PSOA RuleML: Linked Objects and Rules,

Talk at Semantic Days 2011 Oslo, Norway, 7-9 June 2011

Harold Boley

Institute for Information Technology
National Research Council of Canada
Fredericton, NB, E3B 9W4, Canada

Motivation: Linked Object-and-Rule Interchange

- Linking is characteristic for modern object and rule languages on the (Social Semantic) Web
- Employ IRIs – Internationalized Resource Identifiers – as OIDs – Object IDentifiers – (which can also occur as slot fillers), slot keys, type identifiers, and document identifiers for object & rule bases (modules)
- Linked objects and rules can thus be regarded as generalized linked data
- Therefore, object & rule language such as RuleML usable for exchange of information ranging from linked data sets to linked Horn/F-logic knowledge bases to linked first-order and higher-order logic documents

Introduction: Two IT Paradigms

Knowledge representation & problem solving in

- AI
- the (Semantic) Web
- IT at large

can be

1. Logic-based:
   FOL, Horn, LP
2. Object-oriented (and frame-based):
   CLOS, RDF, N3

Combined approaches:

- Description Logics (DLs)
- Object-Oriented Databases (OODBs) /
  Deductive Object-Oriented Databases (DOODs)
- Object-oriented logic languages:
  LIFE and Frame logic (F-logic)
- W3C Rule Interchange Format (RIF):
  - Semantics based on F-logic
  - Serialization syntax based on RuleML

Introduction: Object-Extended Semantics

- F-logic and RIF extend first-order model-theoretic semantics for **objects (frames)**
- Added separately from **function and predicate applications to arguments**
- Resulting complexity of object-extended semantics can be reduced by **integrating objects with applications**

Introduction: Object-Extended Semantics (Cont'd)

- Integration based on **positional-slotted, object-applicative** rules of POSL and RuleML
- **F-logic's model-theoretic semantics in the style of RIF** is also the **starting point** of our **integrated semantics**
- Permits **applications with optional object identifiers** and, orthogonally, **arguments that are positional or slotted**
- Structured by these **independent dimensions** of defining features, language **constructs can be freely combined**

- Integration based on **p**ositional-**s**lotted, **o**bject-**a**pplicative (***psoa***) terms and rules
- Psoa term applies **function or predicate** symbol, possibly **instantiated by object**, to zero or more **positional or slotted (named)** arguments
- For a psoa term as **atomic formula**, predicate symbol is **class (type) of object** as well as **relation between arguments**, which describe object
- Each **argument** of a psoa term can be psoa term applying **function** symbol

- Psoa terms that apply a predicate symbol (as a relation) to *positional arguments* can be employed to make factual assertions

- An example, in simplified RIF (presentation) syntax, is term `married(Joe Sue)` for binary predicate `married` applied to `Joe` and `Sue`, where positional (left-to-right) order can be used to identify husband, as 1st argument, and wife, as 2nd argument

- Psoa terms that apply a predicate symbol (as a class) to *slotted arguments* correspond to typed attribute-value descriptions
- An example is psoa term `family(husb->Joe wife->Sue)` or `family(wife->Sue husb->Joe)` for `family`-typed attribute-value pairs (slots) `{<husb,Joe>, <wife,Sue>}`
  - Easily extended with further slots, e.g. by adding children, as in `family(husb->Joe wife->Sue child->Pete)`

Introduction: Distinctions in Psoa Taxonomy (Cont'd)

- Usually, slotted terms describe an object symbol, i.e. an object identifier (OID), maintaining object identity even when slots of their descriptions are added or deleted
  - This leads to (typed) frames in the sense of F-logic
- E.g., using RIF's membership syntax #, OID `inst1` in class `family` is describable by `inst1#family(husb->Joe wife->Sue)`, `inst1#family(husb->Joe wife->Sue child->Pete)`, etc. Psoa terms can also specialize to class membership terms, e.g. `inst1#family()`, abridged `inst1#family`, represents `inst1` $\in$ `family`

- Like OID-describing slotted terms constitute a (multi-slot) ‘frame’, positional terms that describe an object constitute a (single-tuple) ‘shelf’, similar to a (one-dimensional) array describing its name
- Thus, `family`’s `husb` and `wife` slots can be positionalized as in earlier `married` example: `inst1#family(Joe Sue)` describes `inst1` with tuple `[Joe Sue]`

Introduction: Positional-Slotted OID Description

- *Combined positional-slotted* psoa terms are allowed, similarly as in XML elements (tuple$\leadsto$subelements, slots$\leadsto$attributes), e.g. describing an object, as in RDF descriptions (object$\leadsto$subject, slots$\leadsto$properties)
- For example, `inst1#family(Joe Sue child->Pete)` describes `inst1` with two positional and one slotted argument

Introduction: Positional and Slotted OID Description

- On the other hand, positional `married` example could be made slotted, leading to

  `married(husb->Joe wife->Sue),`

  and even be used to describe a (marriage) object: positionally, as in

  `inst2#married(Joe Sue),`

  or slotted, as in

  `inst2#married(husb->Joe wife->Sue)`

Introduction: Implicit Typing with Root Class

- A **frame without explicit class** is semantically treated as **typing its object with root class** $\top$ (syntactically, `Top`)
- For example, (untyped) frame
  `inst3[color->red shape->diamond]`
  in square-bracketed F-logic/RIF syntax is equivalent to our parenthesized
  `inst3#Top(color->red shape->diamond)`

Introduction: Atom Objectification

- An **atomic formula without OID** is treated as **having implicit OID**
- An OID-less application is *objectified* by syntactic transformation: *The OID of a ground fact is new constant generated by 'new local constant' (stand-alone _); the OID of non-ground fact or atomic formula in rule conclusion,* `f(...)`*, is new, existentially scoped variable* `?i`*, leading to* `Exists ?i (?i#f(...))`*; the OID of other atomic formulas is new variable generated by 'anonymous variable' (stand-alone* `?`*)*
- Objectification allows compatible semantics for an atom constructed as RIF-like slotted (named-argument) term and corresponding frame, solving issue with named-argument terms:

http://lists.w3.org/Archives/Public/public-rif-wg/2008Jul/0000.html

- For example, slotted-fact assertion `family(husb->Joe wife->Sue)` is syntactically objectified to assertion `_#family(husb->Joe wife->Sue)`, and — if `_1` is first new constant from `_1`, `_2`, ... — to `_1#family(husb->Joe wife->Sue)`
- This typed frame, then, is semantically *slotributed* to `_1#family(husb->Joe)` and `_1#family(wife->Sue)`

Introduction: Objectified Atom Querying

- Query `family(husb->Joe)` is syntactically objectified to query `?#family(husb->Joe)`, i.e. — if `?1` is first new variable in `?1`, `?2`, ... — to `?1#family(husb->Joe)`
- Posed against fact, it succeeds with first slot, unifying `?1` with `_1`
- Slotribution ('slot distribution') avoids POSL's 'rest-slot' variables: frame's OID 'distributes' over slots of a description

Introduction: Psoa Rules

- Rules can be defined on top of psoa terms in a natural manner
- A rule derives (a conjunction of possibly existentially scoped) conclusion psoa atoms from (a formula of) premise psoa atoms
- Consider example with rule deriving `family` frames

Introduction: Psoa Rules Exemplified

Example (Rule-defined anonymous family frame)

`Group` is used to collect a rule and two facts. `Forall` quantifier declares orginal universal argument variables and generated universal OID variables `?2`, `?3`, `?4`. Infix `:-` separates conclusion from premises of rule, which derives anonymous/existential `family` frame from `married` relation `And` from `kid` relation of `husb` `Or` `wife` (the left-hand side is objectified on the right).

```
Group (
 Forall ?Hu ?Wi ?Ch (

  family(husb->?Hu wife->?Wi child->?Ch):-
   And(married(?Hu ?Wi)
       Or(kid(?Hu ?Ch) kid(?Wi ?Ch))) )
 married(Joe Sue)
 kid(Sue Pete)
      )
```

```
Group (
 Forall ?Hu ?Wi ?Ch ?2 ?3 ?4 (
  Exists ?1 (
   ?1#family(husb->?Hu wife->?Wi child->?Ch)) :-
    And(?2#married(?Hu ?Wi)
        Or(?3#kid(?Hu ?Ch) ?4#kid(?Wi ?Ch))) )
 _1#married(Joe Sue)
 _2#kid(Sue Pete)
      )
```

Semantically, example is modeled by predicate extensions corresponding to following set of ground facts (the subdomain of individuals $\boldsymbol{D}_{\text{ind}}$ is to be defined):

{*o#family(husb->Joe wife->Sue child->Pete)* } ∪

{*_1#married(Joe Sue)*, *_2#kid(Sue Pete)*}, where $o \in \boldsymbol{D}_{\text{ind}}$.

Presentation Syntax: Alphabet

Definition (Alphabet)

The ***alphabet*** of the presentation language of PSOA RuleML consists of the following disjoint sets:

- A countably infinite set of ***constant symbols*** `Const` (including the root class `Top` ∈ `Const` and the positive-integer-enumerated local constants `_1`, `_2`, `...` ∈ `Const` as well as individual, function, and predicate symbols)
- A countably infinite set of ***variable symbols*** `Var` (including the positive-integer-enumerated variables `?1`, `?2`, `...` ∈ `Var`)
- The connective symbols `And`, `Or`, and `:-`
- The quantifiers `Exists` and `Forall`
- The symbols `=`, `#`, `##`, `->`, `External`, `Import`, `Prefix`, and `Base`
- The symbols `Group` and `Document`

Presentation Syntax: Alphabet (Cont'd)

Definition (Alphabet, Cont'd)

Constants have form `"literal"^^symspace`, where `literal` is a sequence of Unicode characters and `symspace` is an identifier for a symbol space. E.g., `"_123"^^rif:local`. Constants use shortcuts defined in RIF-DTB, including underscored `_literal` (e.g., `_123`) for above form with `symspace` specialized to `rif:local`. `Top` is a new shortcut for root class constant `"Top"^^psoa:global` in PSOA RuleML's global symbol space

Anonymous variables are written as a stand-alone question mark symbol (`?`); named variables, as Unicode strings preceded with question mark symbol

Symbols `=` and `##` are used in formulas that define equality and subclass relationships. The symbols `#` and `->` are used in positional-slotted, object-applicative formulas for class memberships and slots, respectively. Symbol `External` indicates that an atomic formula or a function term is defined externally (e.g., a built-in) and symbols `Prefix` and `Base` enable abridged representations of IRIs (Internationalized Resource Identifiers) □

Presentation Syntax: Terms

In this definition, *base term* means a simple term, an anonymous psoa term (i.e., an anonymous frame term, single-tuple psoa term, or multi-tuple psoa term), or a term of the form `External(t)`, where `t` is an anonymous psoa term. Anonymous term can be *deobjectified* (by omitting main `?#`) if its re-objectification results in old term (i.e., re-introduces `?#`).

Definition (Term)

1. *Constants and variables*. If $\mathtt{t} \in \mathtt{Const}$ or $\mathtt{t} \in \mathtt{Var}$ then `t` is a ***simple term***
2. *Equality terms*. `t = s` is an ***equality term*** if `t`, `s` are base terms
3. *Subclass terms*. `t##s` is a ***subclass term*** if `t`, `s` are base terms
4. *Positional-slotted, object-applicative terms*. $\mathtt{o\#f([t_{1,1}...t_{1,n_1}]...[t_{m,1}...t_{m,n_m}]\ p_1\text{->}v_1...p_k\text{->}v_k)}$ is a ***positional-slotted, object-applicative (psoa) term*** if $\mathtt{f} \in \mathtt{Const}$ and $\mathtt{o}, \mathtt{t_{1,1}}, ..., \mathtt{t_{1,n_1}}, \ldots, \mathtt{t_{m,1}}, ..., \mathtt{t_{m,n_m}}, \mathtt{p_1}, ..., \mathtt{p_k}, \mathtt{v_1}, ..., \mathtt{v_k}, \mathtt{m} \geq 0, \mathtt{k} \geq 0$, are base terms

Presentation Syntax: Terms (Cont'd)

Definition (Term, Cont'd)

Psoa terms can be specialized in the following way

- For `m = 0` they become ***(typed or untyped) frame terms*** $\texttt{o\#f(p}_1\texttt{->v}_1\ \texttt{...}\ \texttt{p}_k\texttt{->v}_k\texttt{)}$. We consider two overlapping subcases
  - For `k = 0` they become ***class membership terms*** `o#f()`, abridged to `o#f`, corresponding to those in F-logic and RIF
  - For $\texttt{k} \geq \texttt{0}$ they can be further specialized in two ways, which can be orthogonally combined
    - For `o` being the anonymous variable `?`, they become ***anonymous frame terms (slotted terms)*** $\texttt{?\#f(p}_1\texttt{->v}_1\ \texttt{...}\ \texttt{p}_k\texttt{->v}_k\texttt{)}$, deobjectified $\texttt{f(p}_1\texttt{->v}_1\ \texttt{...}\ \texttt{p}_k\texttt{->v}_k\texttt{)}$, corresponding to *terms with named arguments* in RIF
    - For `f` being the root class `Top`, they become ***untyped frame terms*** $\texttt{o\#Top(p}_1\texttt{->v}_1\ \texttt{...}\ \texttt{p}_k\texttt{->v}_k\texttt{)}$ corresponding to *frames* in abridged form $\texttt{o[p}_1\texttt{->v}_1\ \texttt{...}\ \texttt{p}_k\texttt{->v}_k\texttt{]}$ of F-logic and RIF, where square brackets are used instead of round parentheses

Presentation Syntax: Terms (Cont'd)

Definition (Term, Cont'd)

- For `m = 1` they become ***single-tuple psoa terms*** $\texttt{o\#f([t}_{1,1}\ \ldots\ \texttt{t}_{1,n_1}\texttt{] p}_1\texttt{->v}_1\ \ldots\ \texttt{p}_k\texttt{->v}_k\texttt{)}$, abridged to $\texttt{o\#f(t}_{1,1}\ \ldots\ \texttt{t}_{1,n_1}\ \texttt{p}_1\texttt{->v}_1\ \ldots\ \texttt{p}_k\texttt{->v}_k\texttt{)}$

  These can be further specialized in two ways, which can be orthogonally combined:

  - For `o` being the anonymous variable `?`, they become ***anonymous single-tuple psoa terms*** $\texttt{?\#f(t}_{1,1}\ \ldots\ \texttt{t}_{1,n_1}\ \texttt{p}_1\texttt{->v}_1 \ldots \texttt{p}_k\texttt{->v}_k\texttt{)}$, deobjectified $\texttt{f(t}_{1,1}\ \ldots\ \texttt{t}_{1,n_1}\ \texttt{p}_1\texttt{->v}_1 \ldots \texttt{p}_k\texttt{->v}_k\texttt{)}$. These can be further specialized:
    - For `k = 0`, they become ***positional terms*** $\texttt{?\#f(t}_{1,1}\ \ldots\ \texttt{t}_{1,n_1}\texttt{)}$, deobjectified $\texttt{f(t}_{1,1}\ \ldots\ \texttt{t}_{1,n_1}\texttt{)}$, corresponding to the usual terms and atomic formulas of classical first-order logic
  - For `f` being the root class `Top`, they become ***untyped single-tuple psoa terms*** $\texttt{o\#Top(t}_{1,1}\ \ldots\ \texttt{t}_{1,n_1}\ \texttt{p}_1\texttt{->v}_1\ \ldots\ \texttt{p}_k\texttt{->v}_k\texttt{)}$. These can be further specialized:
    - For `k = 0`, they become ***untyped single-tuple shelf terms*** $\texttt{o\#Top(t}_{1,1}\ \ldots\ \texttt{t}_{1,n_1}\texttt{)}$ describing object `o` with positional arguments $\texttt{t}_{1,1}$, ..., $\texttt{t}_{1,n_1}$

Presentation Syntax: Condition Language

Example (PSOA RuleML conditions)

This example shows conditions that are composed of psoa terms (`"Opticks"` is shortcut for `"Opticks"^^xs:string`).

```
Prefix(bks  <http://eg.com/books#>)
Prefix(auth <http://eg.com/authors#>)
Prefix(cts  <http://eg.com/cities#>)
Prefix(cpt  <http://eg.com/concepts#>)
```

Formula that uses an anonymous psoa (positional term):

```
?#cpt:book(auth:Newton "Opticks")
```

Deobjectified version:

```
cpt:book(auth:Newton "Opticks")
```

Formula that uses an anonymous psoa (slotted term):

```
?#cpt:book(cpt:author->auth:Newton cpt:title->"Opticks")
```

Deobjectified version:

```
cpt:book(cpt:author->auth:Newton cpt:title->"Opticks")
```

Formula that uses a named psoa (typed frame):

```
bks:opt1#cpt:book(cpt:author->auth:Newton cpt:title->"Opticks")
```

Formula that uses a named psoa (untyped frame):

```
bks:opt1#Top(cpt:author->auth:Newton cpt:title->"Opticks")
```

Deobjectified version of a formula that uses an anonymous psoa (multi-tuple term):

```
cpt:book([auth:Newton "Opticks"] [cts:London "1704"^^xs:integer])
```

Deobjectified version of a formula that uses an anonymous psoa (positional-slotted term):

```
cpt:book(auth:Newton "Opticks"
         cpt:place->cts:London
         cpt:year->"1704"^^xs:integer)
```

Presentation Syntax: Rule Language

Example (PSOA RuleML business rule)

Adapts business rule from POSL logistics use case. Ternary `reciship` conclusion represents `reci`procal `ship`pings, at total cost (as single positional argument), between `source` and `dest`ination (as two slotted arguments). First two premises apply 4-ary `shipment` relation that uses anonymous cargo and named cost variables as two positional arguments, as well as `reciship`'s slotted arguments (in both 'directions').
Third premise is `External`-wrapped `numeric-add` RIF-DTB built-in applied on right-hand side of equality to sum up `shipment` costs for total.
With the two facts, `?cost = ?57.0`.

```
Prefix(cpt  <http://eg.com/concepts#>)
Prefix(mus  <http://eg.com/museums#>)
Prefix(func <http://www.w3.org/2007/rif-builtin-function#>)
Prefix(xs   <http://www.w3.org/2001/XMLSchema#>)
Group (
  Forall ?cost ?cost1 ?cost2 ?A ?B (
    cpt:reciship(?cost cpt:source->?A cpt:dest->?B) :-
      And(cpt:shipment(? ?cost1 cpt:source->?A cpt:dest->?B)
          cpt:shipment(? ?cost2 cpt:source->?B cpt:dest->?A)
          ?cost = External(func:numeric-add(?cost1 ?cost2)) )
                                             )
  shipment("PC"^^xs:string "47.5"^^xs:float
           cpt:source->mus:BostonMoS cpt:dest->mus:LondonSciM)
  shipment("PDA"^^xs:string "9.5"^^xs:float
           cpt:source->mus:LondonSciM cpt:dest->mus:BostonMoS)
      )
```

Presentation Syntax: Rule Language (Cont'd)

Example (PSOA RuleML business rule, Cont'd)

The rule can be objectified as follows (`External`s are not being transformed):

```
Forall ?cost ?cost1 ?cost2 ?A ?B ?2 ?3 (
  Exists ?1 (?1#cpt:reciship(?cost cpt:source->?A cpt:dest->?B)) :-
    And(?2#cpt:shipment(? ?cost1 cpt:source->?A cpt:dest->?B)
        ?3#cpt:shipment(? ?cost2 cpt:source->?B cpt:dest->?A)
        ?cost = External(func:numeric-add(?cost1 ?cost2)) )
                                             )
```

Further, it can be tupributed and slotributed (actually done by the semantics):

```
Forall ?cost ?cost1 ?cost2 ?A ?B ?2 ?3 (
  Exists ?1 (And(?1#cpt:reciship(?cost)
                 ?1#cpt:reciship(cpt:source->?A)
                 ?1#cpt:reciship(cpt:dest->?B))) :-
    And(?2#cpt:shipment(? ?cost1)
        ?2#cpt:shipment(cpt:source->?A)
        ?2#cpt:shipment(cpt:dest->?B)
        ?3#cpt:shipment(? ?cost2)
        ?3#cpt:shipment(cpt:source->?B)
        ?3#cpt:shipment(cpt:dest->?A)
        ?cost = External(func:numeric-add(?cost1 ?cost2)) )
                                             )
```

Conclusion: Semantics and Implementations

- W3C's RIF-BLD has provided a **reference semantics** for extensions, and for continued efforts, as described here
- Implementations of **RIF-BLD engines** are planned or developed, including extensions to F-logic engine **Flora 2** and POSL and RuleML engine **OO jDREW**
- Flora 2, OO jDREW, and other engines could be adapted for our PSOA RuleML semantics
- A subset of PSOA RuleML with single-tuple psoa terms has already been prototyped in OO jDREW

Conclusion: Encodings and Alignments

- Future work on psoa terms includes encoding (multi-)slots and slotribution as (multi-)tuples and tupribution
- Conversely, tuples could be encoded as multi-list values of a `tuple` slot
- Web ontologies, especially taxonomies, in OWL 2, RDF Schema, etc. could be reused for PSOA RuleML's OID type systems by alignments rooted in their classes `owl:Thing`, `rdfs:Resource`, etc. and in `Top`

Conclusion: Horn

- Further efforts concern Horn rules
- Notice introductory example is not Horn in that there is a head existential after objectification
- To address this issue, it can be modified as follows

Conclusion: Psoa Rules Made Horn

Example (Rule-extended named family frame)

Horn version of introductory example retrieves `family` frame with named OID variable in premise and uses its binding to extend that frame in conclusion (left: given; right: objectified).

```
Group (
  Forall ?Hu ?Wi ?Ch ?o (
    ?o#family(husb->?Hu
              wife->?Wi
              child->?Ch)     :-
      And(?o#family(husb->?Hu
                    wife->?Wi)
          Or(kid(?Hu ?Ch)
             kid(?Wi ?Ch))) )
  inst4#family(husb->Joe
               wife->Sue)
  kid(Sue Pete)
      )
```

```
Group (
  Forall ?Hu ?Wi ?Ch ?o ?1 ?2 (
    ?o#family(husb->?Hu
              wife->?Wi
              child->?Ch)     :-
      And(?o#family(husb->?Hu
                    wife->?Wi)
          Or(?1#kid(?Hu ?Ch)
             ?2#kid(?Wi ?Ch))) )
  inst4#family(husb->Joe
               wife->Sue)
  _1#kid(Sue Pete)
      )
```

⇝ Simpler semantics corresponding to this set of ground facts:

{*inst4#family(husb->Joe wife->Sue child->Pete)*, *_1#kid(Sue Pete)*}