RuleML 1.0

The Overarching Specification of Web Rules

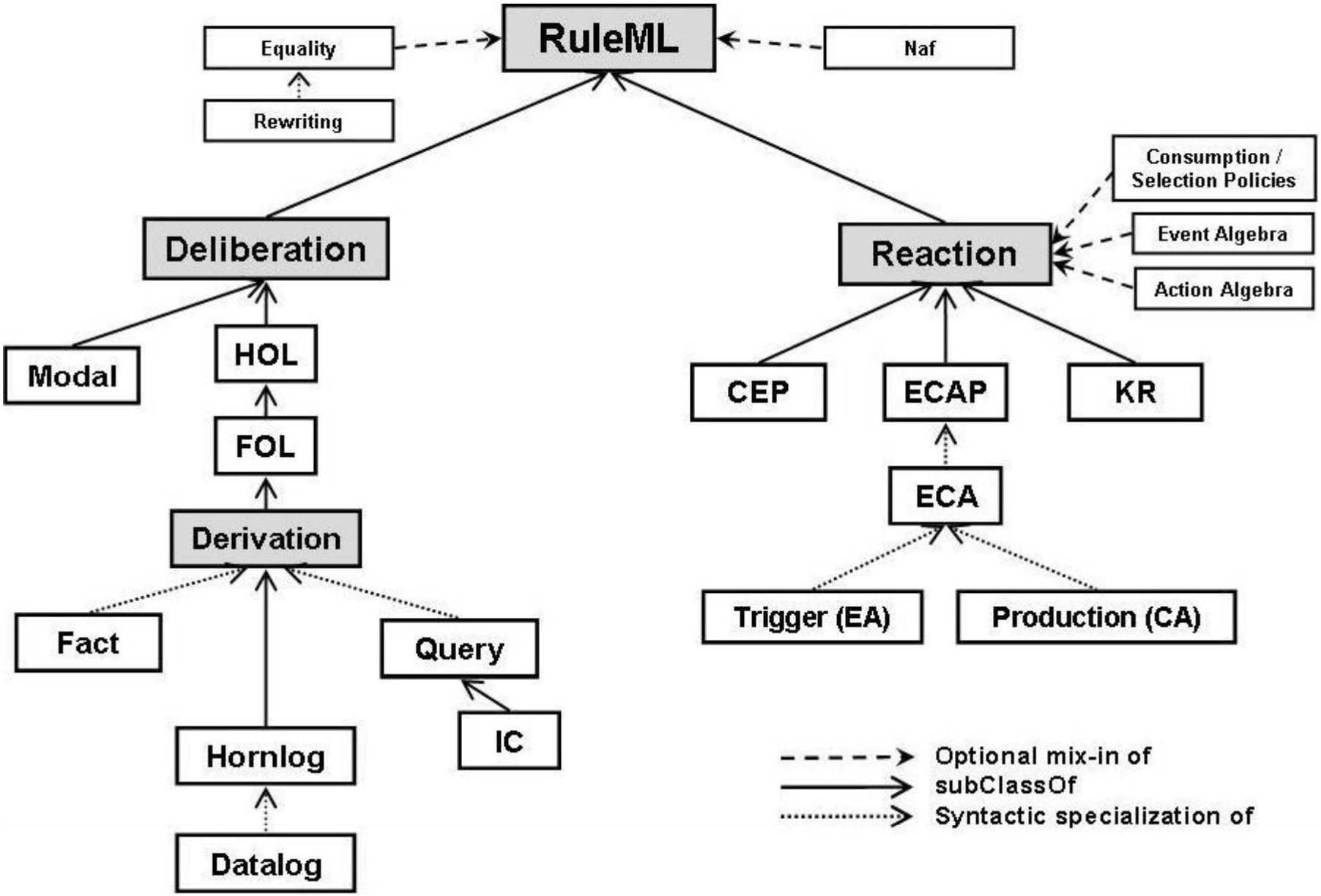
Harold Boley, Adrian Paschke, and Omair Shafiq
NRC, FUB, and UoC

Introduction

- Web Rules use various formats and packaging
- But semantics are often compatible
- Rulebases can then be reused with an interchange technology consisting of
  - a family of canonical rule languages
  - bi-directional translators between canonical languages and the languages to be interchanged

The RuleML Family Revisited (I)

- Taxonomy of subfamilies, languages, and sublanguages classified through
  - syntactic power of rules, as reflected by XML Schema Definitions (XSDs)
  - semantic power, as reflected by model-theoretic, proof-theoretic, and operational semantics
- Often, more syntactic power leads to more semantic power (e.g., introduction of `Expr`ession syntax pushes Datalog to Horn Logic (Hornlog) models)
- Syntactically neutral aspects of semantic power expressed by semantic attributes (e.g., `negation` attribute for semantics of Negation-`a`s-`f`ailure)


RuleML
Equality
Rewriting
Naf
Deliberation
Reaction
Consumption / Selection Policies
Event Algebra
Action Algebra
Modal
HOL
FOL
Derivation
Fact
Query
IC
Hornlog
Datalog
CEP
ECAP
KR
ECA
Trigger (EA)
Production (CA)
Optional mix-in of
subClassOf
Syntactic specialization of

The RuleML Family Revisited (II)

- Diagram shows 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)

The RuleML Family Revisited (III)

- 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
  - Event-less *Production (CA)* rules

The RuleML Family Revisited (IV)

- 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.

The RuleML Family Revisited (V)

- Reaction RuleML syntactically extends 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
- Reaction RuleML is based on ‘pluggable’ ontologies (e.g., algebras) of (complex) actions, events, and – in the KR subfamily – situations

The RuleML Family Revisited (VI)

- Production RuleML defines condition-action rules
- Complex Event Processing (CEP) RuleML defines (complex) events and their efficient processing
- Reaction RuleML extends production rules with event-triggering part, syntactically defining ECA rules, and with further semantic extensions, e.g. for CEP rules

The RuleML Family Revisited (VII)

- RuleML rules 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>`

RuleML and W3C RIF (I)

- RuleML provided input to RIF on several levels
  - Use of ‘striped’ XML
  - Structuring of rule classes into family of sublanguages
- Partial mappings between, e.g.,
  - Datalog RuleML and RIF-Core
  - Derivation RuleML and RIF Basic Logic Dialect (RIF-BLD)
  - Production RuleML and RIF Production Rule Dialect (RIF-PRD)

RuleML and W3C RIF (II)

- RIF WG has terminated end of September 2010 until uncertain revival for a possible RIF 2
- RIF's standard logic Web rule dialects Core and BLD come with rigorous model-theoretic semantics for cascaded design choices
- However, W3C Core and BLD Recommendations cover only fraction of Web rule space and their very rigor gives existing Web rule languages little room for RIF conformance
- The RuleML Initiative has thus been co-hosting development of
  - further (non-standard extensions or) RIF dialects such as Core Answer Set Programming Dialect (RIF-CASPD), using flexibility-enhancing Framework for Logic Dialects (RIF-FLD)
  - RIF RuleML sublanguages such as Datalog with equality plus externals (Dlex) and envisioned Reaction Rule Dialect (RRD)

RuleML Design Rationale: Syntax

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

RuleML Design Rationale: Semantics

- RuleML, as 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
- E.g., 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
- This flexible semantics approach of RuleML allows refining the semantics of a syntactically represented rulebase

Deliberation Rules: Datalog RuleML

Running example: Ternary `Relation` `discount` conditional on unary `premium` and `regular`

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

```
<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>
```

Example: Turned Around & Stripe-Skipped

```
<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>
```

Example: Slotted Variant

Uses pairs *key* → *term* in conclusion's 3-ary relation, represented as metaroles `<slot>`*key term*`</slot>`

```
<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>
```

Example: Typed Variant

Uses `Variables` with attribute `type`, whose values are IRIs pointing to ontological class definitions on the Web specified 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/…/ProfileHierarchy.owl#Product">prod</Var>
      <Data>5.0 percent</Data>
    </Atom>
  </then>
  <if> . . . </if>
</Implies>
```

Hornlog RuleML (I)

Extension of Datalog RuleML, mainly its `Atoms`: Allows `Functional` `Expressions` as terms in `Atoms` and in other `Exprs`. Can be uninterpreted, using attribute `per` with filler `"copy"` or interpreted, using it with filler `"value"`

Refine initial example introducing uninterpreted `Expr` representing the term `percent[5.0]`

Hornlog RuleML (II)

```
<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>
```

First Order Logic (FOL) RuleML

Extension of Hornlog RuleML mainly adding classical negation and (explicit) quantifiers

"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."

```
<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>
```

$$Xor(A,B) \leftrightarrow And(Or(A,B),Not(And(A,B)))$$

RuleML with Equality

Equality formulas act as extension to sublanguages such as Datalog RuleML, Hornlog RuleML, and FOL RuleML. Equal has oriented attribute with value "no" default

```
<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>
```

Naf RuleML

FOL: Strong Negation. Here: Negation-as-failure (as in LP)
Distinguishes Answer Set (incl. stable model) semantics and Well-Founded semantics, using semantic attribute, negation, on the enclosing Rulebase, with default AS

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

Reaction Rules: Four Subfamilies

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

Reaction Rules: Specializable Syntax

```
<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>
```

Reaction RuleML – Example Rule Types

- **Production Rule:**

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

- **Trigger Rule:**

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

- **ECA Rule:**

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

Example: Messages

```
...
<Message mode="outbound" directive="ACL:query-ref">
    <oid> <Ind>RuleML-2008</Ind> </oid>
    <protocol> <Ind>esb</Ind> </protocol>
    <sender> <Ind>User</Ind> </sender>
    <content>
      <Atom>
            <Rel>getContact</Rel>
            <Ind>Sponsoring</Ind>
            <Var>Contact</Var>
      </Atom>
    </content>
</Message>
...
```

FIPA ACL directive

- Event Message is local to the conversation state (oid) and pragmatic context (directive)

Complex Event / Action Algebra Operators

- Action Algebra:

  Succession (Ordered Succession of Actions), *Choice* (Non-Determenistic Choice), *Flow* (Parallel Flow), *Loop* (Loops)

- Event Algebra:

  Sequence (Ordered), *Disjunction* (Or) , *Xor* (Mutal Exclusive), *Conjunction* (And), *Concurrent* , *Not*, *Any*, *Aperiodic*, *Periodic*, *AtLeast*, *ATMost*

- Time and Event Interval Algebra

  During, Overlaps, Starts, Precedes, Meets, *Equals*, *Finishes*

Execution Semantics

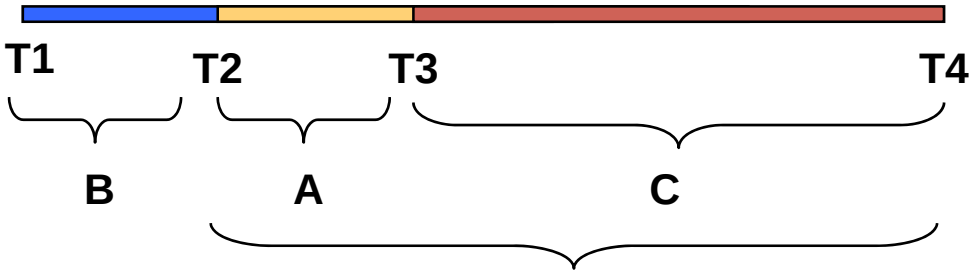
1. Definition
   - Definition of event/action pattern e.g. by event algebra
   - Based on declarative formalization or procedural implementation
   - Defined over an atomic instant or an interval of time, events/actions, situation, transition etc.
2. Selection
   - Defines selection function to select one event from several occurred events (stored in an event instance sequence e.g. in memory, database/KB) of a particular type, e.g. "*first*", "*last*"
   - Crucial for the outcome of a reaction rule, since the events may contain different (context) information, e.g. different message payloads or sensing information
3. Consumption
   - Defines which events are consumed after the detection of a complex event
   - An event may contribute to the detection of several complex events, if it is not consumed
   - Distinction in event messaging between "multiple receive" and "single receive"
   - Events which can no longer contribute, e.g. are outdated, should be removed
4. Execution
   - Actions might have an internal effect i.e. change the knowledge state leading to state transition from (pre)-condition state to post-condition state.
   - The effect might be hypothetical (e.g. a hypothetical state via a computation) or persistent (update of the knowledge base),
   - Actions might have an external side effect

Event & Action Semantics

- Complex Events Semantics based on KR event logics
  - Interval-based KR Event Calculus semantics (model-theory + proof theory) based on time intervals modeled as fluents

$$I : \overline{Ti} \, x \, \overline{Fl} \mapsto \{true, false\}.$$

  - Example: *B;(A;C) (Sequence)*



- Update Actions
  - Updates: Add / Remove / Change extensional and intensional knowledge
  - Transition to new knowledge state: $P' = P \cup U^{pos}_{oid}$ or $P' = P \setminus U^{neg}_{oid}$
  - Transition Paths: $< P, E, U > \rightarrow < P', U, U' > \rightarrow < P'', U', U'' > \rightarrow .. \rightarrow < P^{n+1}, U^{n}, A >$
  - Transactional Logic Semantics

Selected Reaction RuleML Features

- Action Algebra:
  Succession (Ordered Sequence), Choice (Non-Deterministic Selection),
  Flow (Parallel Concurrent Flow), *Loop* (Iteration)
- Event Algebra:
  Sequence (Ordered), *Disjunction* (Or) , *Xor* (Mutal Exclusion),
  Conjunction (And), *Concurrent* , *Not*, *Any*, *Aperiodic*, *Periodic*
- Event / action messaging
- External data models and ontologies
- Different detection, selection and consumption policies
- Intervals (Time, Event)
- Situations (States, Fluents)
- External event query languages
- ...

Conclusion

- RuleML 1.0 is unifying family of languages across all industrially relevant Web rules
- Translators between sublanguages of RuleML, RIF, PRR, SBVR, Jess, Prova (ISO Prolog) have been written and further ones are under development
- Modal RuleML could be further developed in collaboration with corresponding Common Logic extensions, as also needed for SBVR