The Integrated PSOA RuleML for Interoperating SQL Relations and SPARQL Graphs

Harold Boley

Faculty of Computer Science, University of New Brunswick, Canada

The 6th Atlantic Workshop on Semantics and Services (AWoSS 2015)
Faculty of Computer Science, University of New Brunswick
Fredericton, NB, Canada
December 9, 2015

Generally, relational vs. graph distinction based on two orthogonal dimensions, creating system of four quadrants

Object-relational integration achieved by permitting atom to be

- ***predicate-centered*** (without OID) or ***object-centered*** (with OID) – every OID being typed by predicate as its class –

and, orthogonally,

- predicate's arguments to be ***positional*** (sequence), ***slotted*** (bag of pairs), or ***both*** (positional-plus-slotted combination)

Introduction: Psoa Table

Atoms resulting from orthogonal system are

positional-**s**lotted, **o**bject-**a**pplicative (**psoa**)

Can be used in six ways, as shown in *psoa table* (quadrants 1. to 4. expanded by combined options 5., 6.):

	predicate-centered	object-centered
positional	**1. relationships**	2. shelves
slotted	3. pairships	**4. frames**
positional+slotted	5. relpairships	6. shelframes

Of six options, positional data widely used under names like ‘tuples’, ‘vectors’, ‘lists’, and (1D) ‘arrays’ (mostly 1.)

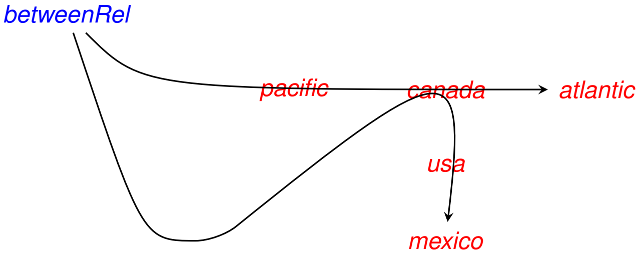
Likewise, slotted data include ‘objects’, ‘records’, ‘maps’, and ‘property lists’ (usually 4.)

Data Model: Relationships – Predicate-Centered, Positional Atoms

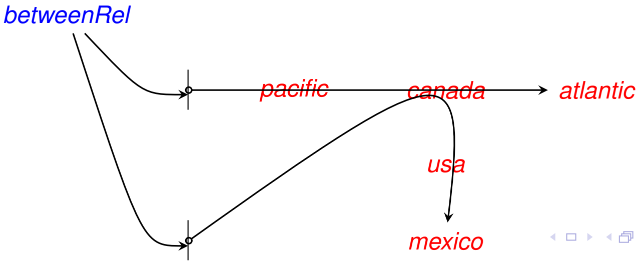
- ***Relationships*** represent n-ary positional information (n$\geq$0)
- In Grailog, relationship becomes ***directed hyperarc***, depicted as: arrow shaft starting at labelnode for relation name or at branch line, cutting through labelnodes for n-1 initial arguments in order they occur, ending with arrow head at labelnode for nth argument
- Sample Grailog figures visualize 3-ary relational betweenness with hyperarcs for two relationships applying relation name betweenRel to three argument individuals

Data Model: Relationships (Cont'd)

Grailog-style visualization syntax (without branch lines):



Grailog-style visualization syntax (with branch lines):



POSL-like presentation syntax:

betweenRel(pacific, canada, atlantic).
betweenRel(canada, usa, mexico).

RIF-like presentation syntax:

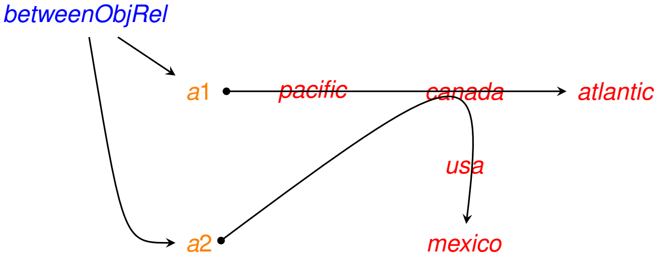
betweenRel(pacific canada atlantic)
betweenRel(canada usa mexico)

Data Model: Shelves – Object-Centered, Positional Atoms

- ***Shelves*** describe an OID with n positional arguments (n$\geq$0)
- A shelf thus endows n-tuple with OID, typed by relation/class, keeping positional representation of n-ary relationships
- Sample Grailog figure visualizes two OIDs, *a*1 and *a*2, typed by relation/class name *betweenObjRel*, and two 3-tuples with three individuals as arguments

Data Model: Shelves (Cont'd)

Grailog-style visualization syntax:



POSL-like presentation syntax:

betweenObjRel(a1^pacific, canada, atlantic).
betweenObjRel(a2^canada, usa, mexico).

RIF-like presentation syntax:

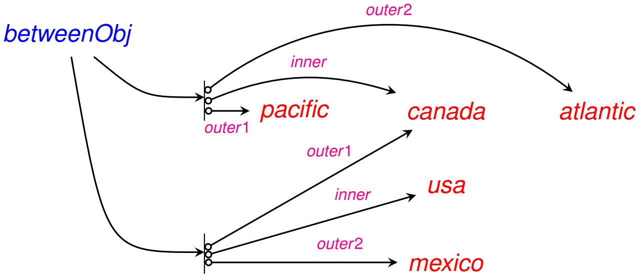
a1#betweenObjRel(pacific canada atlantic)
a2#betweenObjRel(canada usa mexico)

Data Model: Pairships – Predicate-Centered, Slotted Atoms

- ***Pairships*** apply a relation/class to n ***slots***: non-positional attribute-value pairs (n$\geq$0)
- In Grailog, **pairship is depicted as relation/class node pointing, with unary hyperarc, to branch line having n outgoing circle-shaft slot arrows, using: label for attribute, target node for value**
- Sample Grailog figure visualizes 3-slot betweenness of two pairships that apply relation name *betweenObj* to branch line for three slots, with labels *outer1*, *inner*, and *outer2*, targeting three individuals as values

Data Model: Pairships (Cont'd)

Grailog-style visualization syntax:



POSL-like presentation syntax:

betweenObj(outer1->pacific; inner->canada; outer2->atlantic).
betweenObj(outer1->canada; inner->usa; outer2->mexico).

RIF-like presentation syntax:

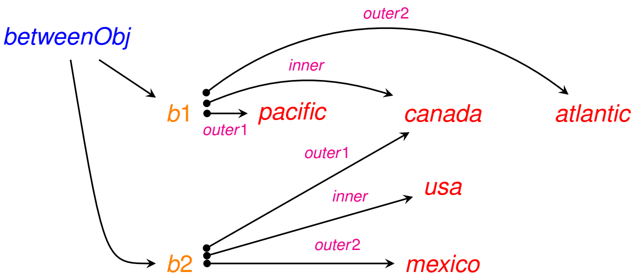
betweenObj(outer1->pacific inner->canada outer2->atlantic)
betweenObj(outer1->canada inner->usa outer2->mexico)

Data Model: Frames – Object-Centered, Slotted Atoms

- ***Frames*** describe OID using n non-positional attribute-value pairs or ***slots*** (n$\geq$0), with kind of object becoming OID-typing class name
- In Grailog, frame is depicted as **typing relation/class node pointing, with unary hyperarc, to central OID node having n outgoing bullet-shaft slot arrows, using: label for attribute, target node for value**
- Sample Grailog figure visualizes object-centered 3-slot betweenness with central nodes, *b*1 and *b*2, for OIDs of two frames typed by relation name *betweenObj*, and three slots, with labels *outer*1, *inner*, and *outer*2, targeting three individuals as values

Data Model: Frames (Cont'd)

Grailog-style visualization syntax:



POSL-like presentation syntax:

betweenObj(b1ˆouter1->pacific; inner->canada; outer2->atlantic).
betweenObj(b2ˆouter1->canada; inner->usa; outer2->mexico).

RIF-like presentation syntax:

b1#betweenObj(outer1->pacific inner->canada outer2->atlantic)
b2#betweenObj(outer1->canada inner->usa outer2->mexico)

SQL-PSOA-SPARQL: Interop Use Case

- Suppose you are working on project using **SQL queries over relational data** and then proceeding to **SPARQL queries over graph data** to be used as metadata repository
- Or, vice versa, on project complementing SPARQL with SQL for querying evolving mass-data store
- Or, on project using SQL and SPARQL from the beginning
- In all of these projects, **object-relational interoperability issues** may arise
- Hence use case on **bidirectional SQL-PSOA-SPARQL transformation (schema/ontology mapping)** for interoperability
- **Core transformation between relational and object-centered paradigms is expressed in language-internal manner within PSOA RuleML itself**

SQL-PSOA-SPARQL: Interop Use Case (Cont'd)

- Use case represents **addresses as (flat) relational facts and as – subaddress-containing – (nested) object-centered facts**, as shown for Seminaris address below
  - Earlier (flat and nested) positional versions have been used to explain XML-to-XML transformation
  - Later, similar use case was employed to demonstrate SPINMap for RDF-to-RDF transformation
- **OID-conclusion direction of implication from relational to object-centered (frame) paradigm** is given as first rule below
- **OID-condition direction from object-centered (frame) to relational paradigm** is given as second rule

SQL-PSOA-SPARQL: Facts and Rules

```
addressRel("Seminaris" "Takustr. 39" "14195 Berlin")
                                             % relational fact

r1#addressObj(name->"Seminaris"              % object-centered fact
              place->r2#placeObj(street->"Takustr. 39"
                                 town->"14195 Berlin"))

Forall ?Name ?Street ?Town (                 % OID-conclusion rule
  Exists ?O1 ?O2 ( ?O1#addressObj(
                     name->?Name
                     place->?O2#placeObj(street->?Street
                                         town->?Town)) )   :-
    addressRel(?Name ?Street ?Town)
                          )

Forall ?Name ?Street ?Town ?O1 ?O2 (   % OID-condition rule
  addressRel(?Name ?Street ?Town)     :-
    ?O1#addressObj(name->?Name
                   place->?O2#placeObj(street->?Street
                                       town->?Town))
                                    )
```

SQL-PSOA-SPARQL: Relational to Object-Centered Queries

Besides directly retrieving relational fact,
OID-condition rule and object-centered fact can be used
for **derivation of relational queries** as follows
(corresponding to **RDF-to-RDB data mapping** direction):

```
addressRel("Seminaris" ?S "14195 Berlin")

  ?O1#addressObj(
            name->"Seminaris"
            place->?O2#placeObj(street->?S
                                town->"14195 Berlin"))

?S = "Takustr. 39"
```

SQL-PSOA-SPARQL: Object-Centered to Relational Queries

Besides directly retrieving object-centered fact, OID-conclusion rule and relational fact can be used for **derivation of object-centered queries** as follows (corresponding to **RDB-to-RDF data mapping** direction):

```
?O1#addressObj(name->"Seminaris"
               place->?O2#placeObj(
                              street->?S
                              town->"14195 Berlin"))

  addressRel("Seminaris" ?S "14195 Berlin")

?O1 = skolem5("Seminaris" "Takustr. 39" "14195 Berlin")
?O2 = skolem6("Seminaris" "Takustr. 39" "14195 Berlin")
?S = "Takustr. 39"
```

SQL-PSOA-SPARQL: SQL to SPARQL

- If object-centered PSOA RuleML fact is replaced by corresponding RDF triple facts, **OID-condition PSOA RuleML rule can also be used for language-internal transformation of SQL-like queries to SPARQL-like queries** as shown shortly
  - 'Neutral' column headings `Col`i, with $1 \leq i \leq 3$, are used to avoid providing slot-name-like information, thus keeping SQL purely positional
- **Paradigm-crossing translation step is done by OID-condition rule completely within PSOA RuleML**, starting at SQL queries "lifted" to PSOA and ending at SPARQL queries "dropped" from PSOA

SQL-PSOA-SPARQL: SQL to SPARQL (Cont'd)

```
SELECT * FROM addressRel                                                    -- SQL
WHERE Col1='Seminaris'

  addressRel("Seminaris" ?S ?T)                                             % PSOA

  ?O1#addressObj(name->"Seminaris"                                          % PSOA
                 place->?O2#placeObj(street->?S
                                     town->?T))

  SELECT ?S ?T                                                            # SPARQL
  WHERE {?O1 rdf:type addressObj. ?O1 name "Seminaris".
                                  ?O1 place ?O2.
         ?O2 rdf:type placeObj. ?O2 street ?S.
                                ?O2 town ?T.}

?S = "Takustr. 39"
?T = "14195 Berlin"
```

SQL-PSOA-SPARQL: SPARQL to SQL

- If relational PSOA RuleML fact is replaced by corresponding SQL table row, **OID-conclusion PSOA RuleML rule can be used for language-internal transformation of SPARQL-like queries to SQL-like queries** as shown shortly
- **Paradigm-crossing translation step is done by OID-conclusion rule completely within PSOA RuleML**

```
SELECT ?S ?T                                                      # SPARQL
WHERE {?O1 rdf:type addressObj. ?O1 name "Seminaris".
                                ?O1 place ?O2.
       ?O2 rdf:type placeObj. ?O2 street ?S.
                              ?O2 town ?T.}

  ?O1#addressObj(name->"Seminaris"                                  % PSOA
                 place->?O2#placeObj(street->?S
                                     town->?T))

  addressRel("Seminaris" ?S ?T)                                     % PSOA

  SELECT * FROM addressRel                                           -- SQL
  WHERE Col1='Seminaris'

?S = "Takustr. 39"
?T = "14195 Berlin"
```