extended Horn logics) suitable for the Web, their (XML and RDF) syntax, semantics, tractability/efficiency, and transformation and compilation. Both derivation rules (which may be evaluated bottom-up as in deductive databases, top-down as in logic programming, or by tabled resolution as in XSB) (10) and reaction rules (also called "ECA" -- "event-condition-action" -- or "trigger" rules), as well as possible combinations, are being considered.

In the context of the Semantic Web, rules may be built on F-logic for RDF inference, as pioneered by SiLRI (4). This work has recently been extended for rules with expressive bodies (full FOL syntax) in TRIPLE (5). Rules may also be used to enhance the content of Web pages and XML documents in various ways. E.g., derivation rules allow the dynamic inclusion of derived facts, while reaction rules allow the specification of behavior in response to browser events.

RuleML started on the basis of pre-existing rule markup languages and has already inspired further rule-markup projects. As examples, we just sketch our RFML, URML, and AORML languages here, but refer readers to http://www.dfki.unikl.de/ruleml/#Participants for the complete picture:

- RFML (Relational-Functional Markup Language) is a (Web-)output format for relational-functional knowledge bases and computations implemented as part of the

Relfun system. The (Web-)input translation of RFML markup into Relfun's Prolog-like syntax is implemented via an XSLT stylesheet.

- URML was initially a project to Webize the ART and ARTScript Rule Language (11). URML is pushing the effort further to integrate Object Oriented Rule-based programming with XML and provide a basis for the implementation of Web objects and their manipulation in rules.
- AORML is a project to define a markup language for agent-oriented business rules in the context of Agent Object Relationship (AOR) models.

Participants in the RuleML Initiative have expressed an urgent need for a standard rule markup language, with translators in and out along with further tools. This need provided the impetus for the RuleML effort.

This paper lays out the design rationale of the Rule Markup Language (RuleML), the Initiative's evolving markup language for the Semantic Web. To accommodate the various (Web) rule-user communities from Knowledge-Based Systems to Intelligent Agents to E-Commerce, a modular hierarchy of sublanguages will be discussed. Rule extensions will concern first-class URIs, Web-suited negations, labelings, certainties/priorities, and packages. The Initiative also examines where current description methods and implementation techniques (e.g., XML DTDs vs. Schemas and C vs. Java-based rule engines) are sufficient for such rule markup and where they would need revisions and extensions.

This paper further attempts to contribute to some open issues of Notation 3 (N3) and DAML-Rules in relation to RuleML. Finally, by studying issues of combining rules and taxonomies via sorted logics, description logics, or frame systems, the paper also touches on the US-European proposal DAML+OIL.

2. The RuleML Initiative as a Web Ontology Effort

The RuleML Initiative started in August 2000 during the Pacific Rim International Conference on Artificial Intelligence (PRICAI 2000). It has brought together expert teams from several countries, including leaders in Knowledge Representation and Markup Languages, from both academia and industry. The RuleML Initiative is developing an open, vendor neutral XML/RDF-based rule language. This will allow for the exchange of rules between various systems including distributed software components on the Web, heterogeneous client-server systems found within large corporations, etc. The RuleML language offers XML syntax for rules Knowledge Representation, interoperable among major commercial and non-commercial rules systems.

Among our industrial participants are rules engine vendors, Web technology vendors, XML/RDF tools vendors and also technology users such as financial corporations, telecom companies and some of the major Web portals and ASPs. The RuleML Initiative is collaborating with numerous related efforts such as the complementary Java Rules Engine API specification, the W3C RDF working group, the DAML group, W3C P3P Activity, PMML, and many others. This collaboration will enable RuleML to share mechanisms and provide a rules language to existing and emerging industry standards such as the Semantic Web and RDF, P3P, CC/PP and EDI (Electronic Data Interchange). The scenario in Appendix 1 exemplifies some inferential and metadata uses of RuleML for the Semantic Web.

Since RuleML participants organized a Birds Of a Feather (BOF) session at W3C's Technical Plenary and WG Meeting Event in February/March 2001, the Initiative has been discussing with W3C about possibilities of a working group devoted to Web rules (axioms) or to a combination of Web-ontology efforts as expressed by the 'equation' **ontology** = **taxonomy** + **axioms**. This would create the chance of a uniform ontology language with a description-logic taxonomy and Horn-logic-like rules.

In particular, large-scale RuleML rulebase exchange will require a taxonomy of the relations defined in the rulebase, where a relation with its arguments becomes a class with its slots. Participants in a rulebase exchange could then align each other's relation hierarchies to detect incompatibilities prior to merging and firing their rule definitions.

Conversely, large-scale DAML+OIL taxonomies will require a rule system to derive/use certain implicit information that is not captured by the taxonomy alone. The required rules could be marked up according to the suitable RuleML expressive class. DAML+OIL taxonomies and RuleML axioms should be expressed in compatible ways, ideally in one unified language. To achieve this, the current RuleML 0.8 and DAML+OIL markups could be co-developed in the Web Ontology Group towards a common version 1.0.

The fact that combined ontology systems quickly become undecidable is not a big issue since the higher RuleML expressive classes, e.g. Horn logic, are already undecidable. A big issue of the collaboration between DAML+OIL and RuleML, however, is the development of an *interleaved* layered system whose decidable taxonomy expressive classes interact well with the decidable or undecidable axiom expressive classes.

Initially, the possible combinations of taxonomies and axioms should be systematically compared w.r.t. criteria such as naturalness vs. formality, expressiveness vs. efficiency, DL terms as types for Horn variables vs. DL terms as Horn premises (or even conclusions), etc. On the taxonomy side, this comparison should span the range from order-sorted logics (which can be regarded as a degenerate description logic without slots, i.e. only the class lattice) to expressive decidable description logics such as *ALLNR*. On the axioms side, we should study the range from versions of Datalog, to Horn logic, to full first-order logic, and conservative extensions (e.g., restricted higher-order syntax). The question then is which of these respective subclasses go together well w.r.t. our criteria.

For example, it is well-known that order-sorted logics go together well with Horn logic and even with full first-order logic, as, e.g., shown by solutions to Schubert's Steamroller Problem such as (3): the combination reduces the search space. On the other hand, as shown by (9), only versions of Datalog seem to go together well with expressive description logics such as *ALLNR*: the combination enlarges the search space. If we allow free variations of both the taxonomy and axioms expressive classes, there are also many possible combinations in between. However, if a user community can state their requirements w.r.t. expressiveness of the taxonomy, the axioms, or both, it will be easier to fix the remaining degrees of freedom.

When building real Web ontologies it seems wise to start with less expressive classes on both the taxonomy and axioms sides, since a builder community cannot anticipate the requirements of future user communities. The ontological content should be packaged in an as lightweight ontology language as possible to make it available to a maximum number of users. The RuleML Initiative tried to prepare such a methodology through the bottom-up construction of a system of sublanguages from RDF-like triples to labeled Horn logic with equations plus URI individuals and relations. This could be complemented by a bottom-up taxonomy-language (re-)construction, and brought together through joint work on ontology layering.

3. The Modular Syntax and Semantics of RuleML

The modular RuleML design is described in this section. RuleML encompasses a hierarchy of rules, from reaction rules (event-condition-action rules), via integrity-constraint rules (consistency-maintenance rules) and derivation rules (implicational-inference rules), to facts (premiseless derivation rules). Till now, we have been mostly working on derivation rules and facts (cf. appendix 2).

The RuleML hierarchy of rules constitutes a partial order rooted in reaction rules. Its second main layer consists of, next to each other, integrity-constraint rules and derivation rules. The third layer just specializes derivation rules to facts. Thus, the global RuleML picture looks as shown in Figure 1.

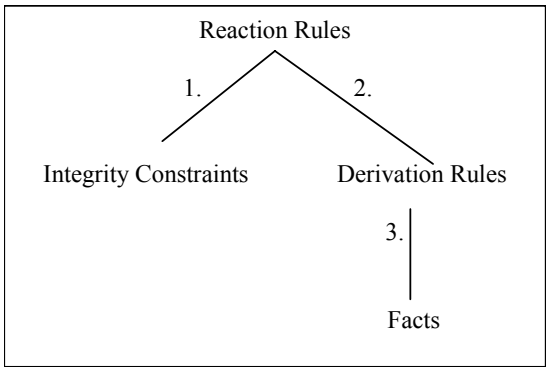


Figure 1: The RuleML hierarchy top-level.

Let us discuss the hierarchy's numbered specialization links in turn. (For a more fine-grained discussion of derivation rules, facts, and their further specialization to RDF triples see Figure 2.)

- Integrity constraints are considered as "denials" or special reaction rules whose only possible kind of action is to signal inconsistency when certain conditions are fulfilled.
- Derivation rules are considered as special reaction rules whose action happens to only add or 'assert' a conclusion when certain conditions (premises) are fulfilled. This asserting of conclusions can be regarded as a purely declarative step, as used for model generation and fixpoint semantics. Such rules can thus also be applied backward for proving a conclusion from premises.
- Facts are considered as special derivation rules that happen to have an empty (hence, 'true') conjunction of premises.

We can now make more precise our views regarding the application direction for the four rule categories:

- General reaction rules can only be applied in the forward direction in a natural fashion, observing/checking events/conditions and performing an action if and when all events/conditions have been perceived/fulfilled.
- Integrity constraints are usually also forward-oriented, i.e. triggered by updates, mainly for efficiency reasons.

- Derivation rules, on the other hand, can be applied in the forward direction as well as in a backward direction, the latter reducing the proof of a goal (conclusion) to proofs of all its subgoals (premises). Since in different situations different application directions of derivation rules may be optimal (forward, backward, or mixed), RuleML does not prescribe any one of these.
- For facts or 'unit clauses' it makes little sense to talk of an application direction.

While reaction rules, as the all-encompassing rule category, could implement all other ones, in RuleML we are introducing tailored special-purpose syntaxes for each of these categories. The following markup syntax only serves for our preliminary distinction of the four categories (for instance, we plan to permit **and**/**or** nestings besides flat conjunctions as premises):

- Reaction rules: **<rule> <_body> <and>** *prem1 ... premN* **</and> </_body> <_head>** *action* **</_head> </rule>**
- Integrity constraints: **<ic> <_body> <and>** *prem1 ... premN* **</and> </_body> </ic>** implemented by **<rule> <_body> <and>** *prem1 ... premN* **</and> </_body> <_head> <signal>** *inconsistency* **</signal> </_head> </rule>**
- Derivation rules: **<imp> <_head>** *conc* **</_head> <_body> <and>** *prem1 ... premN* **</and> </_body> </imp>** implemented by **<rule> <_body> <and>** *prem1 ... premN* **</and> </_body> <_head> <assert>** *conc* **</assert> </_head> </rule>**
- Facts: **<fact> <_head>** *conc* **</_head> </fact>** implemented by **<imp> <_head>** *conc* **</_head> <_body> <and> </and> </_body> </imp>**

Let us now elaborate on RuleML's derivation rules. Because of the infinity of possible rule-markup **syntaxes** and the rich previous work on **semantics** of rule-system classes, RuleML has attempted the following **separation of concerns**:

- **The sublanguage hierarchy**. Figure 2 shows the 12 sublanguages that together constitute the modularized basic RuleML definition. All sublanguages except the 'UR' (URL/URI) group correspond to well-known rule systems, where each sublanguage has a corresponding semantic (model- and proof-theoretic) characterization. Current work concerns a more precise URL/URI/URN semantics, as discussed in section The RuleML 0.8 DTDs. Sections Negation Handling in RuleML and Priorities/Evidences in RuleML prepare modular extensions of this basis for negations and priorities, respectively.
- **The concrete markup**. In recent months, the RuleML 0.7 DTDs have been developed into the RuleML 0.8 DTDs without affecting the above semantics. The new markup uses XML in RDF's 'explicit role-markup' style, relativizing XML's positionality to places where RDF's Seq containers or DAML+OIL lists would be needed. RuleML 0.8 is still being developed in DTDs, but will also be delivered (via translators) in XML Schemas. In the next section it will be illustrated with an earlier RuleML example, upgraded from 0.7 to 0.8.

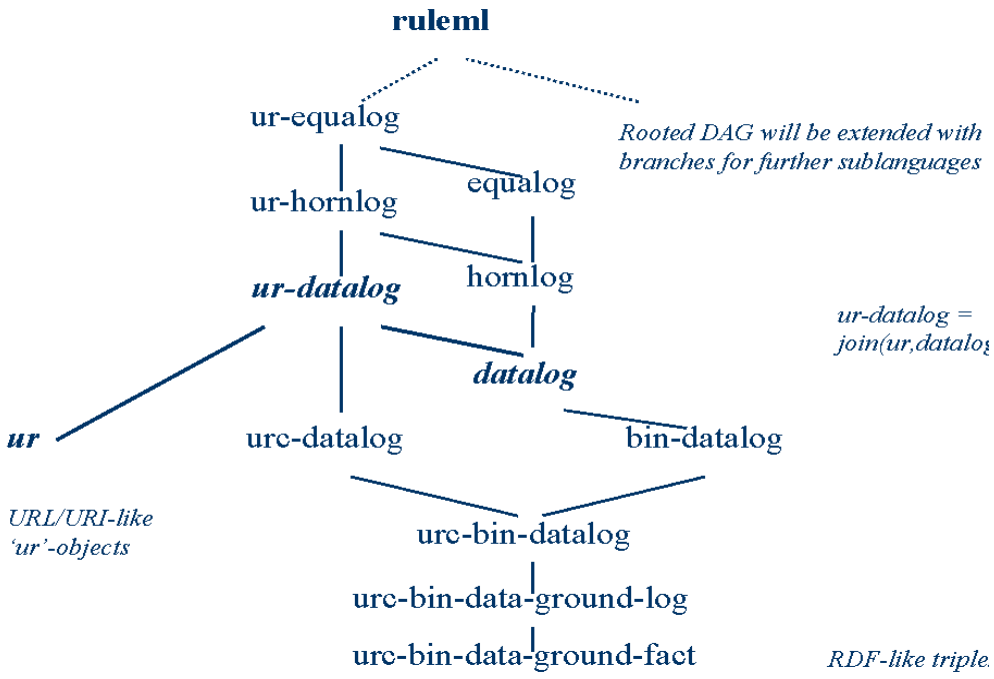


Figure 2: The RuleML hierarchy with 12 derivation-rule sublanguages.

4. The RuleML 0.8 DTDs

The upper layer of the RuleML hierarchy of rules is discussed in section The Modular Syntax and Semantics of RuleML. In that terminology, the system of RuleML DTDs presented here only covers derivation rules, not reaction rules.

This is because we think it is important to start with a subset of simple rules, test and refine our principal strategy using these, and then work 'up' to the more general categories of rules in the hierarchy. For this we choose Datalog, a language corresponding to relational databases (ground facts without complex domains or 'constructors') augmented by views (possibly recursive rules), and work a few steps upwards to further declarative rules from (equational) Horn logic. We also work upwards from a URL/URI language corresponding to simple objects. The join of both of these branches then permits inferences over RDF-like 'resources' and can be re-specialized to RDF triples.

Regarding the concrete markup syntax, we have been experimenting with several DTDs prior to the current, still preliminary, version. The rationale for our current tags is as follows.

- Rather than leaving conjunction implicit, an explicit tag pair **<and>** ... **</and>** with a sequence of N conjuncts is used (this would preferably be a set of conjuncts), preparing the unavoidable explicit markup of other boolean connectives (mainly **<or>** ... **</or>**) and their nesting.
- As a result of previous discussions, RuleML now uses an XML-RDF-unified data model with "Order-Labeled (OrdLab) Trees" (exemplified in appendix 3) as its notational base (2).
- In particular, we conventionally mark up RDF-like predicates, here called 'roles', by "_"-prefixed tags in XML (if all class-like 'type' tags would start with an upper-case

letter, then 'role' tags could also be distinguished, Java-like, by having them start with a lower-case letter, as in The FRODO rdf2java Tool).

- Using an **atom** (for a single premise) or an **and** (for a conjunction of premises) in the role of the body and an **atom**ic conclusion in the role of the head, rules aggregate two commutative roles; in particular, our Horn-like implication rules equivalently become **<imp>** **<_body>** **<and>** *prem1* ... *premN* **</and>** **</_body>** **<_head>** *conc* **</_head>** **</imp>** or become **<imp>** **<_head>** *conc* **</_head>** **<_body>** **<and>** *prem1* ... *premN* **</and>** **</_body>** **</imp>** (thus unifying KIF's "implication" and "reverse implication" syntaxes).
- The main advantage of roles is that of feature-term or **object-centered modeling**: If some extra information is to be added to an element such as a priority factor to the **imp** element, then it is easy to attach, RDF-like, a new **_priority** role with a **float**-type value; on the other hand the insertion, XML-like, of the **float**-type value directly into the child sequence would (be harder to read and) cause all subsequent children to assume a new position in the element (a problem for processing via XSLT etc.).
- In the new data model an element can have "mixed content" in the new sense of having both 'role' and 'type' children (see the **atom** examples below whose content consists of one _opr-role child and _1, _2, ... var-type children): while the 'type' children form an ordered sequence as in XML, without need for RDF's Sequence container, (1) the 'role' children are commutative as in RDF (treating an ordered sequence as a unit, as if it was reified into a Sequence container).

Appendix 2 contains a preliminary DTD for a Datalog subset of RuleML 0.8. Appendix 3 shows a simple example rule base that conforms to that DTD.

As indicated in Figure 2, besides the sublanguages towering above the Datalog DTD, there is another major RuleML branch consisting of the sublanguages on top of 'UR'-object (URL/URI) DTD. In RuleML we try to build on existing W3C work whenever possible. Hence, Uniform Resource Identifiers (URIs) are used to locate, describe and access resources and services such as classes, objects, software agents, Web components, Web services, etc. The representation of objects as URIs in RuleML will also facilitate the integration with related work on ontologies. Web objects and services use a URL/URI as their unique object identifier (cf. SHOE, RDF, URML) and the point of access to the Web (and in some cases standalone or intranet) resource or software agent. URLs/URIs can be embedded in facts, rule conditions and rule actions.

The RuleML language thus offers support for URIs in its system of DTDs starting from the 'UR' sublanguage. For example, in UR-Datalog, names can be assigned to individuals and relations using content markup and/or an URI attribute. The content markup need not be unique while the URI attribute is unique. The modular design of RuleML will allow us to extend URIs to a number of other addressing schemes.

As a simple Datalog example consider the facts in appendix 3, which use content markup to name, perhaps not uniquely, an individual book. Alternatively, in UR-Datalog the first of these facts, say, can use a URI under an **href** attribute of the empty **ind** element as follows:

```
<fact>
  <_head>
    <atom>
      <_opr><rel>sell</rel></_opr>
      <ind>John</ind>
      <ind>Mary</ind>
      <ind href="http://www.ibiblio.org/xml/books/bible2"/>
    </atom>
  </_head>
</fact>
```

Moreover, the second of these facts, say, can now combine the original content markup with the URI attribute as follows:

```
<fact>
  <_head>
    <atom>
      <_opr><rel>keep</rel></_opr>
      <ind>Mary</ind>
      <ind href="http://www.ibiblio.org/xml/books/bible2">XMLBible</ind>
    </atom>
  </_head>
</fact>
```

It should be noted that, content markup not being unique, a given URI can be combined with different content markups in different elements. Thus, the second fact, say, could also use the same URI with this time an extended PCDATA **New XMLBible**. Conversely, of course, two different URIs can be combined with the same content markup.

5. Negation Handling in RuleML

In natural language, and in practical knowledge representation systems, such as the IBM business rule system CommonRules (6) that is based on the formalism of *extended logic programs*, there are two kinds of negation: a *weak* negation expressing *non-truth* (in the sense of "I don't like snow"), and a *strong* negation expressing explicit *falsity* (in the sense of "I dislike snow"). In RuleML, the weak negation connective is denoted by '*not*' and the strong negation connective by '*neg*'. In the case of a complete predicate, such as being an odd number, both negations collapse: 'not odd(x)' is equivalent to 'neg odd(x)', or in other words, the non-truth of the atom 'odd(x)' amounts to its falsity. In the case of an incomplete predicate, such as 'like', we only have that the strong negation implies the weak negation: 'neg like(I,snow)' implies 'not like(I,snow)', but not conversely. Also, while the double negation form 'neg not' collapses (according to partial logic, see [Wag98]), the double negation form 'not neg' does not collapse: not disliking snow does not amount to liking snow.

Using two kinds of negation in derivation rules has been proposed independently in (7) and (12). Rules with weak negation, or with other non-persistent connectives, lead to nonmonotonic inference. It is well-known that the semantics of nonmonotonic knowledge systems is not based on all models of a knowledge base but solely on the set of all *intended* models. E.g., for relational databases, which can be viewed as the most fundamental case of a knowledge system, the intended models are the minimal ones. The model-theoretic semantics of nonmonotonic rules is based on the concept of *stable (generated)* models in

classical and partial logic (see 7, 8 and 9). Notice that classical logic can be viewed as the degenerate case of partial logic when all predicates are total.

Under the preferential semantics of stable (generated) models, the weak negation 'not' corresponds to *negation-as-failure* in Prolog and to the *EXCEPT* operator in SQL in the following way: a query expression "give me all objects x such that 'p(x) and not q(x)'" corresponds to the SQL expression 'P EXCEPT Q' where P and Q denote the tables that represent the extensions of the predicates p and q. Since SQL tables were not intended to be able to represent incomplete predicates, SQL does not contain a strong negation operator.

Because in many computational domains predicates are assumed to be complete (according to the Closed-World Assumption), 'not' is used more frequently than 'neg'. An example of a rule that defines a derived attribute of a certain class in a UML class model is the following: A car is available for rental if it is physically present, is not assigned to any rental order, is not scheduled for service, and does not require service. This rule defines the derived Boolean attribute 'isAvailable' of the class 'RentalCar' by means of the stored Boolean attributes 'isPresent', 'requiresService', 'isScheduledForService', and an association 'isAssignedTo' between cars and rental orders, here called '`isAssignedToRentalOrder`'. The association is shown more explicitly in the UML class model of Figure 3.

```
<imp>
  <_head>
    <atom>
      <_opr><rel>isAvailable</rel></_opr>
      <var>Car</var>
    </atom>
  </_head>
  <_body>
     <and>
      <atom>
        <_opr><rel>isPresent</rel></_opr>
        <var>Car</var>
      </atom>
      <not>
        <atom>
          <_opr><rel>isAssignedToRentalOrder</rel></_opr>
          <var>Car</var>
        </atom>
      </not>
      <not>
        <atom>
          <_opr><rel>isScheduledForService</rel></_opr>
          <var>Car</var>
        </atom>
      </not>
      <not>
        <atom>
          <_opr><rel>requiresService</rel></_opr>
          <var>Car</var>
        </atom>
      </not>
    </and>
  </_body>
</imp>
```

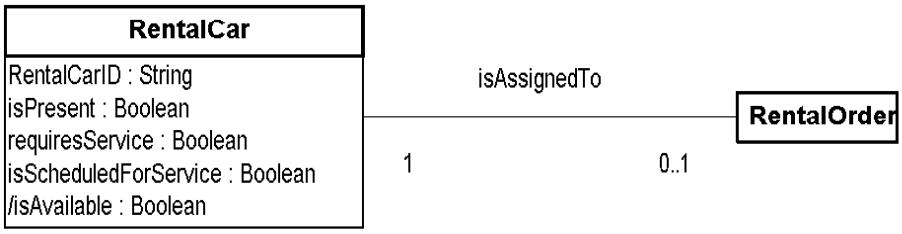


Figure 3: A UML model of the class *RentalCar* with the derived Boolean attribute */isAvailable*.

The strong negation is an "open world" negation, since in an open world such as the Web, the non-truth (or failure) of a statement does not imply its falsity. By combining weak and strong negation, one can express default rules (in the sense of Reiter's default logic) in a natural way. An example of this is the rule "a document that is not classified as being official has normally to be treated as an unofficial document". Such a rule could, for instance, supplement an ontology about enterprise documents and help answering queries about unofficial documents. Let us assume that EEEBizz classifies documents by means of a 'full'/'partial'/'open'-valued *Approval* property, while EEEComm classifies documents with the help of a 'yes'/'no'-valued *Released* property. Then, we may want to use a rule that allows to conclude a strongly negated atom on the basis of either of two weakly negated atoms (the **`or`** in the **`_body`** could be eliminated via separate rules for the disjuncts):

```
<imp>
  <_head>
    <neg>
      <atom>
        <_opr><rel>isOfficialDocument</rel></_opr>
        <var>DocumentName</var>
      </atom>
    </neg>
  </_head>
  <_body><or>
      <not>
        <atom>
          <_opr>
               <rel href="http://www.eeebizz.com/rdf sch#Approval"/>
          </_opr>
          <var>DocumentName</var>
          <ind>full</ind>
        </atom>
      </not>
      <not>
        <atom>
          <_opr>
               <rel href="http://www.eeecomm.net/rdf-voc#Released"/>
          </_opr>
          <var>DocumentName</var>
          <ind>yes</ind>
        </atom>
      </not>
    </or></_body>
</imp>
```

Notice that this rule allows to conclude that a document is unofficial unless the contrary is known. Therefore, it would provide the conclusion that a certain document is unofficial even if it suggests to be official (at its own URI) but is not classified properly (at the metadata's URI). This rule cannot be applied if there is an explicit 'full Approval'

classification and an explicit 'yes, Released' classification of the document (according to the respective definitions of EEEBizz and EEEComm).

For example, suppose the metadata consist only of the following 'full Approval' fact, an RDF triple (according to the URC-bin-data-ground-fact DTD of RuleML, cf. Figure 2) about a joint-mission document:

```
<fact>
  <_head>
    <atom>
      <_opr><rel href="http://www.eeebizz.com/rdf-sch#Approval"/></_opr>
      <ind href="http://www.eeebico.org/docs/joint-mission.html"/>
      <ind>full</ind>
    </atom>
  </_head>
</fact>
```

The first disjunct is false since its 'Approval' atom unifies with the fact (via the binding of **<var>DocumentName</var>** to **<ind href="http://www.eeebico.org/docs/joint-mission.html"/>**); but the second disjunct is true for lack of a corresponding 'yes, Released' fact; so the default rule classifies the document as unofficial.

6. Priorities/Evidences in RuleML

The following is an example using an auto insurance scenario. This example involves two conflicting rules, shown below. The first rule, which applies to drivers under 25 years of age, states that after the accident, the premium will increase by 40%. On the other hand, in the second rule, because the customer is on the family plan, his or her premium will not increase after the first accident. This example is treated in more detail in appendix 1.

The first rule, applicable to drivers under 25:

```
<imp>
  <_rlab><ind>beginners</ind></_rlab>
  <_spriority><ind>0.75</ind></_spriority>
  <_head>
        .......
  </_head>
  <_body>
    <and>
        ...
      <atom>
        <_opr><rel>customerUnder25</rel></_opr>
        <var>customer</var>
      </atom>
    </and>
  </_body>
</imp>
```

The second rule, applicable to drivers with a family plan:

```
<imp>
  <_rlab><ind>family</ind>  </_rlab>
  <_spriority><ind>0.9</ind> </_spriority>
  <_head>
        .......
  </_head>
  <_body>
    <and>
      <atom>
        <_opr><rel>FamilyAutoPlan</rel></_opr>
        <var>customer</var>
        <var>familyauto</var>
      </atom>
    </and>
  </_body>
</imp>
```

Both research prototypes and commercial rules engines offer a facility for controlling rule execution and conflict resolution. In RuleML, one can define either quantitative priorities declaring a numerical *Priority* property for rules or qualitative priorities using *Overrides* facts over rule labels.

A quantitative priority is a numerical value indicating the salience (or the evidence) of a rule. We consider supporting both static and dynamic salience. A static priority value can be represented by a constant or a variable. A dynamic salience is represented using a variable or a function or relation call: the numerical value is calculated at runtime from the current binding environment.

Qualitative priorities are represented using facts comparing rule labels. This approach is influenced by the rules conflict handling in BRML, based on partially-ordered prioritization information (6). Qualitative priorities using the *Overrides* fact can be generated from numerical saliences. For example, in the auto insurance example above, since rule labeled 'family' (salience 0.9) is higher priority than rule labeled 'beginners' (salience 0.75), we can generate the following qualitative priority fact: ***Overrides(family, beginners)***, which means that rule family will always win if it enters in a conflict with rule beginners.

7. Agents and RuleML

Biological and artificial systems that interact with their (natural or virtual) environment on the basis of their *mental state*, and exhibit some degree of autonomy, are called "agents". The most basic mental components of an agent are its *perceptions* of events (in the form of incoming messages) and its *beliefs* (or knowledge). Further important components are

- *memory* about past events and actions,
- *commitments* towards other agents to perform certain actions,
- *claims* against other agents,
- *goals* in the form of state conditions to be achieved by means of planning and plan execution, and
- *intentions* in the form of action plans that have been chosen to be executed.

A sophisticated software agent may be specified by

- an RDFS-based taxononmy for defining the schema of its mental state,
- a set of RDF facts for specifying its factual (extensional) knowledge,
- a set of RuleML integrity constraints for excluding non-admissible mental states,
- a set of RuleML derivation rules for specifying its terminological and heuristic (intensional) knowledge, and
- a set of RuleML reaction rules for specifying its behavior in response to communication and environment events.

Thus, it will be possible to completely specify a software agent using RDF/RDFS and RuleML. Executing such an agent specification requires a combination of a knowledge subsystem (including an inference and an update operation), a perception (or incoming message handling) subsystem and an action (or outgoing message handling) subsystem.

Michael Sintek has recently implemented a much simpler first example of a RuleML querying agent. This is a servlet (running in Tomcat) that receives RuleML rulebases in an

RDF-based RuleML syntax (since it uses The FRODO rdf2java Tool) together with some queries, evaluates them with XSB Prolog (in auto-tabling mode, which should be equivalent to bottom-up evaluation), and returns the result as an HTML page containing the bindings as facts of instantiated queries. A future version must, of course, return a RuleML file. It can be tried at this URL: Click on 'example' and paste the RDF RuleML popping up into the input window (note that pasting XML/RDF cannot be directly done in IE, only in Netscape; use "view source" in IE). Alternatively, you can use the Prolog parser and RDF translator to generate the RDF RuleML. Since it cannot be guaranteed that the above URLs will always work (server reboots etc.), this picture shows the agent in action.

8. RuleML Implementations via XSLT and Rule Engines

XSLT can itself be regarded as a rule-based programming language operating on XML elements. These elements can also be other rules expressed in XML. The RuleML Initiative has been implementing the translation between various rules systems using XSLT stylesheets. The first XSLT stylesheet from RuleML to another system demonstrated the translation of RuleML 0.7 to RDF; it can be seen as a preparation of our transition towards the current more RDF-oriented RuleML 0.8.

One of the most popular (reaction) rule engines currently available free for non-commercial use is JESS (Java Expert System Shell). Jess is implemented in the Java language. It was originally inspired by the CLIPS expert system shell, but has grown into a complete, distinct rule-based tool of its own. CLIPS is a development environment for rule-based and object oriented expert systems. CLIPS is being used by government agencies, research laboratories and universities as well as a number of companies around the world.

Following the release of RuleML 0.8, we will provide an XSLT style sheet that produces Jess code. A style sheet already exists for RuleML 0.7, compatible with Jess 60a5.

The example below shows a RuleML 0.8 rule originally authored using RuleML 0.7 and translated into Jess using an XSLT stylesheet. This kind of process can be automated easily in a Web-based platform using existing XML and XSLT tools and APIs. The same rule is translated into Prolog. This demonstrates the flexibility and the power of the rules exchange mechanism offered in RuleML.

The Rule written in RuleML:

```
<rulebase label="myRules">
  <imp>
    <_head><atom>
        <rel>likes</rel>
        <ind>John</ind>
        <var>x</var>
    </atom></_head>
    <_body><atom>
        <rel>likes</rel>
        <var>x</var>
        <ind>wine</ind>
      </atom></_body>
  </imp>
</rulebase>
```

The transformation to Jess gives the following Jess (and CLIPS) rule:

```
        (defrule myRules-1
                 "This rule has been generated from RuleML"
                 (likes ?x wine)
             =>
                 (likes John ?x))
```

and the transformation to Prolog returns the rule:

```
likes(John, X) :- likes(X, Wine).
```

With GEDCOM, Mike Dean created the first operational RuleML (0.7) rulebase, where rules on family relationships (child, spouse, etc.) are run via XSLT translators to the XSB, JESS, and n3/cwm engines. Besides indirectly, via translators, RuleML implementations should also be done directly, via rule engines.

With Mandarax RuleML, Jens Dietrich has implemented the first complete input-processing-output environment for RuleML (0.8). For a RuleML 0.8 engine we also cooperate with the CommonRules, Euler, and TRIPLE projects and hope to also join forces with W3C's N3 and NILE efforts, and with further interested companies.

9. Conclusions

Looking at the bigger picture of "ontologies", we will now discuss three related requirements for future RuleML versions.

1) Following our earlier 'taxonomy-plus-axioms' notion of "ontology", RuleML, together with DAML-Rules and Euler, can be seen as the "axioms part" working on the "taxonomy part" developed by some other effort such as DAML+OIL. Derivation rules are normally used in the context of an information model, such as a UML class model, an RDFS-based taxonomy (as used in DAML+OIL ontologies), or a predicate logic signature. The underlying information model defines a language for expressing logical statements that can play the role of an assertion, of a query, or of a condition. It should be possible to include a RuleML rulebase (or a reference to a RuleML document) within an XML-based version of an information model (such as a UMI document or a RDFS-based taxononmy). Vice versa, it should be possible to include (a reference to) such an information model within the XML-based RuleML rulebase. Ideally, a 'taxonomy-plus-axioms' ontology should include both parts on the same level, as pioneered by SHOE and N3.

Implied requirement for RuleML: A RuleML rulebase can either be embedded in an information model, or its top-level element ("rulebase") can have an attribute that specifies its context by refering to a respective XML document.

2) We could link to UML classes via RuleML variables: **<var>** could have an attribute giving the class constraining it. Also, a DAML+OIL taxonomy could be linked in such a "sorted logic" manner. We could additionally allow to plug in some other atom-defining formalism as an option. The "atoms" used in the premise and conclusion of a derivation rule in the context of a UML class model would then be expressed in OCL. The "atoms" used in the premise and conclusion of a derivation rule in the context of a DAML+OIL taxononmy would then be expressed in DAML+OIL RDF.

Implied requirement for RuleML: A separation of concerns: the proper rule language is more concerned with sentential connectives and rule keywords, than with the language of "atoms". The language of "atoms" can be called the content language of a RuleML rulebase. It consists of two layered sublanguages: 1) the predefined constructs of the chosen metamodel (like UML or RDFS), and 2) the terms defined by the chosen model/taxononmy.

3) Derivation rules operate on facts that are typically represented in a database, or in an XML or RDF document.

Implied requirement for RuleML: It should be possible to include a RuleML rulebase (or a reference to a RuleML document) within an XML or RDF document. Technically, it is easy to mix RuleML and other XML namespaces (like for, say, MathML), incl. RDF(S) namespaces. For this we assume a **ruleml:** namespace prefix.

Acknowledgements

Benjamin Grosof, MIT Sloan School of Management, continues to make significant contributions to RuleML, in particular to its DTDs. Michael Sintek, DFKI, provided a lot of help and devised the first RuleML querying agent. We acknowledge the remarks of the SWWS reviewers, which have helped us to improve this paper. Thanks go also to the EU for funding Harold Boley's research on collaborative Web technologies within the Clockwork project. The RuleML Initiative has been supported in part by Nisus, Inc. and DFKI who funded Said Tabet's and Harold Boley's active participation and the inception of the RuleML effort.

References

[1] Harold Boley. Relationships Between Logic Programming and RDF. *Proc. 1st Pacific Rim International Workshop on Intelligent Information Agents (PRIIA 2000)*, University of Melbourne, Australia, 2000; LNAI volume to be published.

[2] Harold Boley. A Web Data Model Unifying XML and RDF. *Working Draft*, DFKI Kaiserslautern, July 2001.

[3] A.G. Cohn. On the Solution of Schubert's Steamroller in Many Sorted Logic. *IJCAI-85*, pages 1169--1174, 1985.

[4] Stefan Decker and Dan Brickley and Janne Saarela and Juergen Angele. A Query and Inference Service for RDF. *QL'98 - The Query Languages Workshop*, World Wide Web Consortium, 1998.

[5] Stefan Decker and Michael Sintek. Triple. *RDF Interest Group: Face to face meeting*, World Wide Web Consortium, 2001.

[6] B.N. Grosof. Prioritized Conflict Handling for Logic Programs. *Proc. of the Int. Symposium on Logic Programming (ILPS-97)*, edited by Jan Maluszynski, MIT Press, Cambridge, MA, USA, 1997.

[7] M. Gelfond and V. Lifschitz. Logic Programs with Classical Negation. *Proc. of Int. Conf. on Logic Programming (ICLP'90)*, MIT Press, 1990.

[8] H. Herre and G. Wagner. Stable Models are Generated by a Stable Chain. *J. of Logic Programming* 30:2 (1997), pages 165-177.

[9] A. Levy and M.-C. Rousset. Combining Horn Rules and Description Logics in CARIN. *Artificial Intelligence Journal* 104, September 1998.

[10] K. Sagonas and T. Swift. An Abstract Machine for Tabled Execution of Fixed-Order Stratified Logic Programs. *ACM Transactions on Programming Languages and Systems* 20:3, pages 586-634, May 1998.

[11] S. Tabet, P. Bhogaraju and D. Ash. Using XML as a Language Interface for AI Applications. *Proceedings of the Symposium on the Application of Artificial Intelligence in Industry*, pages 133-142, Sixth Pacific Rim International Conference on Artificial Intelligence, Melbourne, Australia, August, 2000.

[12] G. Wagner. A Database Needs Two Kinds of Negation. In B. Thalheim and H.-D. Gerhardt (eds.), *Proc. of the 3rd Symp. on Mathematical Fundamentals of Database and Knowledge Base Systems (MFDBS'91)*, LNCS 495, pages 357-371, Springer-Verlag, 1991.

[13] G. Wagner. *Foundations of Knowledge Systems - with Applications to Databases and Agents*. Kluwer Acadamic Publishers, 1998.

Appendix 1: A Semantic Web Scenario in the Insurance Industry

In this appendix, we provide a Semantic Web scenario applying RuleML in a common pragmatic situation. After a car accident, one of the questions people are facing is: *how much will my premiums increase and how does this accident affect my insurance policy?* Not all insurance companies follow the same rules or apply the same formula. In the USA this results in premium increases that can vary from hundreds of dollars to over a thousand. Many companies follow the Insurance Services Office (ISO) standard of increasing your premium by 40 percent of their "base rate" after your first at-fault accident. A base rate is the average amount of all claims paid, plus the insurance company's processing fee. For example, if your company's base rate is $600, your premium after the accident will go up by $240.

In our scenario, Olivia is a teenager who unfortunately has just had her first car accident. She is insured on her mother's premium family insurance plan. This situation involves two conflicting rules, as formalized in RuleML below. The first rule, which applies to drivers under 25 years of age, states that after the accident, Olivia's premium will increase by 40%. On the other hand, the second rule, applying to drivers on a family plan, states that her premium will not increase at all after her first accident.

The first rule, applicable to drivers under 25:

```
<imp>
  <_rlab><ind>beginners</ind></_rlab>
  <_spriority><ind>0.75</ind></_spriority>
  <_head>
    <atom>
      <_opr><rel>calculatePremium</rel></_opr>
      <var>customer</var>
      <ind>40</ind>
    </atom>
  </_head>
  <_body>
    <and>
      <atom>
        <_opr><rel>InsurancePolicy</rel></_opr>
        <var>customer</var>
        <var>insurance</var>
      </atom>
      <atom>
        <_opr><rel>lifeEvent</rel></_opr>
        <var>customer</var>
        <ind>accident</ind>
        <var>report</var>
      </atom>
      <atom>
        <_opr><rel>customerUnder25</rel></_opr>
        <var>customer</var>
      </atom>
    </and>
  </_body>
</imp>
```

The second rule, applicable to drivers with a family plan:

```
<imp>
  <_rlab><ind>family</ind></_rlab>
  <_spriority><ind>0.9</ind></_spriority>
  <_head>
    <atom>
      <_opr><rel>calculatePremium</rel></_opr>
      <var>customer</var>
      <ind>0</ind>
    </atom>
  </_head>
  <_body>
    <and>
      <atom>
        <_opr><rel>InsurancePolicy</rel></_opr>
        <var>customer</var>
        <var>insurance</var>
      </atom>
      <atom>
        <_opr><rel>lifeEvent</rel></_opr>
        <var>customer</var>
        <ind>accident</ind>
        <var>report</var>
      </atom>
      <atom>
        <_opr><rel>FamilyAutoPlan</rel></_opr>
        <var>customer</var>
        <var>familyauto</var>
      </atom>
    </and>
  </_body>
</imp>
```

Let us now turn to formalizing the relevant facts.

Olivia is under 25:

```
<fact>
  <_head>
    <atom>
      <_opr><rel>customerUnder25</rel></_opr>
      <ind>Olivia</ind>
    </atom>
  </_head>
</fact>
```

The following RDF-like RuleML facts permit to prove further premises of the above rules and also provide metadata descriptions for the required documents referenced and retrieved by URIs.

Olivia has an insurance policy and this document has link **`.../IMA-0835`**:

```
<fact>
  <_head>
    <atom>
      <_opr><rel>InsurancePolicy</rel></_opr>
      <ind>Olivia</ind>
      <ind href="http://www.BostonInsurance.com/policy/IMA-0835"/>
    </atom>
  </_head>
</fact>
```

Olivia is in a family auto plan and this document has link **`.../FMA-0142`**:

```
<fact>
  <_head>
    <atom>
      <_opr><rel>FamilyAutoPlan</rel></_opr>
      <ind>Olivia</ind>
      <ind href="http://www.BostonInsurance.com/plan/FMA-0142"/>
    </atom>
  </_head>
</fact>
```

Olivia's accident report is available at TrafficReport.biz:

```
<fact>
  <_head>
    <atom>
      <_opr><rel>lifeEvent</rel></_opr>
      <ind>Olivia</ind>
      <ind>accident</ind>
      <ind href="http://www.TrafficReport.biz/MA/report0712"/>
    </atom>
  </_head>
</fact>
```

The 'metafact' below is used to resolve the conflict between rule **beginners** and rule **family** (cf. section Priorities/Evidences in RuleML):

```
<fact>
  <_head>
    <atom>
      <_opr><rel>Overrides</rel></_opr>
      <ind>family</ind>
      <ind>beginners</ind>
    </atom>
  </_head>
</fact>
```

The rulebase presented in this example illustrates the use of Web-based documents in rules for matching and inferencing. In this example, we also show how priorities can be applied to rules. The **Overrides** fact above will allow rule **family** to fire as a higher priority rule and save Olivia a good amount of money: her premium will not increase.

Appendix 2: DTD for a Datalog Subset of RuleML

```
<!-- An XML DTD for a Datalog RuleML Sublanguage: Monolith Version -->
<!-- Last Modification: 2001-07-07 -->

<!-- ELEMENT Declarations -->

<!-- 'rulebase' root element uses 'imp' rules and 'fact' assertions on top-level -->
<!ELEMENT rulebase ((imp | fact)*)>

<!-- 'imp' rules are usable as general implications on the top-level -->
<!-- 'imp' element uses a conclusion role _head before a premise role _body, or -->
<!--  uses a premise role _body before a conclusion role _head -->
<!ELEMENT imp ((_head, _body) | (_body, _head))>

<!-- 'fact' assertions are usable as degenerate rules on the top-level -->
<!-- 'fact' element uses just a conclusion role _head -->
<!-- "<fact>_head</fact>" stands for "_head is implied by true" -->
<!ELEMENT fact (_head) >

<!-- _head role is usable within 'imp' rules and 'fact' assertions -->
<!-- _body role is usable within 'imp' rules -->
<!-- _head uses an atomic formula -->
<!-- _body uses an atomic formula or an 'and' -->
<!ELEMENT _head (atom)>
<!ELEMENT _body (atom | and)>

<!-- an 'and' is usable within _body's -->
<!-- 'and' uses zero or more atomic formulas -->
<!-- "<and>atom</and>" is equivalent to "atom"-->
<!-- "<and></and>" is equivalent to "true"-->
<!ELEMENT and (atom*)>

<!-- atomic formulas are usable within _head's, _body's, and 'and's -->
<!-- atom element uses an: -->
<!-- _opr ("operator of relations") role followed by zero or more arguments, or -->
<!-- one or more argument followed by an _opr role -->
<!-- the arguments may be ind(ividual)s or var(iable)s -->
<!ELEMENT atom ((_opr, (ind | var)*) | ((ind | var)+, _opr))>

<!-- _opr is usable within atoms -->
<!-- _opr uses rel(ation) symbol -->
<!ELEMENT _opr (rel)>

<!-- there is one kind of fixed argument -->
<!-- individual constant, as in predicate logic -->
<!ELEMENT ind  (#PCDATA)>

<!-- there is one kind of variable argument -->
<!-- logical variable, as in logic programming -->
<!ELEMENT var  (#PCDATA)>

<!-- there are only fixed (first-order) relations -->
<!-- relation or predicate symbol -->

<!ELEMENT rel  (#PCDATA)>
```

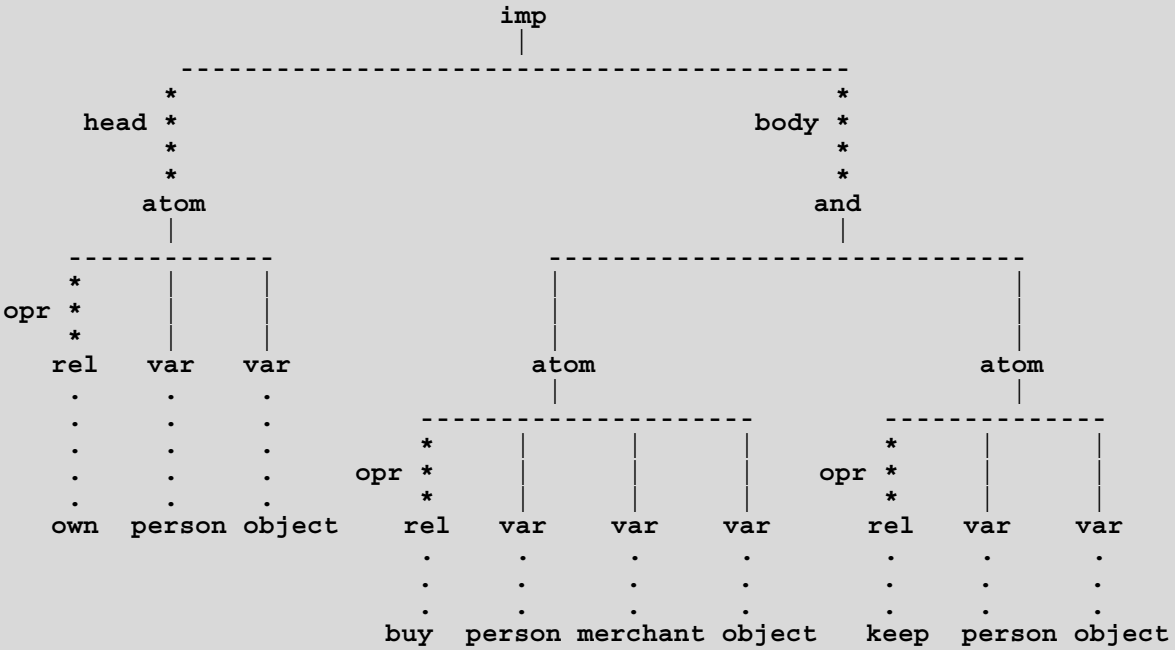
Appendix 3: Example RuleML Document: A Rulebase `own.ruleml`

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE rulebase SYSTEM "http://www.dfki.de/ruleml/dtd/0.8/ruleml-datalog-monolith.dtd">
<rulebase>

<!-- start XML comment ...
This example rulebase contains four rules. The first and second rules are implications; the third and
fourth ones are facts.

The first rule implies that a person owns an object if that person buys the object from a merchant and
the person keeps the object.

As an OrdLab Tree:
```




```
... end XML comment -->

<imp>
 <_head>
  <atom>
   <_opr><rel>own</rel></_opr>
   <var>person</var>
   <var>object</var>
  </atom>
 </_head>
 <_body>
  <!-- explicit 'and' -->
  <and>
   <atom>
    <_opr><rel>buy</rel></_opr>
    <var>person</var>
    <var>merchant</var>
    <var>object</var>
   </atom>
   <atom>
    <_opr><rel>keep</rel></_opr>
    <var>person</var>
    <var>object</var>
   </atom>
  </and>
```

```
  </_body>
</imp>


<!-- The second rule implies that a person buys an object from a merchant
if the merchant sells the object to the person. -->

<imp>
  <_head>
    <atom>
      <_opr><rel>buy</rel></_opr>
      <var>person</var>
      <var>merchant</var>
      <var>object</var>
    </atom>
  </_head>
  <_body>
    <atom>
      <_opr><rel>sell</rel></_opr>
      <var>merchant</var>
      <var>person</var>
      <var>object</var>
    </atom>
  </_body>
</imp>

<!-- The third rule is a fact that asserts that John sells XMLBible to Mary. -->

<fact>
  <_head>
    <atom>
      <_opr><rel>sell</rel></_opr>
      <ind>John</ind>
      <ind>Mary</ind>
      <ind>XMLBible</ind>
    </atom>
  </_head>
</fact>

<!-- The fourth rule is a fact that asserts that Mary keeps XMLBible.

Observe that this fact is binary - i.e., there are two arguments for the relation. RDF viewed as a logical
knowledge representation is, likewise, binary, although its arguments have type restrictions,
e.g., the first must be a resource (basically, a URI). Some of the DTD's on the RuleML website handle
URL's/URI's (UR's); see especially urc-datalog.dtd for inferencing with RDF-like facts

-->

<fact>
  <_head>
    <atom>
      <_opr><rel>keep</rel></_opr>
      <ind>Mary</ind>
      <ind>XMLBible</ind>
    </atom>
  </_head>
</fact>
</rulebase>
```