# Rule markup languages and semantic web rule languages

Paschke, Adrian; Boley, Harold

**Publisher’s version / Version de l'éditeur:**
https://doi.org/10.4018/978-1-60566-402-6.ch001
*Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, pp. 1-24, 2010-05-01*

**NRC Publications Record / Notice d'Archives des publications de CNRC:**
https://nrc-publications.canada.ca/eng/view/object/?id=487ffb96-b1cd-424c-8724-89e75653eec7
https://publications-cnrc.canada.ca/fra/voir/objet/?id=487ffb96-b1cd-424c-8724-89e75653eec7

# RULE MARKUP LANGUAGES AND

# SEMANTIC WEB RULE LANGUAGES

Adrian Paschke

Harold Boley

Free University Berlin, Germany

National Research Council Canada

## ABSTRACT

Rule markup languages will be the vehicle for using rules on the Web and in other distributed systems. They allow publishing, deploying, executing and communicating rules in a network. They may also play the role of 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 languages addressing different consumers. The main purposes of a rule markup language are to permit the publication, interchange and reuse of rules. This chapter introduces important requirements and design issues for general Web rule languages to fulfill these tasks. Characteristics of several important general standardization or standards-proposing efforts for (XML-based) rule markup languages including W3C RIF, RuleML, R2ML, SWRL as well as (human-readable) Semantic Web rule languages such as TRIPLE, N3, Jena, and Prova are discussed with respect to these identified issues.

## INTRODUCTION AND MOTIVATION

Web rule languages provide the required expressiveness enabling machine-interpretation, automated processing and translation into other such Web languages, some of which also being the execution syntaxes of rule engines. One of these languages may act as a "lingua franca" to interchange rules and integrate with other markup languages, in particular with Web languages based on XML and with Semantic Web languages (e.g. W3C's RDF Schema, OWL and its new OWL 2 version) for ontologies serialized in RDF/XML or directly in XML. Web rule languages may also be used for publication purposes on the Web and for the serialization of external data sources, e.g. of native online XML databases or RDF stores. Recently, there have been several efforts aiming at rule interchange and building a general, practical, and deployable rule markup standard for the (Semantic) Web. These encompass several important general standardization or standards-proposing efforts including RuleML (www.ruleml.org), SWRL (www.w3.org/Submission/SWRL/), SWSL (http://www.w3.org/Submission/SWSF-SWSL/), R2ML (oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=R2ML), RIF (www.w3.org/2005/rules/), and others such as XCL (http://www.altheim.com/specs/xcl/1.0/), designed as a concrete (serialization) syntax for ISO's Common Logic (CL) standard.

In this chapter, a system of general requirements and design choices for Web rule languages will be introduced and instantiations discussed in the context of the current prominent general Rule Markup Languages and Semantic Web rule languages. This chapter is intended to be of help to a wide audience. In particular, it is targeted to rule practitioners who want to serialize the declarative rules of their applications in a general rule markup language, and publish and interchange them on the Web. Rule practitioners will find here a discussion of general design criteria with examples from the current rule markup languages. These examples, together with a discussion of advantages and drawbacks, will offer guidance to readers when declaratively representing their own rule-based applications in a Web rule language. The structure of the rest of this chapter is as follows: Section 2 introduces current rule markup languages and rule interchange formats as well as Semantic Web rule languages. Section 3 comprises the main part of this chapter, discussing important design issues and characteristics of the introduced rule languages. Section 4 presents future research issues in Web rule language design. Section 5 concludes this chapter with a summary.

## WEB RULE LANGUAGES

Rule markup (serialization) languages have been developed for the Web-based interchange of, e.g., privacy policies, business rules, and - as focused here - Semantic Web rules. Rules are central to knowledge representation for the Semantic Web (Boley, 2007), hence are increasingly considered as being side by side with ontologies, e.g. in W3C's layered Semantic Web architecture (2007 version shown in Figure 1).

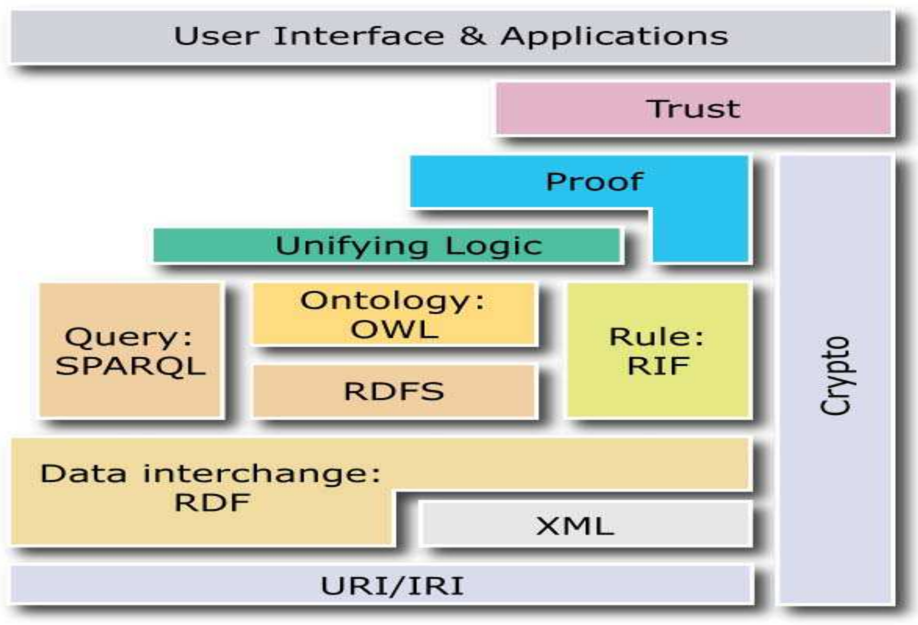

**Figure 1: Tim Berners-Lee's Semantic Web Layer Cake**

Rule interchange in an open format is important for all higher Semantic Web layers, including a Web of Trust and, generally, a Pragmatic Web (Paschke et al, 2007), and is crucial for applications in eBusiness, eGovernment, eHealth, etc. This section introduces major *rule markup languages* including RuleML, R2ML, and RIF, as well as human-readable *Semantic Web rule languages* such as TRIPLE and N3, and platform-specific rule engine languages such as Jena and Prova.

### Rule Markup Languages

We characterize rule languages as rule markup languages if they are serialized in XML, employ URIs/IRIs for constants etc., and can interface with Web ontology languages.

## *RuleML*

The Rule Markup Language (RuleML, www.ruleml.org) is a markup language developed to express a family of Web rules in XML for deduction, rewriting, and reaction, as well as further inferential, transformational, and behavioral tasks. It is defined by the Rule Markup Initiative (www.ruleml.org), an open network of individuals and groups from both industry and academia that was formed to develop a canonical Web language for rules using XML markup and transformations from and to other rule standards/systems. It develops a modular, hierarchical specification for different types of rules comprising facts, queries, derivation rules, integrity constraints (consistency-maintenance rules), production rules, and reaction rules (Reaction RuleML, http://ibis.in.tum.de/research/ReactionRuleML), as well as tools and transformations from and to other rule standards/systems.

Datalog RuleML is defined over both data constants and individual constants with an optional attribute for IRI (URI) webizing[1]. Atomic formulas have n arguments, which can be positional terms or, in Object-Oriented Datalog, slots (F-logic-like key->term pairs); OO Datalog also adds optional types and RDF-like oids/anchors, via IRIs (Boley, 2003). Inheriting all of these Datalog features, Hornlog RuleML adds positional or slotted functional expressions as terms. In Hornlog with equality, such uninterpreted (constructor-like) functions are complemented by interpreted (equation-defined) functions. This derivation rule branch is extended upward towards First Order Logic, has subbranches with Negation-As-Failure, strong-Negation, or combined languages, and is parameterized by 'pluggable' built-ins.

## *SWRL*

The Semantic Web Rule Language (SWRL, www.w3.org/Submission/SWRL/) is defined as a language combining sublanguages of the OWL Web Ontology Language (OWL DL and Lite) with those of the Rule Markup Language (Unary/Binary Datalog).

The specification was submitted to W3C in May 2004 by the National Research Council of Canada, Network Inference (since acquired by webMethods), and Stanford University in association with the Joint US/EU ad hoc Agent Markup Language Committee.

Compared to Description Logic Programs (DLP), a slightly earlier proposal for integrating description logic and Horn rule formalisms by an overlapping authoring team, SWRL takes the opposite integration approach: DLP can be seen as the 'intersection' of description logic and Horn logic; SWRL, as roughly their 'union'. For DLP, the resulting rather inexpressive language corresponds to a peculiar looking description logic imitating special rules. It is hard to see the DLP restrictions, which stem from Lloyd-Topor transformations, being either natural or satisfying. On the other hand, SWRL retains the full power of OWL DL, but adds rules at the price of undecidability and a lack of complete implementations, although the SWRL Tab of

[1] For a description of the term "webizing" see e.g. http://www.w3.org/DesignIssues/Webize

Protégé has become quite popular (http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTab).

Rules in SWRL are of the form of an implication between an antecedent (body) conjunction and a consequent (head) conjunction, where description logic expressions can occur on both sides. The intended interpretation is as in classical first-order logic: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

## *R2ML*

R2ML (http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=R2ML) was developed as a subproject in the EU Network of Excellence REWERSE (http://oxygen.informatik.tu-cottbus.de/rewerse-i1/). The R2ML project is about the design of integrity and derivation rules on the basis of the Rule Markup Language (RuleML) and the Semantic Web Rule Language (SWRL). R2ML defines a general markup framework for integrity rules, derivation rules, production rules and reaction rules. Rule concepts are defined with the help of MOF/UML, a subset of the UML class modeling language proposed by the Object Management Group (OMG) for the purpose of 'meta-modeling', i.e. for defining languages conceptually on the level of an abstract (semi-visual) syntax. From these MOF/UML language models concrete markup syntax is obtained by applying a mapping procedure for generating corresponding languages from parameterized schemas.

## *W3C RIF*

The W3C Rule Interchange Format (RIF) Working Group (http://www.w3.org/2005/rules/wiki/RIF_Working_Group) is an effort, influenced by RuleML, to define a standard *Rule Interchange Format* for facilitating the exchange of rule sets among different systems and to facilitate the development of intelligent rule-based application for the Semantic Web. For these purposes, *RIF Use Cases and Requirements* (RIF-UCR) have been developed. The RIF architecture is conceived as a family of languages, called *dialects*. A *RIF dialect* is a rule-based language with an XML syntax and a well-defined semantics.

So far, the RIF working group has defined the *Basic Logic Dialect* (RIF-BLD), which semantically corresponds to a Horn rule language with equality. RIF-BLD has a number of syntactic extensions with respect to 'regular' Horn rules, including F-logic-like frames, and a standard system of built-ins drawn from *Datatypes and Built-Ins* (RIF-DTB). The connection to other W3C Semantic Web languages is established via *RDF and OWL Compatibility* (RIF-SWC). Moreover, RIF-BLD is a general *Web language* in that it supports the use of IRIs (Internationalized Resource Identifiers) and XML Schema data types. The RIF Working Group has also defined the *Framework for Logic Dialects* (RIF-FLD), of which RIF-BLD was shown to be the first instantiation. RIF-FLD uses a uniform notion of terms for both expressions and atoms in a Hilog-like manner.

Current efforts of the RIF Working Group are expected to introduce a *Core* (RIF-Core) in the intersection of RIF-BLD and a new *Production Rule Dialect* (RIF-PRD) influenced by OMG's PRR, which can then be further extended or supplemented by reaction rules.

## Semantic Web Rule Languages

In contrast to the XML-based rule markup languages in the previous section, the Semantic Web rule languages described in this section are human-readable rule languages, using an ASCII syntax based, e.g., on the ISO Prolog syntax standard. Typically, they are designed as compact presentation languages for human consumption. While they may be serialized in an XML-based rule markup language such as RuleML or RIF, e.g. for interchange purposes, they can also be employed directly: dynamically interpreted by platform-specific rule engines (at runtime) or statically translated into executable code (at compile time).

### *TRIPLE*

TRIPLE (http://triple.semanticweb.org/) was designed as a practical rule language for linked-data applications. It is an RDF query, inference, and transformation language for the Semantic Web extending F-logic with modules. TRIPLE rules have been used to implement RDFS and other schema languages.

### *N3 / Turtle*

Notation3 (w3.org/TeamSubmission/n3/), more commonly known as N3, is a shorthand non-XML serialization of Resource Description Framework (RDF) models, designed with human readability in mind: N3 is much more compact and readable than RDF/XML serializations. N3 has several features that go beyond the serialization of RDF models, such as support for RDF-based rules. Supporting the triple pattern syntax of SPARQL, Turtle (w3.org/TeamSubmission/turtle/) is a simplified, RDF-only subset of N3.

### *Jena Rules*

The default representation format in Jena (jena.sourceforge.net/ ) for a rule in the rule-based reasoner is a Java Rule object with a list of body terms (premises), a list of head terms (conclusions) and an optional name and an optional direction. However, in Jena2 a rather simple parser is included which allows rules to be specified in reasonably compact form in text source files.

### *Prova*

Prova (http://www.prova.ws/) is both a (Semantic) Web rule language and a highly expressive distributed (Semantic) Web rule engine which supports complex reaction rule-based workflows, rule-based complex event processing, distributed inference services, rule interchange, rule-based decision logic, dynamic access to external data sources, Web Services, and Java APIs. Prova follows the spirit and design principles of the W3C Semantic Web initiative and combines declarative rules, ontologies and inference with dynamic object-oriented programming and access to external data sources via query languages such as SQL, SPARQL, and XQuery. One of the key advantages of Prova is its separation of logic, data access, and computation as well as its tight integration of Java, Semantic Web technologies and enterprise service-oriented computing and complex event processing technologies.

## DESIGN AND CHARACTERISTICS OF WEB RULE LANGUAGES

General requirements that need to be addressed by a rule markup language include semantic expressiveness and clarity, computational efficiency and Web scalability, machine-readable and machine-interpretable syntaxes, usability by both human users and automated agents, compact representation, interchangeability with other formats, means for serialization and persistence, as well as tool support in authoring, parsing/generating, and verifying rules. An important property that refers to development-time software engineering quality is the extensibility of the language and its interoperability with other representation formats. In this section, general language design principles, together with a selection of four important issues and criteria for rule markup language design, are identified and characteristics of the current rule markup languages RIF, RuleML, R2ML, SWRL (DAML Rules) as well as specific Semantic Web rule languages are exemplified with respect to them. Further design issues and requirements for Web rule languages have been elaborated in, e.g., (Wagner et al, 2005), (Bry and Marchiori, 2005), (Boley, 2007), and (Paschke, 2007).

### Language Design Principles

Given the large design space of rule languages and rule concepts, the specification of a rule markup language is a difficult integration and conceptualization challenge that calls for balancing many (interrelated) design choices with respect to semantics, syntax, and pragmatics. In this subsection, we will raise four (markup) language design principles and will illustrate the actual design choices of the current rule markup languages with examples.

#### *1. Criteria of Good Language Design*

Rule markup language should be clear, compact, precise and easily adaptable. They should strive to fulfill typical criteria for good language design (Codd, 1971) - as known from logic, databases and programming - such as minimality, referential transparency and orthogonality:

- *Minimality* requires that the language provides only a small set of needed language constructs, i.e., the same meaning cannot be expressed by different language constructs

- *Referential transparency* is fulfilled if the same language construct always expresses the same semantics regardless of the context in which it is used

- *Orthogonality* asks for pairwise independent language constructs, thus permitting their meaningful systematic combination

The RuleML family follows these design principles as far as possible and provides only a set of needed language constructs which can be applied in every meaningful combination. This leads to a compact homogeneous syntax which is easier to maintain, learn, read and understand by end users, as well as easy to process automatically by machines (e.g. translators).
SWRL and RIF, which build on RuleML, basically follow this compact minimalistic design approach. However, SWRL introduces a more fine-grained distinction of constructs than RuleML, e.g. of Atoms into various types of specialized atoms such as *classAtom*, *datarangeAtom*, and *invidiualPropertyAtom*, which can be formed from unary predicates (classes), binary predicates (properties), and equalities or inequalities.
R2ML introduces further differentiated types of terms and atoms. This leads to a rich structure-preserving syntax with many highly specialized constructs. For instance, variables in R2ML are provided in the form of *ObjectVariable* (i.e. variables that stand for objects), *DataVariable* (i.e. variables that stand for data literals), and *GenericVariable* (i.e. variables that do not have a type), whereas RuleML (as well as SWRL and RIF) only provide a generic *Var* construct. Like RuleML, R2ML defines the notion of an individual (constant) and distinguishes between objects and data with the notions of an *object name* and *data value*.
The main design goal of the specific Semantic Web rule languages such as TRIPLE, Jena, and (following ISO Prolog syntax) Prova is to provide a terse scripting syntax with a minimal set of needed constructs. RuleML's POSL syntax combines and extends the terse ISO Prolog and F-logic syntaxes.

## *2. Different Syntactic and Semantic Layers*

A complete specification of Web rule languages consists of a formalization of their syntax, semantics and, often left implicit, pragmatics. As implied by their name, the syntax of markup languages always includes the concrete syntax of (XML) markup, perhaps indirectly through other languages such as via RDF/XML. Often, there is another more or less concrete syntax such as a compact shorthand or presentation syntax, which may be parsed into the XML markup. While a presentation syntax can already disregard certain details, an abstract syntax systematically replaces character sequences with abstract constructors, often in a (UML) diagram form or as an abstract syntax tree (AST). Together with different token dictionaries, it can be used to generate corresponding concrete syntaxes. The semantics is formalized in a model-theoretic, proof-theoretic, or procedural manner, sometimes in more than one. When rules and speech-act-like performatives, such as queries and answers, are transmitted between different systems, their pragmatic interpretation, including their pragmatic context, becomes relevant, e.g. in order to explain the effects of performatives - such as the assertion or retraction of facts - on the internal knowledge base (Paschke et al, 2007).

A general distinction of three modeling layers can be adopted from OMG's model driven architecture (MDA) engineering approach (http://www.omg.org/mda/):

- A platform specific model (PSM) which encodes the rule statements in the language of a specific execution environment

- A platform independent model (PIM) which represents the rules in a common (standardized) interchange format, a rule markup language

- A computational independent model (CIM) with rules represented in a natural or visual language

The *CIM level* comprises visual and verbal rendering and rule modeling, e.g. via graphical representation or a controlled natural language syntax for rules, mainly intended for human consumption. Graphical representations such as UML diagrams or template-driven/controlled languages can also be used as presentation languages.

In order to facilitate rule modeling, R2ML provides a UML-based Rule Modeling Language (URML) (Lukichev and Wagner, 2006) which allows visual rule modeling based on UML class models and OCL constraints. RuleML on the CIM level provides several tools that use a controlled natural rule language approach. Among them are TRANSLATOR (Hirtle, 2006), which is based on Attempto Controlled English (ACE) (Fuchs et al, 2006), the open source Reaction RuleML editor (http://ibis.in.tum.de/research/ReactionRuleML/index.htm#editor), which uses a template driven approach, and the commercial RuleManager (Ensig, 2007). The Protégé tool (http://protege.stanford.edu/) provides facilities for modeling SWRL rules, but only on the PIM level, i.e. rules are directly written in the concrete SWRL XML syntax. RIF, being a rather new standard under development, currently does not provide any such tool support.

The *PIM level* should enable platform-independent machine interpretation, processing, interchange and translation into multiple PSM execution syntaxes of concrete rule engines. Hence, the concrete XML (or RDF/XML-based) syntax of a Web rule language such as RuleML, SWRL or R2ML resides on this level, whereas the abstract syntax is on the borderline between the PIM and CIM levels. The abstract syntax can be defined, e.g., with the help of a suitably general grammar definition language such as the EBNF formalism, used, e.g., in the definition of the abstract syntax of OWL, RuleML, RIF, and SWRL, or with the help of a MOF/UML model, as, e.g., in PRR, R2ML, and RuleML. (Wagner et al, 2004) (Giurca and Wagner, 2005).

The *PSM level* is the result of translating/mapping PIM rule (interchange) languages into execution syntaxes which can be directly used in a specific execution environment such as a rule engine. A general distinction can be made between a compiled language approach, where the rules are statically translated into byte code (at compile time), as e.g. done in the rule engines Take (http://code.google.com/p/take/) and Drools (www.jboss.org/drools/) versus interpreted scripting languages, which are dynamically interpreted (at run-time), as e.g. in the rule engines Prova (Paschke, 2006b) and OO jDREW (Ball et al., 2005). While the compiled approach has obvious efficiency benefits, the interpreted approach is more dynamic and facilitates, e.g., updates at run-time. Often, Semantic Web Rule Languages are directly executable by their respective rule engines; hence reside on the PSM level. As an intermediate step between the concrete PSM level and the PIM level an abstract representation is often introduced, such as N3, which provides an abstract rule syntax based on the RDF syntax, or

POSL and Prova, which both provide ANTLR grammars (http://www.jdrew.org/oojdrew/demo/translator, http://www.prova.ws/gram.html) which are transformed into ASTs as the basis for further translation into interchange markup languages such as RuleML or other, specific execution formats.

The correct execution of an interchanged PIM-level rule set serialized in a rule markup language depends on the semantics of both the rule program and the platform-specific rule inference engine (IE). To address this issue, the IE and the interchanged rule set must reveal their intended/implemented semantics. This may be solved via explicit annotations based on a common vocabulary, e.g. an (Semantic Web) ontology which classifies the semantics. Annotations describing the semantics of an interchanged rule set could even be used to find appropriate IEs on the Web to correctly and efficiently interpret and execute the rule program; for example, (1) by configuring the rule engine for a particular semantics in case it supports different ones, (2) by executing an applicable variant of several interchanged semantic alternatives of the rule program, or (3) by automatic transformation approaches which transform the interchanged rule program into a rule program with an applicable semantics; cf. XTAN (http://www.w3.org/2008/02/xtan/). Another approach is to specify additional meta test cases for testing typical properties of well-known semantics, where by the combination of succeeded and failed meta tests the unknown semantics of an IE can often be uniquely determined (Paschke, 2006).
We remark that, traditionally, rule-based systems have been supported by two types of inferencing algorithms: forward-chaining and backward-chaining. A general rule markup language, as a lingua franca, should support translation and interpretation of both reasoning directions, perhaps again using pragmatic annotations (where by default chaining should be bidirectional, as with the `direction` attribute in RuleML).

Independently from the semantics of an interchanged rule program, the pragmatic context in which the interchange takes place is important for the target environment, in order to know how the received information should be used and which actions should be taken with respect to the pragmatic aspects. A standard nomenclature of pragmatic performatives is defined by the Knowledge Query and Manipulation Language (KQML) (www.cs.umbc.edu/kqml/) and the FIPA Agent Communication Language (ACL) (FIPA, 2000), which define several speech-act-theory-based communicative acts.

## *3. Modular Specialized Schema Layers vs. Flat General Schema*

There are two basic design principles for the concrete rule markup syntax. The language (or language family) may be implemented in one flat (monolithic) general XML schema or in a layered structure, where semantically related constructs are defined within separate modules that are added to the different language layers of the Web rule language (cf. http://www.ruleml.org/modularization/)[2]. This leads to a hierarchical structure where higher language layers build on sublayers and add more expressiveness by extending them. The layers

[2] This modularization and layering is implemented by different XML schema documents which are imported and extended

are not necessarily organized around expressiveness/efficiency to the language core.

R2ML follows the first approach and provides one quite large, flat XML schema for all different rule types and language constructs. In contrast, RuleML (also SWRL and RIF) follow the layered design principle and define new constructs within separate modules which are added to the respective layers in the RuleML language family. The layered and uniform design makes it easier to learn the language and to understand the relationship between the different features, and it provides a certain guidance to users who might be interested only in a particular subset of the features and who do not need support for the full expressiveness of the language. The modularization allows for easy extension of the language's representation capabilities, using the extensibility of XML Schema (e.g. a `redefine` of an XML Schema group definition), without breaking the core language standard. This development path provides a stable, useful, and implementable language design for rule developers to manage the rapid pace of change on the Semantic Web and modern rule systems. Apart from that, modules facilitate the practical and extensible development of a rule language family by bundling language constructs into layers which can be developed, compiled, tested and managed separately. The modularization also enforces the principle of information hiding and can provide a basis for data abstraction. However, a monolithic schema is easier to read by humans than an unevenly modularized one and, by now, some of the extant XML processing tools and editors do not fully support modular XML Schema definitions. This calls for flattening a layered schema on demand via automatic modular-to-monolithic translators, thus combining the advantages of modular development and maintenance with the advantages of monolithic delivery (for some validators and (object model) transformers, e.g. based on JAXB https://jaxb.dev.java.net/).

## *4. XML Elements vs. Attributes*

A general question regarding the implementation of a concrete rule markup language is where to use XML elements and where attributes to define the rule constructs and the rule information content. A general discussion on this element-vs.-attribute issue can be found in the OASIS Cover pages (http://xml.coverpages.org/elementsAndAttrs.html):

- If the information in question could be itself marked up with elements, put it in an element.

- If the information is suitable for attribute form, but could end up as multiple attributes of the same name on the same element, use child elements instead.

- If the information is required to be in a standard DTD-like attribute type such as ID, IDREF, or ENTITY, use an attribute.

- If the information should not be normalized for white space, use elements. (XML processors normalize attributes in ways that can change the raw text of the attribute value.)

(cf. http://www.ibm.com/developerworks/xml/library/x-eleatt.html)

Accordingly, RuleML's general markup conventions provide common principles for its language hierarchy. XML elements are used for representing language constructs as trees while XML attributes are used for distinguishing variations of a given element and, as in RDF, for webizing. Variation can thus be achieved by different attribute values rather than requiring different elements. Since the same attribute can occur in different elements, an orthogonal, two-dimensional classification ensues, which has the potential of quadratic tag reduction.
For example, recent work in RuleML led to orthogonal dimensions extending the RuleML 0.9 role tags for argument, `<arg . . .>`, and slots, `<slot>`. So far, the *unkeyed* `<arg index="...">` was always *ordered*, as indicated by the `index` attribute, and the *keyed* `<slot>` was always *unordered*, as indicated by the lack of an `index` attribute. This was generalized by allowing an optional `index` attribute for both role tags, as shown by the independent distinctions in the following key-order matrix:

| | ordered | unordered |
|---|---|---|
| keyed | <slot index="..."> | <slot> |
| unkeyed | <arg index="..."> | <arg> |

Two extra orthogonal combinations are obtained from this system. First, *keyed*, *ordered* children permit positionalized slots, as in this `cost` fact:

```
<Atom>
 <Rel>cost</Rel>
 <slot index="1"><Ind>item</Ind><Ind>jewel</Ind></slot>
 <slot index="2"><Ind>price</Ind><Data>6000</Data></slot>
 <slot index="3"><Ind>taxes</Ind><Data>2000</Data></slot>
</Atom>
```

Here, slot names `item`, `price`, and `taxes` are provided, e.g. for readability, as well as index positions `1-3`, e.g. for efficiency.

Second, *unkeyed*, *unordered* children permit elements acting like those in a bag (finite multiset), as in this `transport` fact:

```
<Atom>
 <Rel>transport</Rel>
 <arg><Ind>chair</Ind></arg>
 <arg><Ind>chair</Ind></arg>
 <arg><Ind>table</Ind></arg>
</Atom>
```

Here, the arguments are specified to be commutative and 'non-idempotent' (duplicates are kept).
For a general discussion of positional vs. unordered representations see (Boley, 2006).

**R2ML** differs from RuleML, SWRL and RIF as it implements an attribute solution and defines user information content in attributes. For instance a typed object variable "?driver" is represented as follows:

```
<r2ml:ObjectVariable r2ml:name="driver"
r2ml:classID="userv:Driver" />
```

A distinction of positional vs. slotted (named-argument) predicates and functions, as in RuleML and RIF, does not exist in R2ML.

## Expressive Layering

From the perspective of knowledge representation, the main adequacy criterion for a rule markup language is its *epistemological adequacy*, which addresses the ability of the language to represent all relevant knowledge under consideration. Among other representation issues, a general rule interchange format should allow to coherently represent derivation rules, reaction rules, integrity rules, and deontic rules in a homogeneous syntax (Wagner et al, 2005). We use the following general rule classification:

- Facts may comprise various kinds of information such as asserted atoms (formulas), individual-class memberships (of ontology classes), (object-oriented) instances, stored data (e.g., relational, XML), states and event occurrences which might be qualified, e.g., by priorities, temporally, etc.

- Derivation rules infer conclusions from conditions (as in Datalog and Horn logic), where facts (see above) are a special case with constantly true conditions

- Transformation rules specify term rewriting, which can be considered as derivation rules of logics with (oriented) equality

- Integrity rules (or integrity constraints) are assertions which express conditions (or queries) that must always be satisfied. Besides enforcing data integrity, they can constrain, e.g., the rule system structure, its information content, or its behavior:

    - *Structural constraints* (deontic assignments)

    - *State constraints*

    - *Process constraints*

- Deontic rules describe rights and obligations, e.g., of institutions and agents in the context of evolving states (situations triggered by events/actions) and state transitions, where integrity rules (see above) are a special case ('introspectively') affecting the rule set itself

- Reaction rules are (behavioral / action) rules that react on occurred events (external events or changed conditions) by executing actions, where production rules are a special case with events restricted to changed conditions

Because of this broad variety of rules relevant to the Semantic Web, a general rule markup language such as RuleML should have a hierarchical structure that reflects the relevant rule dialects or sublanguages, covering the knowledge representation needs of various subcommunities (cf. Figures 2 and 3).

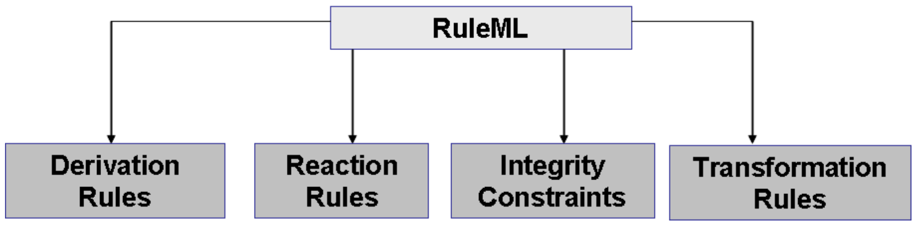

**Figure 2: RuleML Rule Language Family**

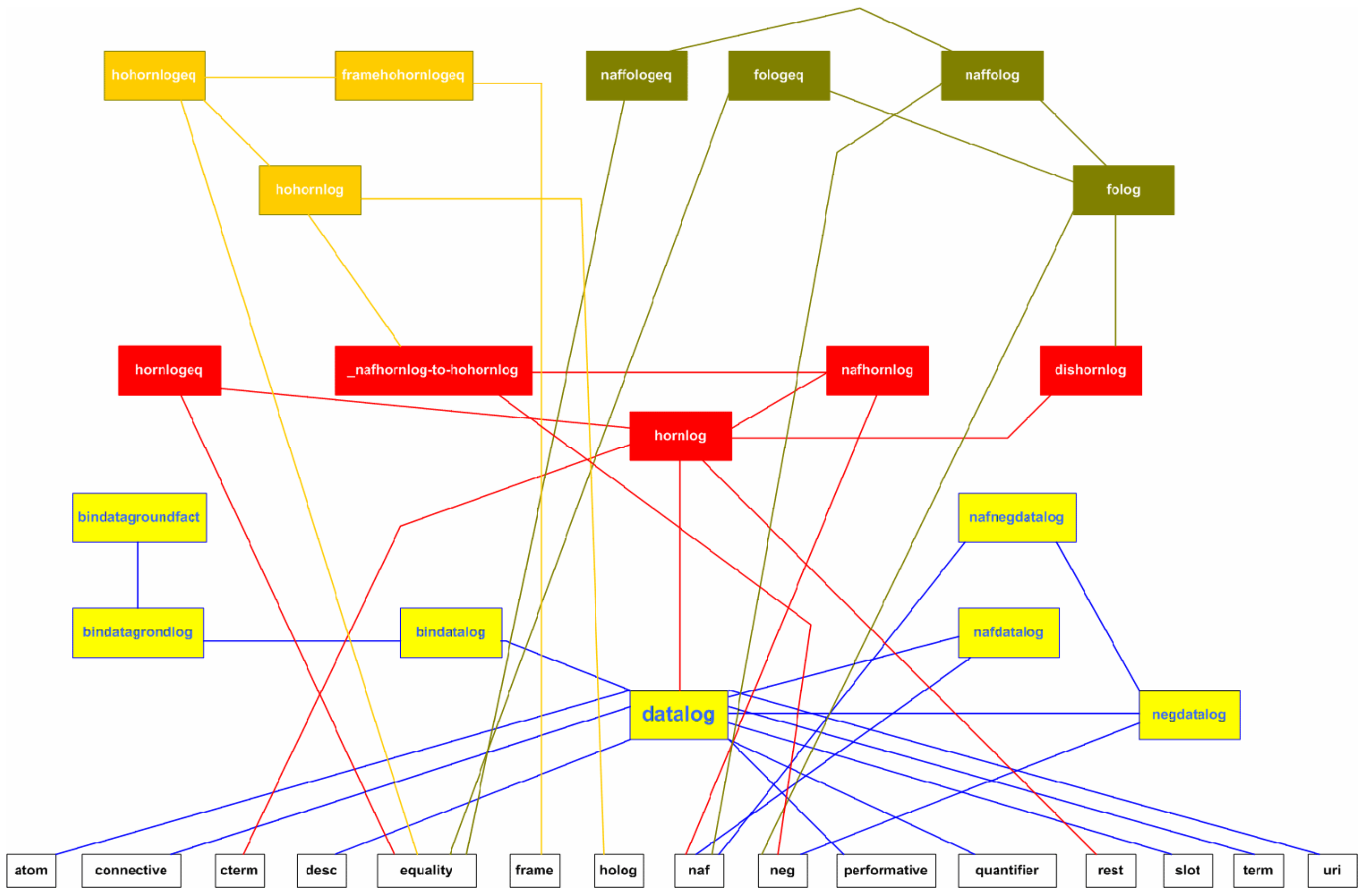

**Figure 3: RuleML 0.91 Derivation RuleML Subfamily**

The main branches contain subbranches, e.g. derivation rules monotonically disallowing or non-monotonically allowing negation as failure (NAF). Branches also exhibit a layering structure with sublanguages of different expressive power, e.g. for monotonic derivation rules proceeding from function-free Horn logic (Datalog) through either full Horn logic or disjunctive Datalog up to first-order logic (FOL). This hierarchy does not form a strict tree (but a directed acyclic graph), i.e. some sublanguages are shared by several more expressive languages; e.g., a sublanguage of conditions is shared by derivation rules and reaction rules. A basic classification of rule languages is introducted by the RuleML family with Derivation Rule Markup Languages (that is, Derivation RuleML), Production Rule Markup Languages (that is, Production RuleML which is a subfamily of Reaction RuleML), Reaction RuleML Markup Languages (that is, Reaction RuleML, http://ibis.in.tum.de/research/ReactionRuleML/), and Transformation Markup Languages (that is, Functional RuleML, http://www.ruleml.org/fun/), as well as other specializations, e.g. dialects for deontic rules (e.g. covered by the RBSLA language (Paschke, 2005)), defeasible rules (Defeasible RuleML, http://defeasible.org/RuleML/) and uncertainty / fuzziness (Fuzzy RuleML, http://www.image.ntua.gr/FuzzyRuleML/).

The coverage of rule markup languages can be roughly divided into logic-based rule formalisms, usually variants of first-order predicate logic, and not logic-based rule formalisms such as (early) production rule systems. In this chapter, we mainly focus on logic-based derivation rule markup languages, but most of the general rule markup languages such as RuleML and R2ML also support the serialization of production rules and reaction rules (e.g., Reaction RuleML covers different types of reaction rule languages). In fact, RuleML and R2ML provide a roughly similar coverage, whereas, e.g., SWRL acts as a more specialized language for homogeneously combining Datalog rules with OWL, hence does not cover, e.g., reaction rules.

## Rules and Object Descriptions

With its URIs (http://www.w3.org/Addressing/), the Web provides a global addressing (URL) and naming (URN) mechanism for objects. A URI consists of a URI scheme name (`http:`, `file:`, etc.) followed by other information that is interpreted relative to the URI scheme. The method for assigning meanings to names varies from one URI scheme to the next, and within each scheme for different sets of names. Each scheme's specification describes how its URIs are intended to be used in certain contexts. As a result, any naming framework must provide mechanisms to enable the creation of new names while avoiding conflicts with existing ones. URIs are also central to the Semantic Web, where RDF metadata are used to describe those objects or resources with classes and properties, which are themselves defined by ontologies (in RDF Schema or OWL). Since SHOE (Heflin et al, 1999), Semantic Web rule languages have associated URIs with constant symbols, predicate names, and other language constructs for reference and disambiguation. For example, the constant symbol `Georgia` could be associated with the unique 'homepage' URI `http://www.georgia.gov` to refer to and disambiguate the state in the Southeastern U.S. in contrast to other entities having the same English name such as the country at the east coast of the Black Sea.

There have been attempts to differentiate the Web notion of URIs into two subnotions, as discussed in (Halpin, 2006): URLs (Uniform Resource Locators), for access, and URNs (Uniform Resource Names), for naming. This distinction is independent from the recent IRI (Internationalized) versions of URIs. In the context of Web knowledge representation, especially for Web rules as explored in POSL, RuleML, and RIF, three central URI uses are emerging (Boley, 2007), given here in the order of further needed research (orthogonal to research in URI normalization (Boley, 2003)).

First, a URI can be used, URL/access-style, for module import (transitive import for nested modules), where it is an error if dereferencing the URI does not yield a knowledge base valid with respect to the expected representation language.

Second, a URI can be used, URN/naming-style, as the identifier of an individual constant in the representation language, where URI dereferencing is not intended as part of the formal knowledge representation. If dereferencing is attempted as part of the metadata about the informal knowledge representation, it should retrieve a descriptive 'homepage' about the individual.

Third, a URI can be used, naming-style, as the identifier of a class, property, relation, or function, and at the same time, access-style, where dereferencing yields (a "`#`"-anchor into) a knowledge base formally defining that identifier (albeit perhaps partially only, as for an RDF Schema knowledge base just giving the superclasses of a class).

Here are examples for the three URI uses in connection with rules.

First, a module of U.S. states could be imported into the current rulebase using the URL/access-style URI `http://modeg.org#us-state`.

Second, the URI `http://en.wikipedia.org/wiki/Pluto` can be used URN/naming-style to refer to a celestial body originally considered a planet, as in this rule specifying its years of planethood (a URI is enclosed in a pair of angular brackets, <... >):

```
planet(<http://en.wikipedia.org/wiki/Pluto>,AD[?year]) :-
 lessThanOrEqual(1930,?year),
 lessThanOrEqual(?year,2006).
```

As part of the formal rule knowledge, the Pluto URI is employed only for naming. The rule can also be employed as metadata about informal knowledge through ('semantic search engine') queries like `planet(?which,2005)`, because one of its solutions will bind `?which` to the URI, whose dereferencing ('clicking') will then retrieve Pluto's Wikipedia entry.

Third, for certain formal purposes a URI like `http://termeg.org#MiniVan` is needed just to provide a name; for other formal purposes, also to provide a total or partial definition found by using that same URI access-style (say, the partial definition of being `rdfs:subClassOf` both `http://termeg.org#Van` and `http://termeg.org#PassengerVehicle`).

In most rule markup languages as well as the specific Semantic Web rule languages, the (user-defined) vocabulary names are globally unique standard identifiers in the form of URI references. Morover, they often define specific builtins for handling URIs such as the SWRL builtins *swrlb:resolveURI (from XQuery op:resolve-uri)*, which is satisfied iff the URI reference in the first argument is equal to the value of the URI reference in the second argument resolved relative to the base URI in the third argument, or *swrlb:anyURI*, which is satisfied iff the first argument is a URI reference consisting of the scheme in the second argument, host in the third argument, port in the fourth argument, path in the fifth argument, query in the sixth argument, and fragment in the seventh argument.
All classes in R2ML are URI references. A class is a type entity for R2ML objects and object variables. Similarily, a *reference property* as well as a *datatype predicate* in R2ML is a URI reference.
RIF uses ***internationalized resource identifiers*** or ***IRI***s (symbol space `rif:iri`) as constants similar to RDF resources.

## Rule-Ontology Combination

A rule markup language should be reasonably integrated with the Semantic Web and should be able to refer to external Semantic Web vocabularies by means of URIs or IRIs, e.g. to use their taxonomic vocabularies as type systems and their individuals as external constants/objects. Domain-independent rules can then be interpreted (relative to each vocabulary) in a domain-dependent manner (with a precise semantics). Accordingly, the original rule set can be much easier interchanged and managed/maintained in a distributed environment. Also, the core Web rule language stays compact and can be easily extended for different vocabulary languages (RDFS, OWL, OWL 2, etc.) on a "per-need-basis".

In recent years, quite an effort has been made to develop a dual expressiveness layering of assertional and terminological knowledge as well as their blends (Antoniou et al., 2005, Kifer et al, 2005). To retain decidability of querying, the *assertional bottom layer* usually consists of Datalog (function-free) assertions, perhaps restricted to unary/binary predicates. For the *terminological bottom layer*, an irreflexive version of RDF Schema's `subClassOf` can be employed, which could later be extended towards the rhoDF (Munoz et al., 2007) fragment of RDF. The two layers can be blended through a hybrid combination (rhoDF classes used as types for Datalog constants and variables, and `subClassOf` defined with order-sorted semantics) or a homogeneous integration (rhoDF classes used as unary predicates in the body of Datalog rules, and `subClassOf` defined as special rules with Herbrand-model semantics).

The higher layers can develop Datalog into Horn (as in Prova's or OO jDREW's hybrid implementation) and FOL (First-Order Logic) assertions, rhoDF into ALC and SHIQ terminologies with classes and properties, and appropriate blends (Rosati, 2006) (Rosati, 2006a), e.g. as advancements of our hybrid DatalogDL (Mei et al, 2007b) or homogeneous ALCuP (Mei et al, 2007). For certain purposes, especially in the early modeling phases, the assertional layers can move even beyond FOL, including towards higher-order and modal logics, as started as part of the RuleML family (Boley, 2006).

To permit the specification of terminologies independent of assertions, a hybrid approach is proposed here adopting the CARIN (Levy and Rousset, 1998) principle as a working hypothesis: A terminological predicate is not permitted in the head of a rule. Intuitively, terminological classes cannot be (re)defined by assertional clauses, because a terminology establishes more stable 'background' knowledge extended by assertions that constitute more volatile 'foreground' knowledge.

Such a hybrid lower layer can use sort restrictions as simple terminological queries in Datalog rule bodies, which in higher layers are extended to terminological queries involving properties, ALC expressions, etc. In the spirit of (Kifer et al, 2005), this should lead to a more realistic Semantic Web architecture with simplified foundations and better computational properties. Our fine-grained bottom-up approach also complements the recent differentiation of OWL-Lite into OWL 1.1 (later: OWL 2) Tractable Fragments (Grau et al., 2006).

The following example uses classes of a `subClassOf` terminology as variable sorts (types) of slightly extended Datalog rules, namely of Horn logic rules employing (unary) functions only for measurement units.

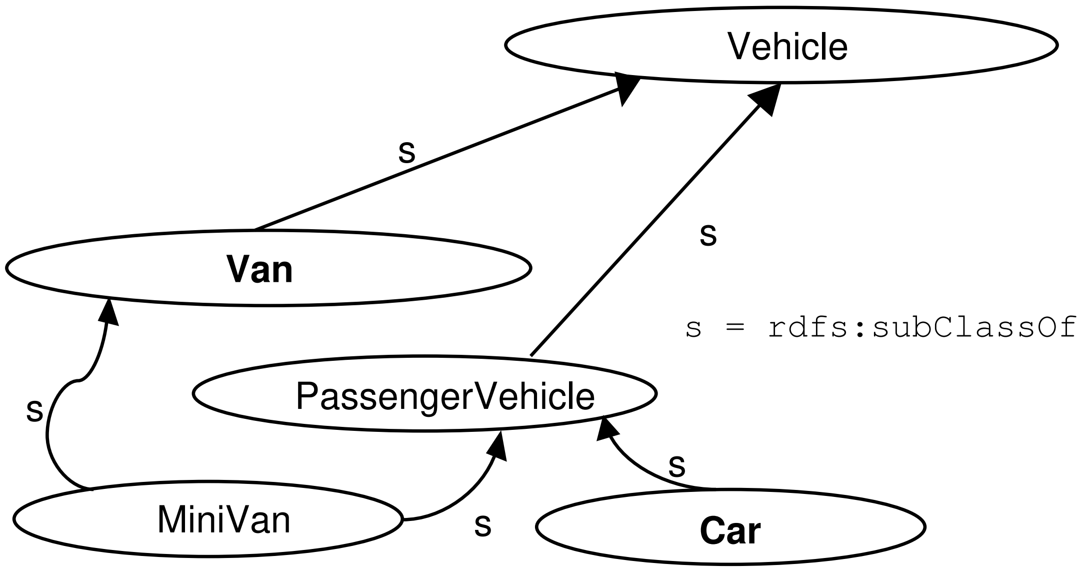

**Figure 4: A Vehicle Terminology (cf. http://www.w3.org/TR/2000/CR-rdf-schema-20000327/)**

The terminology forms a DAG (cf. Figure 4) that introduces `Vehicle`-rooted classes and exemplifies multiple inheritance of `MiniVan` from `Van` and `PassengerVehicle` (the ">" infix is used between a superclass and a subclass):

```
Vehicle > Van
Vehicle > PassengerVehicle
Van > MiniVan
PassengerVehicle > MiniVan
PassengerVehicle > Car
```

The following RuleML/POSL rules specify registration fees for vehicles. The first rule specifies a vehicle variable typed by the `Van` class, while the second refers to the `Car` class (the "`:`" infix is used between a variable and its type):

```
registration(?V:Van,CAD[?R:Decimal]) :-
 emission(?V,CO2[?E]),
 weight(?V,kg[?W]),
 emiweight(CAD[?R],CO2[?E],kg[?W]).

registration(?V:Car,CAD[?R:Decimal]) :-
 emission(?V,CO2[?E]),
 speed(?V,kmh[?S]),
 emispeed(CAD[?R],CO2[?E],kmh[?S]).
```

A `registration` query for a given vehicle class will thus unify only with correspondingly sorted rule conclusions, hence directly branch into the appropriate rule premises (the `emiweight` and `emispeed` premises compute the fees from the emissions as well as the weights and speeds for `Van`s and `Car`s, respectively). The previous section has shown URI-'webized' versions of these terminological classes.

R2ML rules may refer to a vocabulary which can be R2ML's own vocabulary or an imported one (such as RDF(S) and OWL). R2ML's internal default vocabulary is a serialization of a UML fragment of class diagrams. R2ML uses XML Schema datatypes as its standard datatype set. R2ML distinguishes between plain and typed literals and typed and untyped variables. A *DataTerm* is a *DataVariable*, a *DataLiteral*, or a *DataFunctionTerm.* An *ObjectVariable* is a variable that stand for objects of a particular class type, a *DataVariable* is a variable that stand for data literals, and a *GenericVariable* is a variable that does not have a type. For instance, an *ObjectVariable* contains an optional reference to a class which is used as its type:

```
<r2ml:ObjectVariable r2ml:name="driver"
r2ml:classID="userv:Driver" />
```

R2ML allows both typed and untyped individuals. For instance, a *TypedLiteral* consist in a lexical value and a type that is an RDF datatype or a user defined datatype (subclass of rdfs:Literal):

```
<r2ml:TypedLiteral r2ml:typeLiteral="xs:positiveInteger"
r2ml:lexicalValue="90"/>
```

RuleML supports order-sorted terms permitting typed individuals, variables (exemplified above) and data literals (Boley, 2003). Therefore, RuleML/XML defines an optional type attribute for specifying a term's (user-defined) type. Besides referring to the default XML Schema datatypes, typed terms may also link into external object class hierarchies via their fully qualified class names (e.g. Java classes) or taxonomies such as RDF Schema class hierarchies, thus reusing the OO class models and Semantic Web's light-weight ontologies as pluggable external order-sorted type systems. For example:

```
<Var type="rdf://owl:Vehicle">V</Var>
<Ind type="xml://xsd:string">abc</Ind>
<Var type="java://java.lang.Number">X</Var>
```

A *Data* term in RuleML contains a fixed value, like an RDF literal. It may be optionally associated with an XML Schema built-in datatypeusing the xsi:type attribute. For example:

<Data xsi:type="xs:dateTime">2002-10-10T17:00:00Z</Data>

This open order-sorted typing approach of RuleML provides higher levels of abstractions and allows ad-hoc polymorphism with respect to coercion, i.e., automatic type conversion between subtypes, and overloading, i.e., defining multiple cases (rules with the same head except for types) taking different argument types. The ability to integrate external Semantic Web vocabularies, data types and object oriented class hierarchies as type systems provides syntactic expressiveness for easy extension of the language with domain-specific terminologies, and it facilities rule interchange across domain boundaries due to the explicit semantic definition of the used vocabulary, e.g., a Semantic Web ontology.

As in RuleML, RIF provides an optional `type` attribute for typed constants / individuals and a set of default XML Schema primitive data types such as `xsd:long`, `xsd:integer`, `xsd:decimal`, `xsd:string`, `rdf:XMLLiteral` and `rif:text`. For example:

```
<Const type="rif:iri">dc:creator</Const>

<Const type="xsd:string">abc</Const>
```

However, variables are not typed directly in a prescriptive form using, e.g., the `type` attribute in the variable construct to denote that a variable *X* is of type *T*, i.e. *X:T*. Instead, RIF defines classification terms for class memberships, and also for subclass relationships (cf. F-logic's ":" and "::"):

- t#s is a ***membership term*** if t and s are terms.
- t##s is a ***subclass term*** if t and s are terms.

These classification terms are used to describe subclass hierarchies and membership constraints, e.g. expressing that a variable is of a certain class (type).

In SWRL, a homogeneous combination of OWL (OWL DL and OWL Lite) and RuleML (Unary/Binary Datalog), atoms can be of the form *C(x)*, *P(x,y)*, *sameAs(x,y) differentFrom(x,y)*, or *builtIn(r,x,...)* where *C* is an OWL description or data range, *P* is an OWL property, *r* is a built-in relation, *x* and *y* are either variables, OWL individuals or OWL data values, as appropriate. In the context of OWL Lite, descriptions in atoms of the form *C(x)* may be restricted to class names. That is, SWRL defines a rule language on top of OWL ontologies and hence directly supports the definition of class ontologies and their properties which can be used to type variables. For example:

```
<!-- Each person that is a qualified driver can be added to a car rental as
additional driver-->
<ruleml:Implies
 xmlns:ruleml="http://www.ruleml.org/0.91/xsd"
 xmlns:owlx="http://www.w3.org/2003/05/owl-xml"
 xmlns:swrlx="http://www.w3.org/2003/11/swrlx"
 xmlns:srv="http://www.eurobizrules.org/ebrc2005/eurentcs">

 <ruleml:body>
  <swrlx:classAtom>
```

```
    <owlx:Class owlx:name="srv:Rental"/>
    <ruleml:Var>rental</ruleml:Var>
   </swrlx:classAtom>
   <swrlx:classAtom>
    <owlx:Class owlx:name="srv:Person"/>
    <ruleml:Var>person</ruleml:Var>
   </swrlx:classAtom>
   <swrlx:classAtom>
    <owlx:Class owlx:name="srv:QualifiedDriver"/>
    <ruleml:Var>person</ruleml:Var>
   </swrlx:classAtom>
  </ruleml:body>
  <ruleml:head>
   <swrlx:individualPropertyAtom swrlx:property="srv:additionalDriver">
    <ruleml:Var>rental</ruleml:Var>
    <ruleml:Var>person</ruleml:Var>
   </swrlx:individualPropertyAtom>
  </ruleml:head>
</ruleml:Implies>
```

This homogeneous integration approach is adopted by R2ML from SWRL, while in RuleML without OWL one would refer to the external ontology which defines the vocabulary classes and their properties.

## EXTERNAL DATA INTEGRATION AND DATA PROCESSING

Often Web rules refer to or describe functions and queries over data stored in an external database which can be anything from log files to Web sources or relational databases, data warehouses, or XML or RDF databases such as native XML databases or RDF triple stores. The rule language must allow for the direct dynamic integration of these secondary data storages as facts or object values into the rules in order to reduce redundancy and high memory consumption. It should also support outsourcing of expensive (pre-)processing of data to external systems, e.g., of mathematical functions to procedural implementations such as Java, or of SQL/SPARQL aggregation queries (constructive views) to database management systems or RDF triple stores. A tight combination of declarative and object-oriented programming with rich procedural attachments and language built-ins, e.g. for querying, will facilitate the integration of existing functionalities, tools, and external data sources into rule executions at run time. Procedural attachments are procedure calls to external user-defined computational models of a standard programming language, e.g., directly to Java or C# methods. Therefore, procedural attachments are a crucial extension of a modern Web rule language. They permit the combination of the benefits of declarative (rule-based) as well as procedural and object-oriented languages, e.g., to delegate computation-intensive tasks to optimized object code or to invoke procedure calls on object methods which cannot be easily expressed in a declarative rule-based way. Procedural attachments should be supplemented with a typed logic approach with external type systems such as Java or Semantic Web ontologies, e.g. to assign external objects to typed variables, and with mode declarations in order to safeguard the usage of built-ins and calls to external functionalities.

*(Procedural Attachments)* A procedural attachment is a function or predicate whose implementation is given by an external procedure. Two types of procedural attachments are distinguished:

- *Boolean-valued attachments (or predicate attachments)* which call methods that return a Boolean value, i.e., that are of Boolean sort (type).

- *Object-valued attachments (or functional attachments)* which are treated as functions that take arguments and return one or more objects, i.e., that are of a function sort. This also includes access to public object fields.

*(Built-Ins)* Built-in predicates or functions are special restricted predicate or function symbols in the rule language for concrete domains, e.g., built-ins for strings, numerics, Boolean values, date, time, intervals, lists, etc.

All Web rule languages discussed in this chapter provide support for built-ins and some of them also for general procedural attachments.

SWRL provides an extensible library of built-in functions (http://www.daml.org/2004/04/swrl/builtins.html) co-developed with RuleML. SWRL's built-ins approach is based on the reuse of existing built-ins in XQuery and XPath, which are themselves based on XML Schema Part 2: Datatypes. SWRL built-ins are called via a built-in atom, `swrlx:builtinAtom`, which identifies a built-in using the `swrlx:builtin` attribute and lists its arguments as subelements. SWRL built-ins are identified using the http://www.w3.org/2003/11/swrlb namespace, currently also used by RuleML. This is an example of calling the `multiply` built-in:

```
<swrlx:builtinAtom swrlx:builtin="&swrlb;#multiply">

  <ruleml:var>inches</ruleml:var>

  <ruleml:var>feet</ruleml:var>

  <owlx:DataValue owlx:datatype="&xsd;#int">12</owlx:DataValue>

</swrlx:builtinAtom>
```

SWRL does not provide direct support for procedural attachments, but it could easily adopt this feature from Reaction RuleML.

R2ML by default supports SWRL and XPath2 built-ins as predicate names of atoms *r2ml:DatatypePredicateAtom* and symbols of functions *r2ml:DatatypeFunctionTerm.* Functions and operators like addition, subtraction, etc. are translated into corresponding R2ML function terms. The operands of the functions implied by the built-ins are enclosed by *r2ml:dataArguments* and might be class attributes, class operations, data variables, typed variables, or further nested built-in functions. However, mode declarations are missing.

```
<r2ml:DatatypePredicateAtom
  r2ml:datatypePredicate="swrlb:lessThan">
    <r2ml:dataArguments>
       <r2ml:DataVariable r2ml:name="y"/>
       <r2ml:TypedLiteral r2ml:lexicalValue="4"
                          r2ml:datatype="xs:positiveInteger"/>
    </r2ml:dataArguments>
</r2ml:DatatypePredicateAtom>
```

R2ML supports procedural attachments in order to access public data fields of objects which might be bound to object variables.

```
<r2ml:AttributeFunctionTerm r2ml:attributeID="userv:Car.price">

  <r2ml:contextArgument>
   <r2ml:ObjectVariable r2ml:name="car"
                        r2ml:classID="userv:Car"/>
  </r2ml:contextArgument>

</r2ml:AttributeFunctionTerm>
```

For reactive rules such as production rules, R2ML supports assignments of action expressions in order to call object methods as actions in the action part.

```
<r2ml:AssignActionExpression r2ml:propertyID="status">
  <r2ml:contextArgument>
   <r2ml:ObjectVariable r2ml:name="ticket"/>
  </r2ml:contextArgument>
  <r2ml:TypedLiteral r2ml:lexicalValue="Escalate"
                     r2ml:datatypeID="xs:string"/>
</r2ml:AssignActionExpression>
```

The RIF built-ins (http://www.w3.org/2005/rules/wiki/DTB) overlap with the functions and predicates defined in XQuery 1.0 and XPath 2.0 Functions and Operators.

Syntactically, built-in predicates and functions in RIF are enclosed by external terms of the form:

```
 'External' '(' Expr ')'
```

where Expr is a UNITERM, i.e. either a Boolean-valued function expression / predicate or an object-valued functional expression. Since RIF does not support a general typed rule language, it requires special guard predicates for all of its supported datatypes to ensure the correct usage of the arguments of built-ins:

```
External("op:numeric-greater-than"^^rif:iri(
              ?diffdays
              "10"^^xsd:integer))

External(
 "www.w3.org/2007/rif-builtin-predicates#isInteger"^^rif:iri(
                                                    ?diffdays))
```

Note that the above example shows the RIF presentation syntax, not the concrete RIF XML syntax.

RIF-FLD foresees procedurally attached user-defined function terms or predicates to be wrapped as external terms but does not define a concrete approach for calling procedural actions yet. However, it supports frame terms *t[p1->v1 ... pn->vn]* which can be used to describe properties of objects.

The RuleML family (through its Reaction RuleML branch) provides an open flexible approach for pluggable external built-in libraries safeguarded by type and mode declarations. It explicitly denotes the usage by the attribute per="*plain*|*value*|*effect*|*modal*|*builtin*" on functional expressions and atomic relations. A <*Rel*> or <*Fun*> using "*plain*" is left uninterpreted, using "*value*" is interpreted purely for its value, using "*effect*" is interpreted impurely both for its value and its (side-)effect action, e.g. by a procedural attachment, using "*modal*" is interpreted as pure modality, and using "*builtin*" as a built-in.

```
 <!— a call to a builtin function -->
 <Expr>
  <Fun per="builtin" uri="swrlb:stringConcat"/>
  <Var type="java://java.lang.String" mode="+">String1</Var>
  <Var type="java://java.lang.String" mode="+">String2</Var>
 </Expr>
 <!-- a call to a builtin predicate -->
 <Atom>
  <Rel per="builtin" uri="rif:dateTime-equal"/>
  <Var type="xml://xs:dateTime" mode="+">Time1</Var>
  <Var type="xml://xs:dateTime" mode="+">Time2</Var>
 </Atom>
```

The mode ("+": input; "-": output; "?": input or output) and type declarations ensure the correct usage of arguments in built-ins, i.e. that built-ins are called with ground values (not free variables) of the expected types.

RuleML provides a concise integration of procedural attachments. Methods of external object classes can be called, including calls to object constructors and calls to object instance and static object methods as well as access to public object fields. Constructed objects and returned result objects can be assigned to variables. Nested selection patterns can be defined over the result object collections such as “forall ?X where ?X=Person(age  > 30 and age < 40)”.

```
<!-- Assign the constructed Java object to the variable
     Date = java.io.Calendar.getInstance() -->
<Equal>
 <Var>Date</Var>
 <Expr>
  <!-- class -->
  <oid><Ind uri="java://java.util.Calendar"/></oid>
  <!-- constructor -->
  <Fun per="effect">getInstance</Fun>
 </Expr>
</Equal>


<!-- Use the bound object of the variable and call a function
     "isSet" of the object -->
<Atom>
 <!-- object previously assigned to Date -->
 <oid><Var>Date</Var></oid>
 <Rel per="effect">isSet</Rel>
 <Data>1</Data>
```

```
</Atom>

<!-- Call a static C# method -->
<Atom>
 <oid><Ind uri="c-sharp://System.Console"/></oid>
 <Rel uri="WriteLine"/>
 <Data>Hello World</Data>
</Atom>
```

Most of the specific Semantic Web rule languages such as Jena and Prova support all "standard" built-ins of Web Rule languages as well as many additional built-ins e.g. for meta interpretations of literals, exception handling, console printouts, collections, iterations/enumerations, object property constraints, or access to system environment properties. For instance, Prova supports several query built-ins to access files, XML data sources via DOM, XPath, and XQuery, RDF data sources via RDF triples and SPARQL and RDFS/OWL ontologies, as well as various homogeneous or heterogeneous inference queries using external DL reasoners such as, e.g., Pellet. Prova also provides a tight and natural Java integration. Methods of classes in arbitrary Java packages can be dynamically invoked from Prova rules. The method invocations include calls to Java constructors creating Java variables and calls to instance and static methods for Java classes as well as public object data fields.

## FUTURE TRENDS

A general rule markup language such as RuleML or RIF covers many different rule types and rule families. Some of the language families such as classical production rules historically only define an operational semantics, while other rule families such as logical rules (see RuleML family in the section about expressive layering) are based on a model-theoretic and/or proof-theoretic semantics. A general research question is whether there exists a unifying semantic framework for all different rule types. Work in this direction is pursued, e.g. in RIF (http://www.w3.org/2005/rules/wiki/FLD) and Reaction RuleML (transactional transition semantics for reaction rules subsuming all other RuleML rules). However, since there is no general consensus on one particular semantics for all expressive rule languages, an exclusive commitment to one particular semantics for a Web rule language should be avoided (even in well-researched fields such as logic programming several semantics such as well-founded semantics and answer set semantics are competing). Nevertheless, for certain subfamilies a preferred semantics can still be given and semantic mappings between rule families be defined.

General rule markup languages need to include practical language constructs which might not (yet) have a standard formal semantics based on classical model-theoretic logic. For instance, procedural calls to external (object) functions, operational systems, data sources and terminological descriptions, are often vital to deal with practical real-world settings of distributed Web applications. Recent research, e.g. in RuleML and RIF-PRD, is done on adopting such practical language constructs without a standard formal semantics but with a non-standard one. While there is a risk that these concessions to non-standard semantics might endanger the benefits of formal semantics for the overall rule language, they turn out to be a crucial means to avoid limitations of standard rule representations in the exploration of rule markup languages. The rule component will rarely run in isolation, but interact with various external components, hence call for functionalities such as efficient object-oriented or relational/SQL-style retrieval and aggregation methods that are common in modern information systems.

Further examples of useful practical constructs are the annotation of rules and rule sets with additional metadata such as rule qualifications, rule names, module names, Dublin Core annotations, etc., which eases, e.g., the modularization of rules into rule sets (bundling of rules), the creation of constructive views over internal and external knowledge (scoped reasoning), as well as the publication and interchange of rules / rule sets on the Web. Advanced rule qualifications such as validity periods or rule priorities might for example safeguard dynamic updates (e.g. the incorporation of interchanged rules into the existing rule base), where conflicts are resolved by rule prioritizations.

Another domain of research is the engineering and maintenance of large rule-based applications, where the rules are serialized and managed in a distributed manner, and are interchanged across domain boundaries. This calls for support of verification, validation and integrity testing (V&V&I), e.g. by test cases that are written in the same rule markup language and are stored and interchanged together with the rule program. A proposal for self-validating rule bases adapting a test-driven development approach from extreme programming in Software Engineering has been made for RuleML (Paschke et al, 2006) and for RIF (Paschke et al, 2005).

## CONCLUSION

In this chapter several important requirements and design choices for a rule markup language have been described. It was shown how the current Web rule language proposals address these issues and what characteristics derive from those solutions. Commonalities as well as differences between the languages were presented and illustrated with concrete examples. Discussions of the advantages and disadvantages of the language design approaches reveal that all approaches legitimately coexist at this stage, as all have their strengths and weaknesses.

## REFERENCES

(Antoniou et al., 2005) Grigoris Antoniou, Carlos Viegas Damasio, Benjamin Grosof, Ian Horrocks, Michael Kifer, Jan Maluszynski, and Peter F. Patel-Schneider. Combining Rules and Ontologies — A Survey. Deliverables I3-D3, REWERSE, http://rewerse.net/deliverables/m12/i3-d3.pdf, March 2005.

(Ball et al., 2005) M. Ball, H. Boley, D. Hirtle, J. Mei, and B. Spencer. The OO jDREW Reference Implementation of RuleML. In Proc. Rules and Rule Markup Languages for the

Semantic Web (RuleML-2005), pages 218–223. LNCS 3791, Springer-Verlag, November 2005.

(Boley, 2003) Harold Boley, Object-Oriented RuleML: User-Level Roles, URI-Grounded Clauses and Order-Sorted Terms", Invited Talk, Springer LNCS 2876, Oct. 2003, 1-16.

(Boley, 2006) Harold Boley, The RuleML Family of Web Rule Languages. Invited Talk, Proc. Fourth Workshop on Principles and Practice of Semantic Web Reasoning, Budva, Montenegro, June 10-11, 2006, pp. 1-15, Springer LNCS 4187.

(Boley, 2007) Harold Boley: Are Your Rules Online? Four Web Rule Essentials. RuleML 2007: 7-24

(Bry and Marchiori, 2005) François Bry, Massimo Marchiori: Ten Theses on Logic Languages for the Semantic Web. PPSWR 2005: 42-49

(Codd, 1971) E. Codd. Alpha: A data base sublanguage founded on the relational calculus of the database relational model. In ACM SIG-FIDET Workshop on Data Description, Access and Control, San Diego, CA., 1971.

(Ensig, 2007) Marco Ensing, The Rule Manager; a graphical business rules environment. Demo at RuleML-2007, http://2007.ruleml.org/docs/Acumen%20Business%20-%20RuleML.pdf, Orlando, Florida, 2007.

(FIPA, 2000) FIPA Agent Communication Language, http://www.fipa.org/, accessed Dec. 2001, 2000.

(Fuchs et al., 2006) Norbert E. Fuchs, Kaarel Kaljurand, Gerold Schneider (2006). "Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces". FLAIRS 2006.

(Giurca and Wagner, 2005) Adrian Giurca, Gerd Wagner: Towards an Abstract Syntax and Direct-Model Theoretic Semantics for RuleML. RuleML 2005: 45-55.

(Grau et al., 2006) Bernardo Cuenca Grau, Diego Calvanese, Giuseppe De Giacomo, Ian Horrocks, Carsten Lutz, Boris Motik, Bijan Parsia, and Peter F. Patel-Schneider. OWL 1.1 Web Ontology Language Tractable Fragments. W3C Member Submission, http://www.w3.org/Submission/owl11-tractable/, Dec. 2006.

(Halpin 2006) Harry Halpin: Identity, Reference, and Meaning on the Web, WWW2006 Workshop on Identity, Reference, and the Web, http://www.ibiblio.org/hhalpin/irw2006/, May 2006.

(Heflin et al, 1999) Jeff Heflin, James Hendler, and Sean Luke: SHOE: A Knowledge Representation Language for Internet Applications, Technical Report CS-TR-4078 (UMIACS TR-99-71). 1999

(Hirtle, 2006) David Hirtle, TRANSLATOR: A TRANSlator from LAnguage TO Rules, Canadian Symposium on Text Analysis (CaSTA), Fredericton, October 2006.

(Kifer et al., 2005) Michael Kifer, Jos de Bruijn, Harold Boley, and Dieter Fensel. A Realistic Architecture for the Semantic Web. In Asaf Adi, Suzette Stoutenburg, and Said Tabet, editors, RuleML, volume 3791 of Lecture Notes in Computer Science, pages 17-29. Springer, 2005.

(Levy and Rousset, 1998) Alon A. Levy and Marie-Christine Rousset. CARIN: A

Representation Language Combining Horn Rules and Description Logics. Articial Intelligence, 104(1-2):165-209, 1998.

(Lukichev and Wagner, 2006) Sergey Lukichev and Gerd Wagner. In: Irina Virbitskaite and Andrei Voronkov (Eds.). Visual Rules Modeling. Proceedings of the 6th International Conference Perspectives of Systems Informatics. Springer. 2006. volume 4378. pp. 467–673. Lecture Notes in Computer Science.

(Mei et al., 2007) Jing Mei, Zuoquan Lin, Harold Boley, ALCuP: An Integration of Description Logic and General Rules, Web Reasoning and Rule Systems, First International Conference, RR 2007, June 2007, Springer LNCS 4524, 163-177.

(Mei et al., 2007b) J. Mei, Z. Lin, H. Boley, J. Li, and V. C. Bhavsar, "The DatalogDL Combination of Deduction Rules and Description Logics", *Computational Intelligence*, August 2007, 23(3), pp. 356-372.

(Munoz et al., 2007) Sergio Munoz, Jorge Perez, and Claudio Gutierrez. Minimal Deductive Systems for RDF. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, ESWC, volume 4519 of Lecture Notes in Computer Science, pages 53-67. Springer, 2007.

(Paschke, 2005) Adrian Paschke: RBSLA A declarative Rule-based Service Level Agreement Language based on RuleML. CIMCA/IAWTIC 2005: 308-314.

(Paschke et al., 2005) Adrian Paschke, Harold Boley, Jens Dietrich, RIF Use Case "Rule Interchange Through Test-Driven Verification and Validation", http://www.w3.org/2005/rules/wg/wiki/Rule_Interchange_Through_Test-Driven_Verification_and_Validation, 2005.

(Paschke, 2006) Paschke, A.: Verification, Validation and Integrity of Distributed and Interchanged Rule Based Policies and Contracts in the Semantic Web, Int. Semantic Web and Policy Workshop (SWPW'06), Athens, Georgia, USA, 2006.

(Paschke, 2006b) Paschke, A.: A Typed Hybrid Description Logic Programming Language with Polymorphic Order-Sorted DL-Typed Unification for Semantic Web Type Systems, OWL-2006 (OWLED'06), Athens, Georgia, USA, 2006.

(Paschke et al., 2006) Paschke, A., Dietrich, J., Giurca, A., Wagner, G., Lukichev, S.: On Self-Validating Rule Bases, Int. Semantic Web Enabled Software Engineering Workshop (SWESE'06), Athnes, Georgia, USA.

(Paschke, 2007) Paschke, A.: Rule-Based Service Level Agreements - Knowledge Representation for Automated e-Contract, SLA and Policy Management, ISBN 978-3-88793-221-3, IDEA Verlag GmbH, München.

(Paschke et al, 2007) Adrian Paschke, Harold Boley, Alexander Kozlenkov, Benjamin Larry Craig: Rule responder: RuleML-based agents for distributed collaboration on the pragmatic web. ICPW 2007: 17-28

(Rosati, 2006) Riccardo Rosati. The Limits and Possibilities of Combining Description Logics and Datalog. In T. Eiter, E. Franconi, R. Hodgson, and Susie Stephens, editors, RuleML, pages 3-4. IEEE Computer Society, 2006.

(Rosati, 2006a) Integration Ontologies and Rules: Semantic and Computational Issues, Reasoning Web, 2006.

(Wagner et al., 2004) Gerd Wagner, Grigoris Antoniou, Said Tabet, Harold Boley: The Abstract Syntax of RuleML - Towards a General Web Rule Language Framework. Web Intelligence 2004: 628-631

(Wagner et al., 2005) Gerd Wagner, Carlos Viegas Damásio, Grigoris Antoniou: Towards a general web rule language. Int. J. Web Eng. Technol. 2(2/3): 181-206 (2005)