How Object-Centered XML Rules are Configured from the RuleML Lattice, can be Exchanged in JSON, and enable Decision Making over Object-Relational Data

Harold Boley
University of New Brunswick
Chair RuleML

Introduction: Objects & Decisions

- RuleML decision rules are **object-centered** in multiple ways
- Decision languages configurable from desired properties in **lattice of schema objects** for RuleML/XML to obtain precisely required language expressivity, hence engine efficiency
- Decision rulebases serialized in RuleML via ‘fully striped’ normal form of **object elements / property subelements**
- RuleML uses JSON as alternate presentation & exchange syntax, **RuleML/JSON**, round-trippable with RuleML/XML, while keeping RuleML's schema definitions unchanged
- PSOA RuleML enables integration of **relational plus object-centered (graph)** data and rules for decision making
Part I: Rule Configuration from the RuleML Lattice

MYNG: Modular Syntax configurator

For detailed slides see DemoMYNG1.01
Point of Departure: RuleML 1.0 Hierarchy
Point of Departure: RuleML 1.0 Hierarchy

See (RuleML 2014 materials):
- Rules, Events and Actions
- Keynote as well as Reaction
- RuleML 1.0 Demo

Optional mix-in of subClassOf
Syntactic specialization of
RuleML Family of Sublanguages

- RuleML family covers a wide rule spectrum: Deliberation RuleML (1.01) ... Reaction RuleML (1.0)
  - Rule condition part reused across the spectrum
  - Syntactic uniformity enables further reuse
- Family constitutes a deep sublanguage lattice
  - Major sublanguage inclusion path:
    Deliberation ⊃ HOL ⊃ FOL ⊃ Derivation ⊃ Hornlog ⊃ Datalog ⊃ ...
- Naf mix-in customization of Hornlog RuleML (Naf Hornlog RuleML) leads to Logic Programs
Configure Your Own RuleML Language from Over 6 Billion via 2 MYNG Box Rows

- Delib RuleML 1.01 schemas customized by MYNG
- Key new **MYNG 1.01 technology** includes
  - Integration of new Relax NG schema modules – and RuleML sublanguages they define – into MYNG, e.g.
    - Datalog\(^+\), Hornlog\(^+\), and their many extensions
  - Improved functionality of the MYNG GUI and REST interface, e.g.
    - GUI access to automatically generated monolithic XSD schemas that are compatible with XML tools, e.g. JAXB
    - Myng-code display and URL access

**See:** [MYNG 1.01 Challenge Demo](#)
Delib RuleML 1.01 Sublanguages Customized by MYNG 1.01 as Relax NG Schemas (1)
Delib RuleML 1.01 Sublanguages Customized by MYNG 1.01 as Relax NG Schemas (2)

**Usage**

The RNC and XSD Schema URLs may be used directly for online validation - copy and paste as required by the validator. For a demonstration of RNC validation using the online service Validator.nu, see How to Validate with the RuleML Parameterized Relax NG Schema. Some scripts and processing instructions may require that the character "&" be replaced by "&amp;".

Relax NG Schema URL = http://deliberation.ruleml.org/1.01/relaxng/schema_rnc.php?backbone=x3f&default=x7&termseq=x7&lng=x1&propo=x3ff&implies=x7f&terms=x3f&quant=x7&expr=x7&serial=x7

XSD Anchor Schema URL = http://deliberation.ruleml.org/1.01/xsd/naffologeq.xsd
Finer Delib RuleML Modularization: Schema Spec

- **Deliberation RuleML 1.0**: Language lattice retains unnecessary ‘hierarchical restrictions’ to modularity
  - Example: Only Hornlog and ‘up’, not Datalog, allow `<Or>` in the `<then>` parts, so Disjunctive *Datalog* rulebases cannot be validated precisely (only be ‘underspecified’ as Disjunctive *Hornlog*)

- **Deliberation RuleML 1.01**: Language lattice has less hierarchical, finer-grained modularity
  - Example: allows `<Or>` in all `<then>` parts, e.g. for precise validation of Disjunctive *Datalog*
Finer Delib RuleML Modularization: Rule Instance

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="http://deliberation.ruleml.org/1.01/relaxng/disdatalog_min_relaxed.rnc"?>
<RuleML xmlns="http://ruleml.org/spec">
  <Assert>
    <Forall>
      <Var>x</Var>
      <Implies>
        <if>
          <Atom>
            <Rel>integer</Rel>
            <Var>x</Var>
          </Atom>
        </if>
        <then>
          <Or>
            <Atom>
              <Rel>even</Rel>
              <Var>x</Var>
            </Atom>
            <Atom>
              <Rel>odd</Rel>
              <Var>x</Var>
            </Atom>
          </Or>
        </then>
      </Implies>
    </Forall>
  </Assert>
</RuleML>
```
Deliberation RuleML 1.01 Features: Datalog Extensions Yield Datalog$^+$

- **Existential Rules**, where the <then> part of a rule has existentially quantified variables, as needed for [DL (e.g. OWL)](https://www.w3.org/2003/06/ OWL/), F-logic, [PSOA RuleML](https://www.ruleml.org/), [Rule-Based Data Access (RBDA)](https://www.ruleml.org/), etc.

- **Equality Rules**, where the <then> part of a rule is the <Equal> predicate, as needed for user-defined/'semantic' equality in [logics with equality](https://en.wikipedia.org/wiki/Logic_with_equality) and [functional logic programming](https://en.wikipedia.org/wiki/Functional_logic_programming) (this was already allowed in RuleML 1.0)

- **Integrity Rules**, where the <then> part of a rule is falsity, as a convenient way to express negative [integrity constraints](https://en.wikipedia.org/wiki/Integrity_constraint)
Sublattice Focused in MYNG 1.01 Demo
Delib RuleML 1.01 Orthogonality: *Hornlog* Extensions Yield *Hornlog*\(^+\)

- Because of modular schema design, all new features of Delib RuleML 1.01 **freely combinable**, via module inclusion,
  - with each other
  - with existing RuleML sublanguages
- Features available for other logics in Delib RuleML, including Horn logic (*Hornlog RuleML* 1.01), e.g. for *Hornlog*\(^+\) combo of
  - *Hornlog Existential Rules*
  - *Hornlog Equality Rules*
  - *Hornlog Integrity Rules*
Part II: RuleML in JSON

Reuse schemas, stylesheets, and tools (e.g. rule engines), based on XML-JSOM match

Overview (1)

- JavaScript Object Notation (JSON) is human- and machine-readable data-interchange format for
  - lists (ordered) of values and
  - sets (unordered) of name-value pairs
- RuleML uses XML in the positional (ordered) style and in a novel object-centered (unordered) style
- Sharing this basic ordered/unordered distinction, RuleML and JSON can be matched to each other pretty well
Overview (2)

- RuleML can employ JSON as an alternate presentation and interchange syntax
- Round-trippable with RuleML/XML serialization
- Keeping schema definitions of RuleML unchanged (using XML Schema Definition and Relax NG, as e.g. for Deliberation RuleML)
Overview (3)

• RuleML's main data model distinction:
  • Node (Type) elements start with upper-case letter
  • edge (role) elements start with a lower-case letter
• Correspond to RuleML's two normal forms:
  • Compact *stripe-skipped* (ordered)
  • Expanded *fully striped* forms (unordered)
• Focus on JSON's two respective structures:
  • Lists: *arrays*
  • Pair sets: *objects*
Overview (4)

- Each nesting level allows arbitrary combinations of RuleML/XML which is
  - stripe-skipped ('positional')
  - fully striped ('object-centered')
- Such RuleML/XML elements can be translated to corresponding RuleML/JSON combinations of arrays and objects in a round-trippable (bi-directional) fashion
Overview (5)

- XML tools, e.g. XSLT, used for XML-to-JSON direction, could be made generally aware of Node/edge (upper-case/lower-case) distinction.
- However, for RuleML/XML, XSLT stylesheet will take care of this distinction as part of its exhaustive analysis of all (Node and edge) elements used in RuleML sublanguage to be translated.
Overview (6)

- RuleML-JSON mapping mostly allows **direct modeling**
- But some need for **indirect encoding**:
  1. Like for any XML-JS0N mapping, XML's elements must be encoded, and we adopted usual encoding as **one-pair objects** (to be refined via XML attributes)
  2. RuleML has **multi-valued edges** like `<formula>`, while JSON's name : value pairs SHOULD have a unique name within an object; we thus use the recommended and widely implemented arrays for encoding multiple values (although array order information needs to be disregarded)
Overview (7)

- The resulting two kinds of arrays are disambiguated via occurrence in two different syntactic contexts:
  - Each **Node's** array value directly models the intended child order
  - Each **edge's** array value indirectly encodes its single value or multiple values without intending the textual child order (taken from the XML) to carry information
- Should a future version of JSON definitely allow non-unique names for pairs, this encoding would no longer be needed
RuleML XML-to-JSON Translation Principles

- Later slides show principles for the two basic cases (dots indicate recursive translation)
- To emphasize tree structures of XML and JSON, we will employ pretty-print layouts
- For clarity, the RuleML/JSON pretty-print layout is "encoding-aware" in that it displays structures (arrays and objects) depending on whether they perform 1. direct modeling or 2. indirect encoding (of XML's elements as one-pair objects and RuleML's single/multiple values as arrays)
The modeling/encoding distinction is emphasized through different displays of opening curly braces (for objects) and square brackets (for arrays):

1. Direct modeling: Opening curly braces and square brackets are separated from (the first part of) their content by whitespace (used here: newline)

2. Indirect encoding: Opening curly braces and square brackets are not separated from (the first part of) their content by whitespace
Node with Nodes as Children

RuleML/XML element Node0 with subelements Node1 . . . NodeK

```
<Node0>
  <Node1>...</Node1>
  . . .
  <NodeK>...</NodeK>
</Node0>
```

becomes RuleML/JSON object having one pair with name Node0 whose value is array of one-pair objects for Node1 . . . NodeK,

```
{"Node0":[
  {"Node1" : ...},
  . . .
  {"NodeK" : ...}]}
```
Node with Nodes as Children: Ex.

For example, with \( K=2 \),

\[
\begin{align*}
&<\text{Implies}> \\
&<\text{And}> \ldots <\text{And}> \\
&<\text{Atom}> \ldots <\text{Atom}>
\end{align*}
\]

becomes

\[
\{
"\text{Implies}": [
"\text{And}": \ldots ,
"\text{Atom}": \ldots ]
\}
\]
Node with edges as Children (1)

RuleML/XML element Node0 containing a unique edge

```xml
<Node0>
  . . .
  <edgeI>...</edgeI>
  . . .
</Node0>
```

becomes RuleML/JSON object having one pair with name Node0 whose value is object including pair with edgeI child in its array value,

```
{"Node0":{
  . . .
  "edgeI":[...],
  . . .}}
```
Node with edges as Children, Ex. (1)

As an example of a Node with two unique edges,

```xml
<Implies>
  <if>
    <And>...</And>
  </if>
  <then>
    <Atom>...</Atom>
  </then>
</Implies>
```

becomes

```json
{"Implies":{
  "if": {
    "And": [...]
  },
  "then": {
    "Atom": [...]
  }
}}
```
Node with edges as Children (2)

RuleML/XML element Node0 containing multiple edgeI's

```
<Node0>
  ...
  <edgeI>...</edgeI>
  ...
  <edgeI>...</edgeI>
  ...
</Node0>
```

becomes RuleML/JSON object having one pair with name Node0 whose value is object containing pair with consecutive edgeI children in its array value,

```
{"Node0":{
  ...
  "edgeI":
    [...,
    ...],
  ...
}}
```
Node with edges as Children, Ex. (2)

Refining previous example at child Node with two non-unique edges,

\[
\text{\textless And}\textgreater  \\
\text{\textless formula}\textgreater  \\
\text{\textless Atom}\textgreater  \\
\text{\textless op\textgreater Rel\textless buy\textgreater Rel\textgreater \textless op\textgreater} \\
\ldots  \\
\text{\textless /Atom\textgreater}  \\
\text{\textless /formula\textgreater}  \\
\text{\textless formula\textgreater}  \\
\text{\textless Atom\textgreater}  \\
\text{\textless op\textgreater Rel\textless keep\textgreater Rel\textgreater \textless op\textgreater} \\
\ldots  \\
\text{\textless /Atom\textgreater}  \\
\text{\textless /formula\textgreater}  \\
\text{\textless /And\textgreater}
\]

becomes

\[
\{
\text{"And":{}
\text{ "formula":}
\text{ ["Atom":{
\text{ "op":["Rel":"buy"},
\ldots
\text{ }},
\text{ "Atom":{
\text{ "op":["Rel":"keep"},
\ldots
\text{ }]}]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
\text{ }]
Sublanguage Datalog RuleML XML Serialization, Grailog Visualization for Extended KB

```
<Assert mapClosure="universal">
  <Implies>
    <!-- explicit 'And' -->
    <And>
      <Atom>
        <Rel>buy</Rel>
        <Var>person</Var>
        <Var>merchant</Var>
        <Var>object</Var>
      </Atom>
      <Atom>
        <Rel>keep</Rel>
        <Var>person</Var>
        <Var>object</Var>
      </Atom>
      <Atom>
        <Rel>own</Rel>
        <Var>person</Var>
        <Var>object</Var>
      </Atom>
    </And>
  </Implies>
</Assert>
```
If a person buys an object from a merchant and the person keeps the object then that person owns the object.
Sub...Sublanguage Datalog RuleML
XML Serialization, Parse-Tree Visualization

44  <Assert mapClosure="universal">
45
46  <Implies>
47    <!-- explicit 'And' -->
48    <And>
49      <Atom>
50        <Rel>buy</Rel>
51        <Var>person</Var>
52        <Var>merchant</Var>
53        <Var>object</Var>
54      </Atom>
55      <Atom>
56        <Rel>keep</Rel>
57        <Var>person</Var>
58        <Var>object</Var>
59      </Atom>
60    </And>
61    <Atom>
62      <Rel>own</Rel>
63      <Var>person</Var>
64      <Var>object</Var>
65    </Atom>
66  </Implies>
Stripe-skipped RuleML/XML

<Implies>
  <And>
    <Atom>
      <Rel>buy</Rel>
      <Var>person</Var>
      <Var>merchant</Var>
      <Var>object</Var>
    </Atom>
    <Atom>
      <Rel>keep</Rel>
      <Var>person</Var>
      <Var>object</Var>
    </Atom>
    <Atom>
      <Rel>own</Rel>
      <Var>person</Var>
      <Var>object</Var>
    </Atom>
  </And>
</Implies>

Complete Example: Stripe-Skipped Array-focusing RuleML/JSON

{"Implies":[
  {"And":[
    {"Atom":[
      {"Rel":"buy"},
      {"Var":"person"},
      {"Var":"merchant"},
      {"Var":"object"}]},
    {"Atom":[
      {"Rel":"keep"},
      {"Var":"person"},
      {"Var":"object"}]},
    {"Atom":[
      {"Rel":"own"},
      {"Var":"person"},
      {"Var":"object"}]}
  ]}
}

One-pair objects used to associate an element with its content; arrays used for the positional subelements
Complete Example: Fully Striped

Fully striped RuleML/XML

```xml
<Implies>
  <if>
    <And>
      <formula>
        <Atom>
          <op><Rel>buy</Rel></op>
          <arg index="1"><Var>person</Var></arg>
          <arg index="2"><Var>merchant</Var></arg>
          <arg index="3"><Var>object</Var></arg>
        </Atom>
      </formula>
      <formula>
        <Atom>
          <op><Rel>keep</Rel></op>
          <arg index="1"><Var>person</Var></arg>
          <arg index="2"><Var>object</Var></arg>
        </Atom>
      </formula>
    </And>
  </if>
  <then>
    <Atom>
      <op><Rel>own</Rel></op>
      <arg index="1"><Var>person</Var></arg>
      <arg index="2"><Var>object</Var></arg>
    </Atom>
  </then>
</Implies>
```

Object-focusing RuleML/JSON

```json
{"Implies": {
  "if": {
    "And": {
      "formula": {
        "Atom": {
          "op": ["Rel": "buy"],
          "arg": ["@index": "1", "Var": "person"],
          "@index": "2", "Var": "merchant"],
          "@index": "3", "Var": "object"]
      }
    },
    "Atom": {
      "op": ["Rel": "keep"],
      "arg": ["@index": "1", "Var": "person"],
      "@index": "2", "Var": "object"
    }
  },
  "then": {
    "Atom": {
      "op": ["Rel": "own"],
      "arg": ["@index": "1", "Var": "person"],
      "@index": "2", "Var": "object"
    }
  }
}}
```

Objects used for describing Node elements with edge elements; attributes distinguished form edge elements by an "@" prefix; arrays used for encoding single/multi-valued edge children.
Complete Example: Partially Striped

Partially striped RuleML/XML  Array/Object RuleML/JSON

<Implies>
  <if>
    <And>
      <Atom>
        <Rel>buy</Rel>
        <Var>person</Var>
        <Var>merchant</Var>
        <Var>object</Var>
      </Atom>
      <Atom>
        <Rel>keep</Rel>
        <Var>person</Var>
        <Var>object</Var>
      </Atom>
    </And>
  </if>
  <then>
    <Atom>
      <op><Rel>own</Rel></op>
      <arg index="1">person</arg>
      <arg index="2">object</arg>
    </Atom>
  </then>
</Implies>

{"Implies":{
  "if":
  [{"And":[
    {"Atom":[
      {"Rel":"buy"},
      {"Var":"person"},
      {"Var":"merchant"},
      {"Var":"object"}]},
    {"Atom":[
      {"Rel":"keep"},
      {"Var":"person"},
      {"Var":"object"}]}
  ]},
  "then":
  [{"Atom":{
    "op":{"Rel":"own"},
    "arg":{[@index":"1","Var":"person"],
      [@index":"2","Var":"object"]}
  }}]}}
Part III: Decision Making over Object-Relational Data

Example: Graph

(PSOA Positional-Slotted-Term) Logic

marrried(Joe Sue) kid(Sue Pete) )
Rule Languages

- Paradigms for modeling entity dependencies:
  - Relational
  - Graph (Object-Centered)
  - Combined

- Since Knowledge Bases (KBs) have been developed in languages following all three paradigms, cross-paradigm translation, integration, and reuse is often necessary

- Need for an interoperation language and technology: Positional-Slotted Object-Applicative (PSOA) RuleML

- Naturally combinable with portability technology: Platform-independent implementation of PSOA RuleML
Hypergraph Example – Relational Betweenness

Directed hyperarcs cut through intermediate nodes (cf. Grailog)

Facts

\[
\text{betweenRel}(\text{pacific}, \text{canada}, \text{atlantic}) \\
\text{betweenRel}(\text{canada}, \text{usa}, \text{mexico})
\]
Graph Example – Object-Centered Betweenness

**Facts**

“#” denotes “∈” for class membership; “→” associates a slot name with its filler

\[ b_0 \# \text{betweenObj} (outer1 \rightarrow \text{pacific}; \ inner \rightarrow \text{canada}; \ outer2 \rightarrow \text{atlantic}) \]

\[ b_1 \# \text{betweenObj} (outer1 \rightarrow \text{canada}; \ inner \rightarrow \text{usa}; \ outer2 \rightarrow \text{mexico}) \]
Example – Integrated Betweenness (Enriched)

betweenObjRel

westEast

orient

b0

pacific

canada

atlantic

dim

2

usa

mexico

northSouth

dim

orient

b1

Facts

b0 # betweenObjRel (pacific, canada, atlantic; dim → 2; orient → westEast)
b1 # betweenObjRel (canada, usa, mexico; dim → 2; orient → northsouth)
Relational Rule Languages

- Widely used for relational DBs (SQL views) and KBs, representing information in classical logic
- Model dependencies among $n$ entities as an $n$-ary predicate applied to an ordered sequence of $n$ arguments, called *positional arguments*
- Languages: Common Logic, Prolog, TPTP-FOF, ...
Graph (Object-Centered) Rule Languages

- Receive increasing attention because of expanding research and development in linked data on the Web, graph/‘triple’ stores, and big data in NoSQL DBs
- Each object is represented by a unique Object IDentifier (OID), typed by a class, and described by an unordered collection of slots, each being a pair of a name and a filler
- An OID-describing slotted term in AI is called a frame (represents a resource/‘subject’-describing property list on the Semantic Web)
- Languages: RDF, N3, ...
Object-Relational Rule Languages

- Combine the object-centered and relational paradigms, either in a heterogeneous or a homogeneous way
- Heterogeneous
  - Allow atomic formulas in both relational and object-centered forms, even mixed in the same rule
  - Languages: F-logic and RIF
- Homogeneous
  - Integrate relational and object-centered atomic formulas into a unified form
  - Language: PSOA RuleML
**PSOA RuleML**

- Integrates relational and object-centered modeling
- Generalizes F-logic, RIF-BLD, and POSL
- Uses **positional-slotted object-applicative (psoa)** terms, permitting a relation application to have an OID – typed by the relation – and, orthogonally, its arguments to be positional or slotted.

**General case (multi-tuple):**
\[
\circ \#f([t_{1,1} \ldots t_{1,n_1}] \ldots [t_{m,1} \ldots t_{m,n_m}] \ p_1 \rightarrow v_1 \ldots p_k \rightarrow v_k)
\]

**Special cases (single-tuple brackets and zero-argument parentheses optional):**
- **Combined:**  \[\circ \#f([t_1 \ldots t_n] \ p_1 \rightarrow v_1 \ldots p_k \rightarrow v_k)\]
- **Positional:**  \[\circ \#f([t_1 \ldots t_n])\]
- **Slotted:**  \[\circ \#f(p_1 \rightarrow v_1 \ldots p_k \rightarrow v_k)\]
- **Member-only:**  \[\circ \#f()\]
Example of Querying a PSOA Fact and Rule

KB:

bl#betweenObjRel(canada usa mexico
dim->2 orient->northSouth)

For all ?out1 ?in ?out2 ?b

( ?in#GeoUnit(neighborNorth->?out1
neighborSouth->?out2) :-
?b#betweenObjRel(?out1 ?in ?out2
orient->northSouth)
)

English Query: “Which GeoUnit has Canada as its northern neighbor?”
Query: ?X#GeoUnit(neighborNorth->canada)
Answer: ?X=usa

TPTP version in Common Logic would contribute to **COLORE**
Conclusion: Getting Involved

- Tutorial introduction ([http://ruleml.org/papers/Primer](http://ruleml.org/papers/Primer))
- RuleML MediaWiki ([http://wiki.ruleml.org](http://wiki.ruleml.org))
  - Includes the Wiki Issue system for users to request enhancements and report errata
- RuleML Blog & Social Mediazine ([http://blog.ruleml.org](http://blog.ruleml.org))
  - Enabled Public Review of Deliberation RuleML 1.01
  - Has become a resource in its own right (courtesy to Binarypark)
- RuleML sources hosted on Github ([https://github.com/RuleML](https://github.com/RuleML))