RuleML 1.0: The Overarching Specification of Web Rules

Harold Boley[1], Adrian Paschke[2], and Omair Shafiq[3]

[1] Institute for Information Technology, National Research Council Canada
Fredericton, NB, Canada
harold.boley@nrc.gc.ca
[2] Freie Universitaet Berlin, Germany
paschke@mi.fu-berlin.de
[3] University of Calgary, AB, Canada
moshafiq@ucalgary.ca

Abstract. RuleML is a family of languages, whose modular system of XML schemas permits high-precision Web rule interchange. The family's top-level distinction is deliberation rules vs. reaction rules. Deliberation rules include modal and derivation rules, which themselves include facts, queries (incl. integrity constraints), and Horn rules (incl. Datalog). Reaction rules include Complex Event Processing (CEP), Knowledge Representation (KR), and Event-Condition-Action (ECA) rules, as well as Production (CA) rules. RuleML rules can combine all parts of both derivation and reaction rules. This allows uniform XML serialization across all kinds of rules. After its use in SWRL and SWSL, RuleML has provided strong input to W3C RIF on several levels. This includes the use of 'striped' XML as well as the structuring of rule classes into sublanguages with partial mappings between, e.g., Datalog RuleML and RIF-Core, Hornlog RuleML and RIF-BLD, as well as Production RuleML and RIF-PRD. We discuss the rationale and key features of RuleML 1.0 as the overarching specification of Web rules that encompasses RIF RuleML as a subfamily, and takes into account corresponding OASIS, OMG (e.g., PRR, SBVR), and ISO (e.g., Common Logic) specifications.

1 Introduction

Rules on the Web come in various formats and with diverse packaging. Often, however, the semantics of Web-distributed rule content are compatible. In such cases, rulebases can be reused with an interchange technology consisting of a family of canonical rule languages and bi-directional translators between canonical languages and the languages to be interchanged. The need for Web rule interchange has been increasing with the amount of business rules (incl. policies, regulations, laws, ...) in many domains (e.g. finance, engineering, healthcare, ...) on the Web 1.0, 2.0 (Social), and 3.0 (Social Semantic).

RuleML has been designed for the interchange of the major kinds of Web rules in an XML format that is uniform across various rule languages and platforms. It has broad covergage and is defined as an extensible family of languages, whose

M. Dean et al. (Eds.): RuleML 2010, LNCS 6403, pp. 162–178, 2010.

modular system of XML schemas permits rule interchange with high precision, as follows.

When a rulebase is prepared for interchange by a sender,

- it is translated to RuleML if the source document is not in the RuleML format already,
- the Most Specific Schema (MSS) is determined against which the RuleML document can be validated,
- the Internationalized Resource Identifier (IRI) of the MSS is pointed to from the rulebase or is otherwise transmitted along with the rulebase.

When a rulebase is obtained by a receiver,

- it is validated against the same RuleML schema to exclude any too specific MSS assignments and transmission errors,
- it is converted to the local format if the target document is not to be in RuleML anyway.

The RuleML family constitutes a taxonomy of subfamilies, languages, and sublanguages classified through the syntactic power of rules, as reflected by their XML Schema Definitions (XSDs), and through their semantic power, as reflected by their model-theoretic, proof-theoretic, and operational semantics. Often, more syntactic power leads to more semantic power (e.g., the introduction of `Expr`ession syntax pushes Datalog to Horn Logic (Hornlog) models in Section 3.2). Syntactically neutral aspects of semantic power will be expressed by semantic attributes (e.g., by a `negation` attribute for the semantics of `Negation-as-f`ailure in Section 3.5).

Fig. 1, a simplified version of the RuleML taxonomy, shows the semantic subfamilies of *Deliberation* rules for inference and *Reaction* rules for (re)action. Deliberation rules, via *Higher Order Logic (HOL)* and *First Order Logic (FOL)*, subsume *Derivation* rules. Derivation rules subsume Hornlog and Datalog languages and (syntactically) specialize to the condition-less *Fact* and conclusion-less *Query* languages (subsuming *Integrity Constraint (IC)* languages). Reaction rules subsume *Complex Event Processing (CEP)* and *Knowledge Representation (KR)* rules, as well as *Event-Condition-Action-Postcondition (ECAP)* rules. *ECAP* rules specialize to *Event-Condition-Action (ECA)* rules, which themselves specialize to Condition-less *Trigger (EA)* rules and to the rule subfamily of Event-less *Production (CA)* rules. The *RuleML* family also has 'mix-ins' for *Equality* and (oriented) *Rewriting*, as well as for *Naf*. The *Reaction* subfamily has mix-ins for *Event Algebra*, *Action Algebra*, etc.

While not shown in Fig. 1, RuleML languages make use of 'pluggable' libraries of built-ins such as from the Semantic Web Rule Language (SWRL) [HPSB+04] and the Rule Interchange Format (RIF) [PBK10]. There are also entire RuleML languages we cannot further discuss in the confines of this paper, including for uncertainty and fuzzy rules[1] [DPSS08] and for defeasible rules[2] [KBA08].

[1] Fuzzy RuleML: `http://www.image.ntua.gr/FuzzyRuleML`
[2] Defeasible RuleML: `http://defeasible.org`

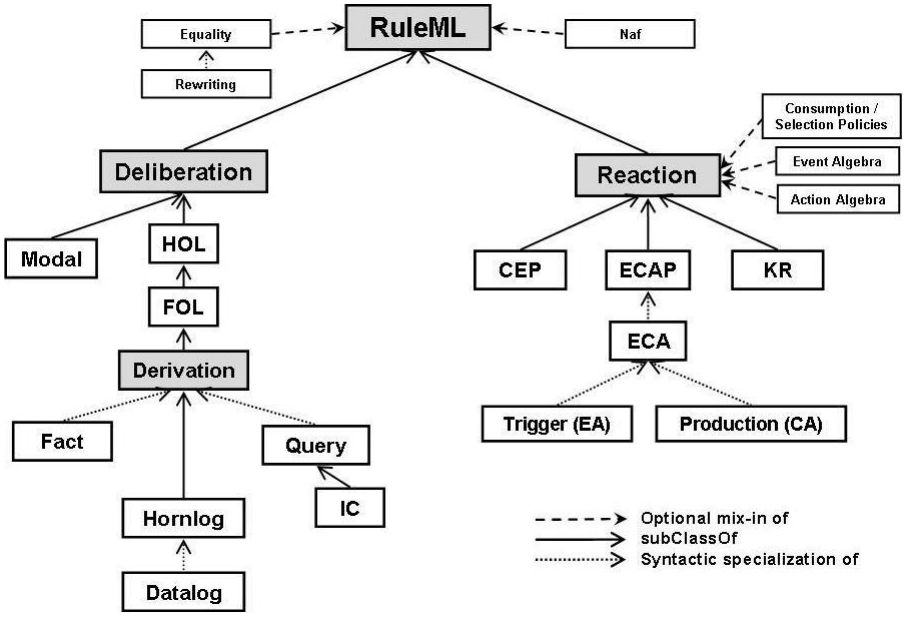

Fig. 1. Taxonomy of RuleML rules

The derivation rule languages have a Datalog language as their kernel. Datalog RuleML is defined over both `Data` constants and `Ind`ividual constants with an optional `iri` attribute for webizing. `Atom`ic formulas have n arguments, which can be positional or slotted (*key* → *term* pairs). Object Oriented Datalog adds optional `type`s (sorts) and RDF-like `oid`s via IRIs. Inheriting all of these Datalog features, Hornlog RuleML adds positional and slotted `Func`tional `Expr`essions as terms. In Hornlog – and other languages – with `Equal`ity, such uninterpreted (constructor-like) functions are complemented by interpreted (equation-defined) functions. This derivation rule branch is extended upward to FOL, including disjunction (`Or`) in conclusions and strong `Neg`ation.

Reaction RuleML syntactically extends the condition (query) part of Derivation RuleML, whose condition-conclusion rules can be seen as 'pure' production rules with conclusions as actions that just assert derived facts. For a discussion of relationships between active and deductive rules see [Wid93]. Reaction RuleML is based on 'pluggable' ontologies (e.g., algebras) of (complex) actions, events, and – in the KR subfamily – situations. Production RuleML defines condition-action rules. Complex Event Processing (CEP) RuleML defines (complex) events and their efficient processing. Reaction RuleML extends production rules with an event-triggering part, syntactically defining ECA rules, and with further semantic extensions, e.g. for CEP rules.

RuleML rules can combine all parts of both derivation and reaction rules. This allows uniform XML serialization across the rules from the taxonomy. A

general `<Rule>` element specifies the kind of rule with a `style` attribute, where shortcuts allow specialized elements such as `<Implies>` and `<Reaction>`.

After its use in SWRL and the Semantic Web Services Language (SWSL) [BBB+05], RuleML has provided input to W3C RIF [BK10a] on several levels. This includes the use of 'striped' XML and the structuring of rule classes into a family of sublanguages with partial mappings between, e.g., Datalog RuleML and RIF-Core [PBK10], Derivation RuleML and RIF Basic Logic Dialect (RIF-BLD), as well as Production RuleML and RIF Production Rule Dialect (RIF-PRD), where RuleML's `<if>` ... `<do>` was adopted as RIF's `<if>` ... `<then><Do>`.

The RIF WG – after having achieved W3C Recommendation status in June 2010 – is scheduled to terminate with the end of September 2010 until an uncertain revival for a possible RIF 2. RIF's standard logic Web rule dialects Core and BLD come with a rigorous model-theoretic semantics, embodying the WG's cascaded design decisions. However, the W3C Core and BLD Recommendations cover only a fraction of the Web rule space and their very rigor gives existing Web rule languages little room for RIF conformance. The RuleML Initiative – whose symposia have been a forum for RIF advances from its beginning – has thus been co-hosting the development of further ("non-standard extensions"[3] or) RIF dialects such as the Core Answer Set Programming Dialect (RIF-CASPD) [HK09] and Semantic Inferencing on Large Knowledge (SILK) [GDK09], using the flexibility-enhancing Framework for Logic Dialects (RIF-FLD) [BK10b], as well as RIF RuleML sublanguages such as Datalog with equality plus externals (Dlex) [Bol09] and the envisioned Reaction Rule Dialect (RRD).

Even languages that will not become ("standard extensions"[3] or) RIF 2 Recommendations themselves can help with Web rule interoperability by consolidating the terrain and acting as connectors to other standards bodies such as OMG and OASIS as well as business rule organizations such as BRF[4] and stakeholders in the private and public sectors. Based on [WATB04] and Production RuleML, members of the Reaction RuleML Technical Group have already contributed to OMG's Production Rule Representation (PRR). RuleML is founding member of the Event Processing Technical Society (EPTS), where it contributes to, and co-chairs, the EPTS Reference Architecture group (ETPS-RA).

This paper, building on our experience with RuleML as the de facto standard for Web rules, discusses the design and definition of RuleML 1.0. As our running example, we will give variations on the discount rule[5] `Implies 1` from the RuleML 1.0 `exa` directory.[6]

The rest of the paper is organized as follows. Section 2 discusses the design rationale of the overarching RuleML family of languages. Section 3 expands on RuleML 1.0 deliberation rules. Section 4 explains its reaction rules. Section 5 presents selected tools and applications of RuleML. Section 6 concludes the paper. Appendix A gives hints on the RuleML 1.0 XSLTs and XSDs.

[3] `http://www.w3.org/2005/rules/wg/charter.html`
[4] `http://www.businessrulesforum.com`
[5] `http://ruleml.org/lib/discount-variations.ruleml`
[6] `http://ruleml.org/1.0/exa/Datalog/discount.ruleml`

2 Design Rationale for RuleML

The specification of an overarching rule markup family with the primary purpose of rule interchange between platform-specific rule languages as well as between other rule standards for, e.g., Semantic Web rules or production rules, requires the balancing of many (interrelated) design choices with respect to semantics, syntax, and pragmatics. For instance, for a rulebase with advanced constructs such as `Naf`, a single predefined semantics would limit its use as it becomes impossible for many rule languages to be compliant with this specific semantics. Similarly, a rigorous syntax which does not support extensibility will necessarily lead to problems if more and more major rule types will be included in this overarching rule markup family. The design rationale for RuleML addresses these requirements.

The RuleML syntax strives for the following widely accepted design principles for good language design:

- Minimality: the language provides only the set of needed language features, i.e., except for macro-like extensibility shortcuts and an order-insensitive abstract role syntax, the same construct is not expressed by different syntax.
- Referential transparency: the same language construct always expresses the same semantics regardless of the context in which it is used.
- Orthogonality: the language constructs are pairwise independent, thus permitting their meaningful systematic combination.

RuleML is designed as an extensible family of languages. In each of these languages it provides a minimal set of needed language constructs which can be applied in every meaningful combination in the respective expressiveness class of the language. The language constructs are structured as modules in the XML Schema definitions. This leads to a clear, compact, and precise design which is easily adaptable, manageable, and extensible.

RuleML, as a general interchange format, can be customized for various semantics of underlying (platform-specific) rule languages that should be represented and interchanged. Although a specific default semantics is always predefined for each RuleML language, the intended semantics of a rulebase can override it by using explicit values for corresponding semantic attributes. For instance, a derivation rulebase represented in Datalog RuleML with `Naf` can be explicitly declared to have Well-Founded (`WF`) or Answer Set (`AS`) semantics, with `AS` as the default (cf. Section 3.5). Moreover, RuleML supports external domain semantics such as ontologies, e.g. RDFS or OWL taxonomies, or class hierarchies, e.g. object oriented models such as UML class models or Java class hierarchies. These can be used as external order-sorted type systems for rule constructs, e.g. variables and constants, giving them an interchangeable and machine interpretable domain semantics. This flexible semantics approach of RuleML allows refining the semantics of a syntactically represented rulebase.

From a pragmatic perspective, the layered RuleML design of Fig. 1 leads to a compact syntax (in terms of language constructs) which is easier to learn, read,

understand, and apply by end users, as well as easier to extend in a modular way with new languages and semantics. The modular family of languages also makes it easy for machines to process RuleML, e.g. by translators that map between platform-specific rule languages and an equivalent RuleML language. Additionally, the pluggable-semantics approach supports correct machine understanding and interpretation.

In summary, these design principles allow the overarching RuleML specification to evolve into a standard for rule interchange that provides full coverage of all major rule languages and their underlying semantics while still being an easily usable and further extensible interchange language. A more detailed discussion of the design principles of RuleML and how it compares to other rule markup and Semantic Web languages can be found in [PB09a].

3 Deliberation Rules

This section describes deliberation rules with a focus on derivation rules, proceeding bottom-up from Datalog. The `inference-style Rule` element `<Rule style="inference">` can be equivalently shortcut to the `<Implies>` element.

3.1 Datalog RuleML

Datalog [CGT89] is at the core of many rule languages and is close to relational databases with recursive views. Datalog RuleML is defined over both `Data` constants and `Ind`ividual constants with an optional attribute, `iri`, for webizing. RuleML's `Rel`ational `Atom`s have $m + n$ arguments ($m \geq 0$, $n \geq 0$), where m arguments are positional and n are slotted ($key \rightarrow term$ pairs). In Datalog RuleML, terms (e.g. used as positional arguments and slot fillers) can only be constants or `Var`iables. Datalog RuleML also has optional RDF-like `type` (on constants and variables) and `oid` attributes via IRIs. It allows for an `Equal`ity extension, e.g. to call built-ins from 'pluggable' libraries.

To initialize our running example, let us consider Datalog rule `Implies 1` for deriving discounts, with the ternary `Rel`ation `discount` and the unary `premium` and `regular` all being positional. Three versions are given in the columns, where the order of role-tagged children does not matter, and for skipped `<if>`/`<then>` role stripes the first child is understood as the `<if>` role, the second as `<then>`:

```
<!-- Implication Rule 1:
     Backward notation of 'then' and 'if' roles, as in Logic Programming, and forward notation
     using natural 'if' ... 'then' order, as in textbook logic, with exact same meaning

"The discount for a customer buying a product is 5.0 percent
if the customer is premium and the product is regular."

Notice that the ternary discount relation is applied via an Atom.
Furthermore, a Data constant can syntactically be an entire phrase
like "5.0 percent". It will unify only with variables and with Data
having exactly the same spelling (incl. spaces)
-->
```

```
<Implies>
  <then>
    <Atom>
      <Rel>discount</Rel>
      <Var>cust</Var>
      <Var>prod</Var>
      <Data>5.0 percent</Data>
    </Atom>
  </then>
  <if>
    <And>
      <Atom>
        <Rel>premium</Rel>
        <Var>cust</Var>
      </Atom>
      <Atom>
        <Rel>regular</Rel>
        <Var>prod</Var>
      </Atom>
    </And>
  </if>
</Implies>
```

```
<Implies>
  <if>
    <And>
      <Atom>
        <Rel>premium</Rel>
        <Var>cust</Var>
      </Atom>
      <Atom>
        <Rel>regular</Rel>
        <Var>prod</Var>
      </Atom>
    </And>
  </if>
  <then>
    <Atom>
      <Rel>discount</Rel>
      <Var>cust</Var>
      <Var>prod</Var>
      <Data>5.0 percent</Data>
    </Atom>
  </then>
</Implies>
```

```
<Implies>

  <And>
    <Atom>
      <Rel>premium</Rel>
      <Var>cust</Var>
    </Atom>
    <Atom>
      <Rel>regular</Rel>
      <Var>prod</Var>
    </Atom>
  </And>


  <Atom>
    <Rel>discount</Rel>
    <Var>cust</Var>
    <Var>prod</Var>
    <Data>5.0 percent</Data>
  </Atom>

</Implies>
```

A slotted variant of our example uses pairs $key \rightarrow term$ in the conclusion's 3-ary relation, and represents them as metaroles `<slot>`*key term*`</slot>` (we will continue the `<then>` ... `<if>` version, in the first column above, and elide the unchanged condition, where slots would not add much to the readability of unary relations):

```
<Implies>
  <then>
    <Atom>
      <Rel>discount</Rel>
      <slot><Data>buyer</Data><Var>cust</Var></slot>
      <slot><Data>item</Data><Var>prod</Var></slot>
      <slot><Data>rebate</Data><Data>5.0 percent</Data></slot>
    </Atom>
  </then>
  <if> . . . </if>
</Implies>
```

A typed variant of our initial example can use `Var`iables with the attribute `type`, whose values are IRIs pointing to ontological class definitions on the Web defined in RDFS and OWL:

```
<Implies>
  <then>
    <Atom>
      <Rel>discount</Rel>
      <Var type="http://xmlns.com/foaf/spec/#term_Person">cust</Var>
      <Var type="http://daml.org/services/owl-s/1.0/ProfileHierarchy.owl#Product">prod</Var>
      <Data>5.0 percent</Data>
    </Atom>
  </then>
  <if> . . . </if>
</Implies>
```

3.2 Hornlog RuleML

Horn logic [Mak87] is the pure kernel of Prolog-like rule languages. In RuleML, the corresponding Hornlog sublanguage is regarded as an extension of Datalog RuleML, in particular of its `Atom`s: Besides constants and variables, Hornlog

RuleML allows positional and slotted Functional Expressions as terms in Atoms and, recursively, in other Exprs. Expressions can be uninterpreted, using an attribute per with filler "copy" or interpreted, using it with filler "value". Other per fillers are "effect", for (side-)effectful Expressions, and "modal", for modal Exprs.

We refine the initial example by introducing an uninterpreted Expr representing the constructor term percent[5.0], thus proceeding from Datalog to Horn logic, for XSDs and Herbrand models (we again elide the unchanged condition):

```
<Implies>
  <then>
    <Atom>
      <Rel>discount</Rel>
      <Var>cust</Var>
      <Var>prod</Var>
      <Expr><Fun per="copy">percent</Fun><Data>5.0</Data></Expr>
    </Atom>
  </then>
  <if> . . . </if>
</Implies>
```

3.3 FOL RuleML

First Order Logic (FOL) [End01] has been widely used as a knowledge representation language. FOL RuleML is an extension of Hornlog RuleML mainly adding classical negation and (explicit) quantifiers. An earlier version of FOL RuleML is part of the W3C member submission SWRL FOL.[7]

We modify our initial example as follows:

```
<!--
"A customer receives either a discount of 5.0 percent for buying a product
or a bonus of 200.00 dollar if the customer is premium and the product is regular."

Notice that an 'eXclusive or' is used to shortcut
And(Or(A,B),Not(And(A,B))) to Xor(A,B) in the conclusion.
-->

<Implies>
  <then>
    <Xor>
      <Atom><Rel>discount</Rel><Var>cust</Var><Var>prod</Var><Data>5.0 percent</Data></Atom>
      <Atom><Rel>bonus</Rel><Var>cust</Var><Data>200.00 dollar</Data></Atom>
    </Xor>
  </then>
  <if> . . . </if>
</Implies>
```

3.4 RuleML with Equality

Logics with a distinguished equality predicate [Nie07] have been used for specification languages, where equality has been kept symmetric (via paramodulation) or become oriented (via term rewriting or narrowing). In RuleML, Equality formulas act as an extension to sublanguages such as Datalog RuleML, Hornlog

[7] http://www.w3.org/Submission/2005/01

RuleML, and FOL RuleML. `Equal` has an `oriented` attribute whose `"no"` value is assumed as the default.

We modify our initial example as follows:

```
<!--  Equational Rule 1':
      Conditional oriented equation returns rewritten value of first (left) Equal element
      ('call-by-value'-interpreted binary discount function applied via Expr)
      through second (right) Equal element (an alphanumeric Data value)
-->

<Implies>
  <then>
    <Equal oriented="yes">
      <Expr><Fun per="value">discount</Fun><Var>cust</Var><Var>prod</Var></Expr>
      <Data>5.0 percent</Data>
    </Equal>
  </then>
  <if> . . . </if>
</Implies>
```

3.5 Naf RuleML

Besides strong `Neg`ation in FOL RuleML (cf. Section 3.3), Deliberation RuleML also allows `Negation-as-failure`, as used in Logic Programming. This Naf RuleML can be parameterized for `A`nswer `S`et (`AS`) semantics (subsuming stable model semantics) and for `W`ell-`F`ounded (`WF`) semantics, using a semantic attribute, `negation`, on the enclosing `Rulebase`, whose default value is `AS`, accommodating RIF-CASPD [HK09].

The following `Rulebase` example enforces `W`ell-`F`ounded semantics for `Naf`s in the conditions of `discount` rules such as to exclude `late-paying` customers:

```
<Rulebase negation="WF">
  <Implies>
    <then>
      <Atom><Rel>discount</Rel><Var>cust</Var><Var>prod</Var><Data>5.0 percent</Data></Atom>
    </then>
    <if>
      <And>
        <Naf>
          <Atom><Rel>late-paying</Rel><Var>cust</Var></Atom>
        <Naf>
        . . .
      </And>
    </if>
  </Implies>
  .  .  .
</Rulebase>
```

4 Reaction Rules

Reaction rules are concerned with the invocation of actions in response to events and actionable situations [PB09b]. They state the conditions under which actions must be taken and describe the effects of action executions. In the last decades various reaction rule languages and rule-based event processing approaches have been developed, which for the most part have been advanced separately. The Reaction RuleML subfamily addresses the four major reaction rule types:

- Production Rules (Condition-Action rules) in the Production RuleML subfamily
- Event-Condition-Action (ECA) rules in the ECA RuleML subfamily
- Rule-based Complex Event Processing (complex event processing reaction rules, (distributed) event messaging reaction rules, query reaction rules etc.) in the CEP RuleML subfamily
- Knowledge Representation Event/Action/Situation Transition/Process Logics and Calculi in the KR Reaction RuleML subfamily

The syntax of reaction rules in Reaction RuleML is defined on top of Derivation RuleML by a general rule format which can be specialized in the different Reaction RuleML subfamilies to the four different reaction rule types (and variants of these types).

```
<Rule style="active|messaging|reasoning">

    <oid>     <!-- object id of the rule -->                          </oid>
    <label>   <!- (semantic) metadata of the rule -->               </label>
    <scope>   <!- scope of the rule e.g. a rule module -->          </scope>
    <evaluation>    <!-- intended semantics -->                 </evaluation>
    <qualification> <!- e.g. qualifying rule declarations, e.g.
                         priorities, validity, strategy -->  </qualification>
    <quantification>  <!-- quantifying rule declarations,
                          e.g. variable bingings -->       </quantification>
    <on>      <!- event part -->                                    </on>
    <if>      <!- condition part -->                                </if>
    <then>    <!- (logical) conclusion part -->                   </then>
    <do>      <!--  action part -->                                 </do>
    <after>   <!- postcondition part after action,
                   e.g. to check effects of execution -->          </after>
    <else>  <!- (logical) else conclusion -->                      </else>
    <elsedo>  <!-- alternative/else action,
                   e.g. for default handling -->                 </elsedo>
</Rule>
```

The execution style of a reaction rule is defined by the optional attribute `style`.

- active: 'actively' polls/detects occurred events in ECA and CEP rules or changed conditions in production rules
- messaging: waits for incoming complex event message (inbound) and sends messages (outbound) as actions
- reasoning: logical / inference reasoning as e.g., KR formalisms such as event / action / transition logics (as e.g. in Event Calculus, Situation Calculus, temporal action languages formalizations)

The evaluation semantics (interpretation and/or execution) of reaction rules is defined in the optional role subchild `evaluation`. This can be used to define rule evaluation semantics such as weak or strong evaluation which defines the "execution lifecycle" of the rule execution.

A rule instance can be uniquely identified by an object identifier `<oid>`. The metadata `<label>` is used to annotate the rule with optional metadata. The scope `<scope>` defines a (constructive) view on the rulebase, e.g. the rule only applies to a module in the rulebase. The qualification `<qualification>` defines

an optional set of rule qualifications such as a validity value, fuzzy value or a priority value. The quantification `<quantification>` is used to define quantifiers such as the typical existential and unversal quantification; it can also be used for extensions such as variable binding patterns to restrict pattern matching in production rules or define other operator definitions.

4.1 Production RuleML

A production rule is a statement of rule programming logic that specifies the execution of one or more actions in case its conditions are satisfied, i.e. production rules react to states changes (not to explicit events). The essential syntax is `if` *Condition* `do` *Action*. Accordingly, standard production rules in the Production RuleML subfamily are written as follows (an `active-style Rule` can be shortcut to `Reaction`, which can be stripe-skipped for `if` as first child and `do` as second):

```
<Rule style="active">
  <if>...</if>
  <do>---</do>
</Rule>
```

```
<Reaction>
  <if>...</if>
  <do>---</do>
</Reaction>
```

```
<Reaction>
  ...
  ---
</Reaction>
```

Actions are `Assert` (add knowledge); `Retract` (retract knowledge); `Update` (update/modify knowledge); `Equal` (single-assign term to variable); `Execute` (execute (external) function).

Let us modify our initial example to a production rule which instead of just deriving discounts does an `Assert` of them (`Retract`/`Update` would be similar):

```
<!--  Reaction Rule 1a (Production Rule with "Condition" and "Action"):
      If premium and regular derivable do assert discount for customer -->

<Reaction>
  <if>
    <And>
      <Atom><Rel>premium</Rel><Var>cust</Var></Atom>
      <Atom><Rel>regular</Rel><Var>prod</Var></Atom>
    </And>
  </if>
  <do>
    <Assert>
      <Atom><Rel>discount</Rel><Var>cust</Var><Var>prod</Var><Data>5.0 percent</Data></Atom>
    </Assert>
  </do>
</Reaction>
```

Relationships between Production RuleML and RIF-PRD: Members of the Reaction RuleML Technical Group have co-edited the W3C RIF Production Rule Dialect (RIF-PRD). RIF-PRD with inflationary negation is a less expressive subset of PR RuleML. Syntactically, production rules in RIF-PRD are written in `if-then` syntax instead of PR RuleML's `if-do` syntax, which allows a clear semantic distinction of a conclusion (`then` part) and action (`do` part), when both are allowed for the same rule. `Do` as a type tag is used in RIF-PRD to syntactically denote a compound action which is a sequence of standard production rule actions (`Assert`, `Retract`, and `Modify`), whereas Reaction RuleML supports expressive complex action definitions using action algebra operators. Quantifying variable binding declarations are supported by RIF-PRD (declare)

and by Production RuleML (quantification), which in addition also supports rule qualifications.

Relationships between Production RuleML and OMG PRR: Based on [WATB04] and Production RuleML, members of the Reaction RuleML Technical Group have co-edited the OMG Production Rule Representation (PRR). RuleML is one of the languages whose features are to be covered by PRR on an abstract level. Since PRR is a meta-language, Production RuleML's XML syntax can be used as a concrete expression language instantiating PRR models. That is, OMG PRR provides a way to include rules into the (UML) model of an application at design time and Production RuleML then provides a standard means of translating the model and feeding the executable rules into a PR application at run time.

4.2 ECA RuleML

In contrast to production rules, Event-Condition-Action (ECA) rules define an explicit event part which is separated from the conditions and actions of the rule. Their essential syntax is `on` *Event* `if` *Condition* `do` *Action*. ECA RuleML extends Production RuleML with an explicit `<on>` event part and rich (complex) event and action constructs defined in event/action libraries (the `active Rule` is again shortcut to `Reaction`, but the `<on>`/`<if>`/`<do>` role stripes are kept):

```
<Rule style="active">
  <on>***</on>
  <if>...</if>
  <do>---</do>
</Rule>
```

```
<Reaction>
  <on>***</on>
  <if>...</if>
  <do>---</do>
</Reaction>
```

We modify our example as follows:

```
<!--  Reaction Rule 1c (ECA Rule with "Event", "Condition", and "Action"):
      On receiving premium notification from marketing and if regular derivable
      do send discount to customer -->

<Reaction>
  <on>
    <Receive>
      <from><Ind>marketing</Ind></from>
      <content>
        <Atom><Rel>premium</Rel><Var>cust</Var></Atom>
      </content>
    </Receive>
  </on>
  <if>
    <Atom><Rel>regular</Rel><Var>prod</Var></Atom>
  </if>
  <do>
    <Send>
      <to><Var>cust</Var></to>
      <content>
        <Atom><Rel>discount</Rel><Var>cust</Var><Var>prod</Var><Data>5.0 percent</Data></Atom>
      </content>
    </Send>
  </do>
</Reaction>
```

Variants of this standard ECA rule are, e.g., Event-Action triggers (EA rules) and ECAP rules (ECA rules with Postconditions after the action part).

With its typed logic, RuleML supports the (re)use of external event/action ontologies and metamodels which can be applied in the definition of semantic event/action types. For instance, the following standard library defines a set of typical event and action algebra operators:

```
Event Algebras and Action Algebras

Event Algebra:
    Sequence (Ordered), Or (Disjunction), Xor (Mutal Exclusion),
    And (Conjunction), Concurrent, Not, Any, Aperiodic, Periodic
Action Algebra:
    Succession (Ordered Succession of Actions), Choice
    (Non-Determenistic Choice), Flow (Parallel Flow),
    Loop (Iterative Loop)
```

Furthermore different selection, consumption, and (transactional) execution policies for events and actions can be specified in the complex event/action descriptions. This allows for a highly extensible and flexible Semantic CEP (SCEP) approach which (re-)uses external semantic models.

4.3 CEP RuleML

Complex Event Processing (CEP) is about the detection of complex events and reaction to complex events in near realtime. CEP rules might adopt the style of ECA rules in CEP RuleML, where the `<on>` event part might be a complex event type definition; or, they might adopt the style of CA producion rules where the complex event patterns are defined as restrictions on the variable binding definitions in the rule quantifications. However, it is also possible to represent serial messaging CEP reaction rules which `receive` and `send` events in arbitrary combinations. A serial (messaging) reaction rule starts either with a receiving event `on` – the trigger of the *global* reaction rule – or with a rule conclusion `then` – the head of the local *inline* reaction rule – followed by an arbitrary combination of conditions `if`, events `receive` and actions `send` in the body of the rule. This flexibility with support for modularization and aspect-oriented weaving of reactive rule code is in particular useful in distributed systems where event processing agents communicate and form a distributed event processing network, as e.g. in the following example:

```
<Rule style="active">
  <on><Receive> receive event from agent 1 </Receive></on>
  <do><Send> query agent 2 for regular products in a new sub-conversation </Send></do>
  <on><Receive> receive results from sub conversation with agent 2 </Receive></on>
  <if> prove some conditions, e.g. make decisions on the received data </if>
  <do><Send> reply to agent 1 by sending results received from agent 2 </Send></do>
</Rule>
```

For better modularization, the sub-conversation can be also written with an inlined reaction rule as follows:

```
<Rule style="active">
  <on><Receive> receive event from agent 1 </Receive></on>
  <if> <!- this goal activates the inlined reaction rule -- see below -->
    <Atom><Rel>regular</Rel><Var>prod</Var></Atom>
  </if>
```

```
  <do><Send> reply to agent 1 by sending results received from agent 2 </Send></do>
</Rule>

<Rule style="active">
  <then>
    <Atom><Rel>regular</Rel><Var>prod</Var></Atom>
  </then>
  <do><Send> query agent 2 for regular products in a new sub-conversation </Send></do>
  <on><Receive> receive results from sub conversation with agent 2 </Receive></on>
</Rule>
```

This messaging reaction rule can be translated e.g. into a serial messaging Horn rule and executed in the Prova rule engine.[8]

Relationships between CEP RuleML and EPTS work: RuleML is a founding member of the Event Processing Technical Society (EPTS). Members of the Reaction RuleML Technical Group are contributing to the work on an Event Processing glossary, use cases, reference architectures, and event processing language models. With its flexible and extensible approach, CEP RuleML is a highly expressive rule-based Event Processing Language (rule-based EPL) which can make use of external event and action metamodels / ontologies such as the many existing event ontologies or the planned OMG Event Model Profile. Since CEP RuleML syntactically builds on top of Production RuleML and ECA RuleML – besides flexible (messaging) reaction rules – both major rule types can be used for representing (complex) event processing rules. Moreover, CEP RuleML can adequately represent typical use cases and functionalities in Event-Driven Architectures (EDAs) and (distributed) Event Processing Network (EPN) architectures.

4.4 KR Reaction RuleML

Event/action logics, which have their origins in the area of knowledge representation (KR), focus on the inferences that can be made from the happened or planned events/actions, i.e. they define the inferences of the effects of events/actions on changeable properties of the world (situations, states). KR Reaction RuleML defines syntax and semantics for KR event/action calculi such as Situation Calculus, Event Calculus and Temporal Action Languages etc. Specifically the notion of an explicit *state* (a.k.a. as state or fluent in Event Calculus) is introduced in KR Reaction RuleML. An event/action *initiates* or *terminates* a state. That is, a state explicitly represents the abstract effect of occurred events and executed actions. Such states can be e.g. used for situation reasoning in the condition part of reaction rules.

```
<Rule style="reasoning">
  <on> <Message mode="inbound"> event message </Message> </on>
  <if> <HoldsState> state individual </HoldsState> </if>
  <do> <Message mode="outbound"> action message </Message> </do>
</Rule>
```

[8] `http://prova.ws`

5 RuleML Tools and Applications

Several tools have already been built around RuleML, including rule engines (e.g., OO jDREW[9], Prova[10]), rule editors (e.g., Acumen Business Rule Manager[11], Syntactic-Semantic RuleML Editor (S2REd)[12]), as well as translators such as the Reaction RuleML translator (Web) service framework[13]. Most of these tools contribute to interoperability by making use of translators between presentation syntaxes such as Pure Prolog (or extensions such as POSL[14] and Prova) and RuleML/XML as well as between RuleML/XML and other XML-based languages such as RIF/XML. RIF RuleML interoperation was started with a common subset [Bol09].

RuleML-based multi-agent architectures for distributed rule inference services include Rule Responder[15] [PBKC07] and Emerald[16]. Rule Responder extends the Semantic Web towards a Pragmatic Web infrastructure for collaborative rule-based agent networks implemented as distributed rule inference services, where agents engage in conversations by exchanging messages and cooperate to achieve (collaborative) goals. Rule Responder utilizes messaging reaction rules from Reaction RuleML for communication between the distributed agent inference services. The Rule Responder middleware is based on Enterprise Service Bus (ESB) and Semantic Web technologies for implementing intelligent agent services that access data and ontologies, receive and detect events (e.g. for complex event processing in event processing agent networks), and make rule-based inferences and autonomous pro-active decisions for reactions based on these representations. Rule Responder has become the infrastructure for several Web 3.0 applications (e.g., PatientSupporter[17]).

6 Conclusion

RuleML 1.0 is the unifying family of languages spanning across all industrially relevant kinds of Web rules. It accommodates and extends other languages including W3C RIF. Yet, as shown by this paper, the major RuleML constructs are easy to learn. FOL RuleML deliberation rules could be regarded as an instantiation of the RIF Framework for Logic Dialects. However, for RIF-PRD and Production RuleML no corresponding RIF Framework for Production Rule Dialects exists, and for Reaction RuleML even a RIF instance dialect, RRD, is only envisioned yet, although the ongoing RIF RuleML collaboration should sustain progress here. On the other hand, Modal RuleML deliberation rules could be further developed

[9] http://www.jdrew.org/oojdrew
[10] http://www.prova.ws
[11] http://www.acumenbusiness.com
[12] http://sourceforge.net/projects/s2red
[13] http://reaction.ruleml.org/translation.htm
[14] http://ruleml.org/submission/ruleml-shortation.html
[15] http://responder.ruleml.org
[16] http://lpis.csd.auth.gr/systems/emerald/emerald.html
[17] http://ruleml.org/PatientSupporter

in collaboration with corresponding Common Logic extensions, as also needed for Semantics of Business Vocabulary and Business Rules (SBVR).

Object Oriented RuleML's slotted facts and rules can be used to define cases and associated solutions in Case Based Reasoning (CBR). With its optional use of types, which also accommodate finite domains, RuleML is well-prepared for a Constraint Logic Programming (CLP) extension. A related Constraint Handling Rules (CHR) extension could follow next.

Translators between sublanguages of RuleML, RIF, PRR, SBVR, Jess, Prova (ISO Prolog) have been written and further ones are under development. RuleML 1.0 as the overarching specification of Web rules will thus help to unify and drive the development of Web-based rule interoperation.

References

[BBB+05] Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., Tabet, S.: Semantic Web Services Language (SWSL). Release Version 1.0 (May 2005), http://www.daml.org/services/swsf/1.0/swsl/

[BK10a] Boley, H., Kifer, M.: A Guide to the Basic Logic Dialect for Rule Interchange on the Web. IEEE Transactions on Knowledge and Data Engineering (2010) (forthcoming)

[BK10b] Boley, H., Kifer, M.: RIF Framework for Logic Dialects, W3C Recommendation (June 2010), http://www.w3.org/TR/rif-fld

[Bol09] Boley, H.: RIF RuleML Rosetta Ring: Round-Tripping the Dlex Subset of Datalog RuleML and RIF-Core. In: Governatori, G., Hall, J., Paschke, A. (eds.) RuleML 2009. LNCS, vol. 5858, pp. 29–42. Springer, Heidelberg (2009)

[CGT89] Ceri, S., Gottlob, G., Tanca, L.: What You Always Wanted to Know About Datalog (And Never Dared to Ask). IEEE Trans. on Knowledge and Data Eng. 1(1) (March 1989)

[DPSS08] Damásio, C.V., Pan, J.Z., Stoilos, G., Straccia, U.: Representing Uncertainty in RuleML. Fundam. Inf. 82(3), 265–288 (2008)

[End01] Enderton, H.B.: A Mathematical Introduction To Logic, 2nd edn. Harcourt/Academic Press, San Diego (2001)

[GDK09] Grosof, B., Dean, M., Kifer, M.: The SILK System: Scalable Higher-Order Defeasible Rules. In: Governatori, G., Hall, J., Paschke, A. (eds.) RuleML 2009. LNCS, vol. 5858. Springer, Heidelberg (2009)

[HK09] Heymans, S., Kifer, M.: RIF Core Answer Set Programming Dialect. W3C RuleML Specification (December 2009), http://ruleml.org/rif/RIF-CASPD.html/

[HPSB+04] Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: Semantic Web Rule Language (SWRL). W3C Member Submission (May 2004), http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/

[KBA08] Kontopoulos, E., Bassiliades, N., Antoniou, G.: Deploying Defeasible Logic Rule Bases for the Semantic Web. Data Knowl. Eng. 66(1), 116–146 (2008)

[Mak87] Makowsky, J.A.: Why Horn formulas matter in computer science: Initial structures and generic examples. Journal of Computer and System Sciences 34, 266–292 (1987)

[Nie07] Nieuwenhuis, R.: A survey of some recent trends in rewrite-based and paramodulation-based deduction (2007)

[PB09a] Paschke, A., Boley, H.: Rule Markup Languages and Semantic Web Rule Languages. In: Giurca, A., Gasevic, D., Taveter, K. (eds.) Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, pp. 1–24. IGI Publishing (May 2009)

[PB09b] Paschke, A., Boley, H.: Rules Capturing Events and Reactivity. In: Giurca, A., Gasevic, D., Taveter, K. (eds.) Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, pp. 215–252. IGI Publishing (May 2009)

[PBK10] Polleres, A., Boley, H., Kifer, M.: RIF Datatypes and Built-ins 1.0, W3C Recommendation (June 2010), `http://www.w3.org/TR/rif-dtb`

[PBKC07] Paschke, A., Boley, H., Kozlenkov, A., Craig, B.: Rule Responder: RuleML-Based Agents for Distributed Collaboration on the Pragmatic Web. In: 2nd ACM Pragmatic Web Conference 2007. ACM, New York (2007)

[WATB04] Wagner, G., Antoniou, G., Tabet, S., Boley, H.: The Abstract Syntax of RuleML – Towards a General Web Rule Language Framework. In: Web Intelligence, pp. 628–631. IEEE Computer Society Press, Los Alamitos (2004)

[Wid93] Widom, J.: Deductive and Active Databases: Two Paradigms or Ends of a Spectrum? In: Rules in Database Systems, pp. 306–315 (1993)

A XSLTs and XSDs for RuleML 1.0

The specification of RuleML 1.0 differs from the RuleML 0.91 specification by putting more emphasis on XSLT, besides XML Schema: XSLT translators normalize RuleML 1.0 serializations, so XML Schema Definitions (XSDs) need to validate normal forms only. Normal forms provide an abstract-syntax level, where the equality of arbitrary RuleML 1.0 abstract syntaxes is reduced to the identity of their normal forms. They also simplify the XSDs, e.g., avoiding the permutation of role children. For instance, the normal form for derivation rules uses explicit role tags for `<if>` and `<then>` in that order, as shown by the middle-column version of Implication Rule 1 in Section 3.1.

The XSDs of RuleML 1.0 change those of RuleML 0.91 as follows: Type tags `Hterm` and `Con` are replaced with `Uniterm` and `Const`, respectively. Role tags `body` and `head` are replaced with `if` and `then`, respectively. Role tags `lhs` and `rhs`, with `left` and `right`, respectively. Attribute `in="no|semi|yes|effect|modal"` and respective values are replaced with `per="copy|open|value|effect|modal"`. Attribute `uri` becomes `iri`. The online RuleML 1.0 specification is based on *normalidation*, including XSLTs for *normali*zation and XSDs for subsequent vali*dation*.[18] The specification is illustrated by test cases grouped according to sublanguages.[19]

[18] `http://ruleml.org/1.0` (`http://ruleml.org/1.0/xslt` and `http://ruleml.org/1.0/xsd`)

[19] `http://ruleml.org/1.0/exa`