The RuleML Family of Web Rule Languages

PPSWR’06, Budva, Montenegro, 10 June 2006

Harold Boley
University of New Brunswick, Canada
National Research Council of Canada
Introduction

- Rules are central to the Semantic Web
- Rule interchange in an open format is important for e-Business
- RuleML is the de facto open language standard for rule interchange/markup
- Collaborating with W3C (RIF), OMG (PRR, SBVR), OASIS, DARPA-DAML, EU-REWERSE, and other standards/gov't bodies
RuleML Enables ...

Rule

modelling
markup
translation
interchange
execution
publication
archiving

in

UML
RDF
XML
ASCII
RuleML Identifies ...

- Expressive sublanguages
  - for Web rules
  - started with
    - *Derivation* rules: extend SQL views
    - *Reaction* rules: extend SQL triggers
  - to empower their subcommunities
RuleML Specifies ...

- Derivation rules via XML Schema:
  - All sublanguages: (OO) RuleML 0.9
  - First Order Logic: FOL RuleML 0.9
  - With Ontology language: SWRL 0.7
    - A Semantic Web Rule Language Combining OWL (W3C) and RuleML
  - With Web Services language: SWSL 0.9

- Translators in & out (e.g. Jess) via XSLT
Modular Schemas

“RuleML is a **family** of sublanguages whose **root** allows access to the language as a whole and whose **members** allow to identify customized subsets of the language.”

- RuleML: Rule Markup Language
  - RuleML derivation rules (shown here) and production rules defined in XML Schema Definition (XSD)
  - Each XSD of the family corresponds to the expressive class of a specific RuleML sublanguage

- The most recent schema specification of RuleML is always available at [http://www.ruleml.org/spec](http://www.ruleml.org/spec)
- Current release: RuleML 0.9
- Pre-release: RuleML 0.91
Schema Modularization

- XSD URIs identify expressive classes
  - Receivers of a rulebase can validate applicability of tools
    (such as Datalog vs. Hornlog interpreters)
  - Associated with semantic classes
    (such as function-free vs. function-containing Herbrand models)
- Modularization (Official Model)
  - Aggregation:
    e.g., Datalog part of Hornlog
  - Generalization:
    e.g., Bindatalog is a Datalog
- **Rectangles** are sublanguages
  - Inheritance between schemas
- **Ovals** are auxiliary modules
  - Elementary, including only element and/or attribute definitions
  - Become *part of* sublanguages

E.g., in [http://www.ruleml.org/0.9/xsd/hornlog.xsd](http://www.ruleml.org/0.9/xsd/hornlog.xsd)

```xml
<xs:redefine
    schemaLocation= "datalog.xsd">
<xs:include
    schemaLocation= "modules/cterm_module.xsd"/>
```
Bring Datalog to the Semantic Web

- Start with n-ary relations (not binary properties)
- Keep variable typing optional (reuse RDFS’ subClassOf taxonomies as sort lattices)
- Allow signature declarations of arities and types
- Employ function-free facts as well as Horn rules (rather than 1st: RDF descriptions; 2nd: RDF rules)
- Use function-free Herbrand model semantics (querying stays decidable)
- Provide three syntactic levels:
  - User-oriented: Prolog-like, but with “?”-variables
  - Abstract: MOF/UML diagrams
  - XML serialization: Datalog RuleML
"The **discount** for a **customer** buying a **product** is 5 percent if the **customer** is **premium** and the **product** is **regular**."
Extend Datalog for the Semantic Web (I)

- Allow slots as name→filler pairs in atoms (cf. F-logic’s methods and RDF’s properties)
- Extend optional types and signatures for slots
- Add optional object identifiers (oids) to atoms
- Separate Data literals from Individual constants
"The **discount** for a *customer* buying a *product* is 5 percent if the *customer* is **premium** and the *product* is **regular**."
Extend Datalog for the Semantic Web (II)

- Permit IRI webizing for **Data** (XML Schema Part 2), **Individuals** (RDF’s **resources**), **Relations**, **slot** names, types (RDFS’ classes), and **oids** (RDF’s **about**)

- Introduce **Module** (scope) construct for clauses (cf. RDF’s named graphs)

- Add scoped-default (**Naf**), strong (**Neg**), scoped-default-of-strong negation (unscoped: cf. **ERDF**)

- Integrate with Description Logics
  - Homogeneous (SWRL, Datalog RuleML + OWL-DL)
  - Hybrid (AL-log, Datalog\textsuperscript{DL}, DL+log, …)
Bring Horn Logic to the Semantic Web

- Augment Datalog with uninterpreted Functions and their Expressions; also for extended Datalog
- Augment Datalog’s Herbrand model semantics with such Functions (querying becomes undecidable)
- Extend Datalog syntaxes
  - XML Schema of Hornlog RuleML inherits and augments XML Schema of Datalog RuleML
- Add Equality and interpreted Functions (XML serialization: attribute in="yes")
- Reuse XQuery/XPath functions and operators as built-ins
Specify a First-Order Logic Web Language

- Layer on top of either
  - Disjunctive Datalog: Or in the head generalizing Datalog
  - Disjunctive Horn Logic: Or in head of near-Horn clauses

- Alternatively, layer on top of either
  - Disjunctive Datalog with restricted strong Negation
  - Disjunctive Horn Logic with restricted strong Neg

- Permit unrestricted Or, And, strong Neg, and quantifiers Forall and Exists to obtain FOL

- Use semantics of classical FOL model theory

- Extend Hornlog RuleML syntax to FOL RuleML
Equality for Functions

- Functional programming (FP) plays increasing Web role: MathML, XSLT, XQuery
- Functional RuleML employs orthogonal notions freely combinable with Relational RuleML
- Also solves a Relational RuleML issue, where the following ‘child-of-parent’ elements are separated:
  - Constructor (Ctor) of a complex term (Cterm)
  - User-defined function (Fun) of a call (Nano)
- Proceed to a logic with equality
Function Interpretedness (I)

- Different notions of ‘function’ in LP and FP:
  - **LP:** *Uninterpreted functions* denote unspecified values when applied to arguments, not using function definitions
  - **FP:** *Interpreted functions* compute specified returned values when applied to arguments, using function definitions

- E.g.:  **first-born**: Man $\times$ Woman $\rightarrow$ Human
  - Uninterpreted: \texttt{first-born(John, Mary)} denotes first-born
  - Interpreted: using \texttt{first-born(John, Mary) = Jory}, so the application returns \texttt{Jory}
Function Interpretedness (II)

- Uninterpreted <Ctor> vs. interpreted <Fun> functions now distinguished with attribute values: <Fun in="no"> vs. <Fun in="yes">
- Function applications with Cterm vs. Nano then uniformly become Expressions
- Two versions of example marked up as follows (where "u" stands for "no" or "yes"):
  
  <Expr>
  <Fun in="u">first-born</Fun>
  <Ind>John</Ind>
  <Ind>Mary</Ind>
  </Expr>
Unconditional Equations

- Modified `<Equal>` element permits both symmetric and oriented equations
- E.g.: `first-born(John, Mary) = Jory` can now be marked up thus:

```xml
<Equal oriented="yes">
  <lhs>
    <Expr>
      <Fun in="yes">first-born</Fun><Ind>John</Ind><Ind>Mary</Ind>
    </Expr>
  </lhs>
  <rhs>
    <Ind>Jory</Ind>
  </rhs>
</Equal>
```
Conditional Equations

- Use `<Equal>` as the conclusion of an `<Implies>`, whose condition may employ other equations
- E.g.: \(?B = \text{birth-year}(?P) \Rightarrow \text{age}(?P) = \text{subtract}(\text{this-year}(), ?B)`

```xml
<Implies>
  <Equal oriented="no">
    <Var>B</Var>
    <Expr>
      <Fun in="yes">birth-year</Fun>
      <Var>P</Var>
    </Expr>
  </Equal>
  <Equal oriented="yes">
    <Expr>
      <Fun in="yes">age</Fun>
      <Var>P</Var>
    </Expr>
    <Expr>
      <Fun in="yes">subtract</Fun>
      <Expr>
        <Fun in="yes">this-year</Fun>
        <Var>B</Var>
      </Expr>
    </Expr>
  </Equal>
</Implies>
```
Accommodate SWSL-Rules

- **HiLog**: Higher-order *Variables*, *Constants*, and *Hterms* (complex terms and atomic formulas at the same time)
- **Equal**: As in Horn Logic with (unoriented) *Equality*
- **Frames**:
  - Value molecules: *Atoms* with an *oid*, an optional *Rel* class, and zero or more name->filler instance *slots*
  - Signature molecules: name=>filler class *slots*, which can have \{min:max\} cardinality constraints
- **Reification**: A formula (e.g., a rule) embedded in a *Reify* element is treated (e.g., unified) as a term
- **Skolems**: Unnamed, represent new individual constants (like RDF's blank nodes); otherwise, uniquely named ones
HiLog Examples: Hterms (I)

- First-order terms: $f(a, ?X)$

  <Hterm>
  <op><Con>f</Con></op>
  <Con>a</Con>
  <Var>X</Var>
  </Hterm>

- Variables over function symbols: $?X(a, ?Y)$

  <Hterm>
  <op><Var>X</Var></op>
  <Con>a</Con>
  <Var>Y</Var>
  </Hterm>
HiLog Examples: Hterms (II)

- Parameterized function symbols: $f(\mathbf{?X,a}) (\mathbf{b, ?X(c)})$

```xml
<Hterm>
  <op>
    <Hterm>
      <op><Con>f</Con></op>
      <Var>X</Var>
      <Con>a</Con>
    </Hterm>
    <Con>b</Con>
  </op>
  <Hterm>
    <op><Var>X</Var></op>
    <Con>c</Con>
  </Hterm>
</Hterm>
```
Equality Example

- **Equality**: in rule head: \( f(a, ?X) := g(?Y, b) :- p(?X, ?Y) \).

```xml
<Implies>
  <head>
    <Equal>
      <Hterm>
        <op><Con>f</Con></op>
        <Con>a</Con>
        <Var>X</Var>
      </Hterm>
      <Hterm>
        <op><Con>g</Con></op>
        <Var>Y</Var>
        <Con>b</Con>
      </Hterm>
    </Equal>
  </head>
  <body>
    <Hterm>
      <op><Con>p</Con></op>
      <Var>X</Var>
      <Var>Y</Var>
    </Hterm>
  </body>
</Implies>
```
Frame Example: Value Molecule

- Parameterized-name->filler slot: o[f(a,b) -> 3]

```xml
<Atom>
  <oid><Con>o</Con></oid>
  <slot>
    <Hterm>
      <op><Con>f</Con></op>
      <Con>a</Con>
      <Con>b</Con>
    </Hterm>
    <Con>3</Con>
  </slot>
</Atom>
```
Reification Example: Reified Rule

- Rule as slot filler: \( john[\text{believes} \rightarrow \{p(?X) \text{ implies } q(?X)\}] \).

```xml
<hterm>
  <oid>john</oid>
  <slot>
    <con>believes</con>
    <reify>
      <implies>
        <body>
          <hterm>
            <op><con>p</con></op>
            <var>X</var>
          </hterm>
        </body>
        <head>
          <hterm>
            <op><con>q</con></op>
            <var>X</var>
          </hterm>
        </head>
      </implies>
    </reify>
  </slot>
</hterm>
```
Skolem Examples (I):

- Named Skolem: holds(a, #1) and between(1, #1, 5).

  <And>
  <Hterm>
    <op><Con>holds</Con></op>
    <Con>a</Con>
    <Skolem>1</Skolem>
  </Hterm>
  <Hterm>
    <op><Con>between</Con></op>
    <Con>1</Con>
    <Skolem>1</Skolem>
    <Con>5</Con>
  </Hterm>
  </And>
Skolem Examples (II):

- **Unnamed Skolem**: holds(a, _) and between(1, _, 5).

\[
\begin{align*}
\text{And} & \quad \text{Hterm} \\
& \quad \text{op} \text{holds} \text{Con} \text{a} \text{Skolem} \\
& \quad \text{Hterm} \\
& \quad \text{op} \text{between} \text{Con} 1 \text{Skolem} \text{Con} 5
\end{align*}
\]
Proceed to Modal Logics

- Modal operators **generically** represented as \textit{Relations} at least one of whose arguments is a proposition represented as an \textit{Atom} with an uninterpreted \textit{Relation} (including another modal operator)
  - \textit{Alethic} \textbf{necessary} (□) and \textbf{possible} (♢)
  - \textit{Deontic} \textbf{must} and \textbf{may} (e.g., in business rules)
  - Open for \textit{temporal} (e.g., when planning/diagnosing reactive rules), \textit{epistemic} (e.g., in authentication rules), and further modal operators
- Towards a **unified framework** for multi-modal logic based on Kripke-style possible worlds semantics
Modal Examples: Alethic Operator

- Necessity: □ \texttt{prime}(1)

\[
\begin{align*}
\text{Atom} & \quad \text{Rel modal="yes">necessary</Rel>}
\text{Atom} & \quad \text{Rel in="no">prime</Rel>}
\text{Data} & \quad 1
\end{align*}
\]
Modal Examples: Epistemic Operator

- Knowledge: $\text{knows(Mary, material(moon, rock))}$

<Atom>
  <Rel modal="yes">knows</Rel>
  <Ind>Mary</Ind>
  <Atom>
    <Rel in="no">material</Rel>
    <Ind>moon</Ind>
    <Ind>rock</Ind>
  </Atom>
</Atom>
Modal Examples: Epistemic Reasoning

- Veridicality axiom: \( \text{Knows}_A \text{proposition} \rightarrow \text{proposition} \)

\[ \text{Knows}_M \text{material}(\text{moon}, \text{rock}) \rightarrow \text{material}(\text{moon}, \text{rock}) \]

Serialization in previous slide

\[ <\text{Atom}> <\text{Rel in="yes"}>\text{material}</\text{Rel}> <\text{Ind}>\text{moon}</\text{Ind}> <\text{Ind}>\text{rock}</\text{Ind}> </\text{Atom}> \]
Modal Examples: Nested Operators

- Knowledge of Necessity: \( \text{knows}(\text{Mary}, \Box \text{prime}(1)) \)

\[
\begin{align*}
\text{<Atom>} \\
\text{  <Rel modal="yes">knows</Rel>} \\
\text{  <Ind>Mary</Ind>} \\
\text{  <Atom>} \\
\text{    <Rel modal="yes" in="no">necessary</Rel>} \\
\text{    <Atom>} \\
\text{      <Rel in="no">prime</Rel>} \\
\text{      <Data>1</Data>} \\
\text{    </Atom>} \\
\text{  </Atom>} \\
\text{</Atom>}
\]
Protect Knowledge Bases by Integrity Constraints

- A knowledge base KB is a formula in any of our logic languages.
- An integrity constraint IC is also a formula in any of our logic languages, which may be chosen independently from KB.
- KB obeys IC iff KB entails IC (Reiter 1984, 1987).
  - Entailment notion of 1987 uses epistemic modal operator.
- Serialization: `<Protect> IC KB </Protect>`
Integrity Constraint Example: Rule with $\exists$-Head

- Adapted from (Reiter 1987):
  \[ IC = \{ (\forall x) \text{emp}(x) \Rightarrow (\exists y) \text{ssn}(x,y) \} \]
  
  \[ KB_1 = \{ \text{emp}(\text{Mary}) \} \quad \text{KB}_1 \text{ violates IC} \]
  
  \[ KB_2 = \{ \text{emp}(\text{Mary}), \text{ssn}(\text{Mary,1223}) \} \quad \text{KB}_2 \text{ obeys IC} \]

  <Protect> IC KB_i </Protect>

<table>
<thead>
<tr>
<th>IC:</th>
<th>KB_i:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\forall x) \text{emp}(x) \Rightarrow (\exists y) \text{ssn}(x,y)</td>
<td>&lt;Atom&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Rel&gt;emp&lt;/Rel&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Ind&gt;Mary&lt;/Ind&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/Atom&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;And&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Atom&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Rel&gt;emp&lt;/Rel&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Ind&gt;Mary&lt;/Ind&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/Atom&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Atom&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Rel&gt;ssn&lt;/Rel&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Ind&gt;Mary&lt;/Ind&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Data&gt;1223&lt;/Data&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/Atom&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/And&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/Atom&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/Forall&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/Exists&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/Implies&gt;</td>
</tr>
</tbody>
</table>
Approach Production and Reaction Rules

- Share Condition (C) part with earlier languages as proposed for the **RIF Condition Language**
- Develop Action (A) part of Production Rules via a taxonomy of actions on KBs (Assert, Retract, ...), on local or remote hosts, or on the surroundings
- Develop Event (E) part of Reaction Rules via a corresponding taxonomy
- Create CA and ECA families bottom-up and map to relevant languages for Semantic Web Services
- Serialization: `<React> E C A </React>`
RDF Rules

- RDF-like Rules: Important RuleML sublanguage
  - Datalog: Relational databases augmented by views
  - RDF Properties: Slots permit non-positional, keyed arguments
  - RDF URIs: Anchors provide object identity via webzing through URIs
    - oids: Can be Individuals, Variables, etc.
    - uris: Now used for both RDF’s about and resource
  - RDF Blank Nodes: F-logic/Flora-2 Skolem-constant approach
    - E.g., Skolem generator ‘_’ becomes <Skolem/>
“For a product whose price is greater than 200 and whose weight is less than 50, no shipping is billed.”
Bidirectional Interpreters in Java

- Two varieties of reasoning engines
  - **Top-Down**: backward chaining
  - **Bottom-Up**: forward chaining

- **jDREW**: *Java Deductive Reasoning Engine for the Web* includes both TD and BU
  [http://www.jdrew.org](http://www.jdrew.org)

- **OO jDREW**: *Object-Oriented* extension to jDREW
  [http://www.jdrew.org/oojdrew](http://www.jdrew.org/oojdrew)

- Java Web Start online demo available at
  [http://www.jdrew.org/oojdrew/demo.html](http://www.jdrew.org/oojdrew/demo.html)
OO jDREW Slots

- Normalized atoms and complex terms
  - **oids** (object identifier)
  - **Positional** parameters (in their original order)
  - **Positional** rest terms
  - **Slotted** parameters (in the order encountered)
  - **Slotted** rest terms

- Efficient unification algorithm
  - Linear $O(m+n)$: instead of $O(m*n)$
    - No need for positional order
    - Slots internally sorted
  - Steps:
    - Scan two lists of parameters
      - Matching up roles and positions for positional parameters
      - Unifying those parameters
    - Add unmatched roles to list of rest terms
    - Generate dynamically a Plex (RuleML’s closest equivalent to a list) for a collection of rest terms
discount(?customer, ?product, percent5) :- premium(?customer), regular(?product).

premium(PeterMiller).
regular(Honda).

premium(cust->PeterMiller).
regular(prod->Honda).
OO jDREW Types

- Order-sorted type system
  - RDF Schema: lightweight taxonomies of the Semantic Web
  - To specify a partial order for a set of classes in RDFS

- Advantages
  - Having the appropriate types specified for the parameters
  - To restrict the search space
  - Faster and more robust system than when reducing types to unary predicate calls in the body

- Limitations
  - Only modeling the taxonomic relationships between classes
  - Not modeling properties with domain and range restrictions
base_price(customer->[sex->male; !]; vehicle:"Car"; price->650:Integer).
base_price(customer->[sex->male; !]; vehicle:"Van"; price->725:Integer).
OO jDREW OIDs

- **oid**: Object Identifier
- Currently: symbolic names
  - In `<Atom>` & `<Implies>`
- Planned: `uri` attribute
- E.g., give name to fact `keep(Mary, ?object).

```xml
<Atom>
  <oid><Ind mary-12</Ind></oid>
  <Rel>keep</Rel>
  <Ind>Mary</Ind>
  <Var>object</Var>
</Atom>

<Atom>
  <oid><Ind uri="http://mkb.ca"/></Ind>
  <Rel>keep</Rel>
  <Ind>Mary</Ind>
  <Var>object</Var>
</Atom>

<Atom>
  <oid><Var>object</Var></oid>
  <Rel>keep</Rel>
  <Ind>Mary</Ind>
  <Var>object</Var>
</Atom>
```
Conclusions

- RuleML is modular family, whose root allows to access the language as a whole and whose members allow customized subsets
- New members joining, e.g. Fuzzy RuleML
- Concrete & abstract syntax of RuleML
  - Specified by modular XSD (shown here) & MOF
- Formal semantics of OO Hornlog RuleML
  - Implemented by OO jDREW BU & TD
- Interoperability/Interchange of/with RuleML
  - Realized by translators, primarily via XSLT