RDF and XML: Towards a Unified Query Layer

Nuno Lopes, Stefan Bischof, Orri Erling, Axel Polleres, Alexandre Passant, Diego Berrueta, Antonio Campos, Jérôme Euzenat, Kingsley Idehen, Stefan Decker, Stéphane Corlosquet, Jacek Kopecky, Janne Saarela, Thomas Krennwallner, Davide Palmisano, and Michal Zaremba
(supporters of the W3C member submission)
Semantic Data Access…
… that would be the vision

- RDF: common simple Data format underlying the Semantic Web

Queries (SPARQL1.1)

Rules (RIF)

Ontologies (OWL2)

<table>
<thead>
<tr>
<th>i</th>
<th>FNAM</th>
<th>LNAM</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bob</td>
<td>McBob</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>Johnson</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>Steve</td>
<td>Smith</td>
<td>38</td>
</tr>
</tbody>
</table>

```
<s1:messages>
  <s1:msg id="m1" format="bin">
    <s1:sendername>3PO</s1:sendername>
    <s1:receivername>R2D2</s1:receivername>
    <s1:payload>0111010001</s1:payload>
  </s1:msg>
  <s1:msg id="m2" format="text">
    <s1:sendername>Obiwan</s1:sendername>
    <s1:receivername>Luke</s1:receivername>
    <s1:payload>Use the force</s1:payload>
  </s1:msg>
</s1:messages>
```
Semantic Data Access…
… that would be the vision

Queries

<emp:#1> a <emp:Employee>
<emp:#1> <name> “Bob McBob”.
...
<s1:3PO> a <s1:Agent>
<s1:3PO> <name> “3PO”.
<s1:3PO> <sent> <m1> . <m1> <format> <binary>.
<s1:Obiwan> a <s1:Agent>
<s1:Obiwan> <name> “Obiwan Kenobi”.
<s1:Obiwan> <sent> <m2> . <m2> <format> <text>.
...
<axel> a <Person>
<axel> <name> “Axel Polleres”.

Rules

Ontologies

<table>
<thead>
<tr>
<th>id</th>
<th>FNAM</th>
<th>LNAM</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bob</td>
<td>McBob</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>Johnson</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>Steve</td>
<td>Smith</td>
<td>38</td>
</tr>
</tbody>
</table>

Semantic Data Access… that would be the vision
Semantic Data Access...  
... that would be the vision

**Queries (SPARQL)**

```
SELECT ?X WHERE { ?X a <Person> }
```

**Ontologies (OWL2, DL)**:  
"<emp:Employee> is a subclass of <Person>"

\( \text{Employee} \sqsubseteq \text{Person} \)

**Rules (RIF, LP)**:  
"<s1:Agents> who never sent binary are Persons"

\[ \text{Person}(x) : - \text{Agent}(x), \text{not} \ \text{sent}(x, \text{"binary"}) \]
Bridging Gaps to the data “below”:

Can’t we just query “in one go”?

**Queries**

- Ontologies (OWL)
- Rules (RIF)
- SQL (RDB2RDF)
- XQuery/XSLT (GRDDL)
- XML
- WWW

**Digital Enterprise Research Institute**

www.deri.ie
Our starting motivation (2008):

SOAP/WSDL

RSS

HTML

SPARQL

XSPARQL

XSLT/XQuery

<XML/>
Example:

```xml
<relations>
  <person name="Alice">
    <knows>Bob</knows>
    <knows>Charles</knows>
  </person>
  <person name="Bob">
    <knows>Charles</knows>
  </person>
  <person name="Charles"/>
</relations>
```

Lowering:
```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:b1 a foaf:Person;
  foaf:name "Alice";
  foaf:knows _:b2;
  foaf:knows _:b3.
_:b2 a foaf:Person; foaf:name "Bob";
  foaf:knows _:b3.
_:b3 a foaf:Person; foaf:name "Charles".
```

Lifting:
So, why are XSLT, XQuery not enough?

Because RDF ≠ RDF/XML !!!

1) many different RDF/XML representations...

2) ... and actually a lot of RDF data residing in RDF stores, accessible via SPARQL endpoints already, rather than in RDF/XML
Our approach: XSPARQL (W3C submission)

- New query language… but don’t reinvent!

**XQuery + SPARQL = XSPARQL**

| Prolog: | P | declare namespace prefix="namespace-URI"
| | or prefix prefix: <namespace-URI> |
| Body: | F | for var in XPath-expression |
| | L | let var := XPath-expression |
| | W | where XPath-expression |
| | O | order by expression |
| | F' | for varlist |
| | D | from/from named <dataset-URI> |
| | W | where {pattern } |
| | M | order by expression |
| | | limit integer > 0 |
| | | offset integer > 0 |
| Head: | C | construct |
| | | { template (with nested XSPARQL) } |
| | R | return XML + nested XSPARQL |
Example: Mapping from RDF to XML

```xml
<relations>
{ for $Person $Name
  from <relations.rdf>
  where { $Person foaf:name $Name }
  order by $Name
  return
    <person name="{$Name}">
    {for $FName
      from <relations.rdf>
      where {
        $Person foaf:knows $Friend .
        $Person foaf:name $Name .
        $Friend foaf:name $Fname }
      return <knows>{$FName}</knows>
    } </person>
  }
</relations>
```

```xml
<relations>
  <person name="Alice">
    <knows>Bob</knows>
    <knows>Charles</knows>
  </person>
  <person name="Bob">
    <knows>Charles</knows>
  </person>
  <person name="Charles"/>
</relations>
```
Example: Adding value generating functions to SPARQL

```
construct { :me foaf:knows _:b .
  _:b foaf:name {fn:concat(""",?N," ",?F,""""})  }
from <MyAddrBookVCard.rdf>
where {
}

...:
:me foaf:knows _:b1. _:b1 foaf:name “Peter Patel-Schneider” .
:me foaf:knows _:b2. _:b2 foaf:name “Stefan Decker” .
:me foaf:knows _:b3. _:b3 foaf:name “Thomas Eiter” .
...
Implementation and semantics:

- **Formal Semantics (XSPARQL1.0):**
  - Based on XQuery formal Semantics
  - Can be implemented based on rewriting to XQuery

- **Challenges/Limitations:**
  - Nesting, scope of RDF dataset...
  - different “type systems” of RDF/XML (sequences)
  - adding ontological inference (to resolve heterogeneities)
  - We are working on this in XSPARQL1.1!
Formal Semantics:

- Initial idea (and formalised in XSPARQL1.0):
  - extension of the XQuery semantics by plugging in SPARQL semantics in a modular way

- Rewriting algorithm is defined for embedding XSPARQL into native XQuery plus interleaved calls to a SPARQL endpoint
Rewriting XSPARQL to XQuery...

```
construct { _:b foaf:name {fn:concat(""",$N," ","",$F,""""}) } 
from <vcard.rdf>
where { $P vc:Given $N . $P vc:Family $F . }
```

    fn:encode-for-uri(
        "select $P $N $F from <vcard.rdf>
        where {${$P vc:Given $N . $P vc:Family $F.}}")
    ))

for $aux_result at $aux_result_pos
    in doc($aux_query)//sparql_result:result

let $P_Node := $aux_result/sparql_result:binding[@name="P"]
let $N_Node := $aux_result/sparql_result:binding[@name="N"]
let $F_Node := $aux_result/sparql_result:binding[@name="F"]

let $N := data($N_Node/*)
let $N_NodeType := name($N_Node)
let $N_RDFTerm := local:rdf_term($N_NodeType,$N)

```
1. Encode SPARQL in HTTP call SELECT Query
2. Execute call, via fn:doc function
3. Collect results from SPARQL result format(XML)
4. construct becomes return that outputs triples.
```
Current formalisation embeds rewriting in the functional semantics of XQuery:

http://xsparql.deri.org/spec/xsparql-semantics.html#id:flwor-expressions

mapping rules $\text{Expr}'$ inherit from the definitions of XQuery's $\text{Expr}$
Ongoing work/Outlook:

- Simple rewriting semantics has some limitations, which we are currently working on:
  - Adding RDFS/OWL entailment (and other SPARQL1.1 features)
  - Integrate RDB Querying (SQL), Json, etc.
  - Optimisations…
  - Nesting, scope of RDF dataset…
  - Different “type systems” of RDF/XML (sequences)…

![Diagram showing integrated engine(s)]
Extract foaf:knows relations from a RDB with two tables: containing persons and their knows relations

prefix foaf: <http://xmlns.com/foaf/0.1/>

for p1.name as $x, p2.name as $y
from person as p1, person as p2, relation as r
where { p1.name = r.person and
         p2.name = r.knows }

construct { $x foaf:knows $y }

Use as an RDB2RDF exporter…

Unified RDF/XML/RDB Query Layer = XSPARQL + RDB2RDF?
Optimisations – Example:

- E.g. dependent Join... i.e.

```xml
<relations>
{ for $Person $Name
    from <relations.rdf>
    where { $Person foaf:name $Name }
    order by $Name
    return
    <person name="{$Name}">
        {for $FName
            where {
                $Person foaf:knows $Friend .
                $Friend foaf:name $Fname }
            return <knows>{$FName}</knows>
        }
    </person>
}</relations>
```

```xml
<relations>
{ let $aux := select $Person $Name $FName
    from <relations.rdf>
    where { $Person foaf:name $Name .
        $Person foaf:knows $Friend .
        $Friend foaf:name $Fname }
    for $Name in  $aux.Name
    return
    <person name="{$Name}">
        { for $FName in $aux.Fname
            where $aux.Name = $Name
            return <knows>{$FName}</knows>
        }
    </person>
}</relations>
```

Only one SPARQL query
Enable optimisation across layers

query each source format in native language instead of multistep transformation via “narrow“ interfaces (e.g. SPARQL → SPARQL-Result/XML → XQuery/XSLT)

Enable declarative view (a la Relational Algebra for core fragment of the language)

Tighter integration of various source formats that populate the (Semantic) Web

This needs a cross-activity effort in W3C? Semantic Web + XML + others?

One Unified RDF/XML/RDB Query Layer to combine XQuery+SPARQL+GRDDL+SQL?
Remember the query from before…. We were slightly cheating:

```xml
<relations>
{ for $Person $Name
  from <relations.rdf>
  where { $Person foaf:name $Name } 
  order by $Name 
  return
  <person name="{$Name}">
  {for $$FName
  from <relations.rdf>
  where {
    $Person foaf:knows $Friend .
    $Person foaf:name $Name .
    $Friend foaf:name $$Fname }
  return <knows>{$FName}</knows>
  } </person>
} </relations>
```
Remember the query from before…. This is what one would rather expect

\[
\text{<relations>}
\begin{align*}
\{ & \text{for } $Person \text{ $Name}
\text{ from } <\text{relations.rdf>}
\text{ where } \{ $Person \text{ foaf:name } $Name \} \\
\text{ order by } $Name
\text{ return }
\text{<person name="{$Name}"} \\
\text{ {for } $FName}
\text{ where } \{
$Person \text{ foaf:knows } $Friend . \\
$Friend \text{ foaf:name } $Fname \}
\text{ return } <\text{knows>}{ $FName } </\text{knows>}
\} </\text{person>}
\}\text{<relations>}
\]

- The nested query should be over the same Dataset as the outer query, bindings to bnodes should be preserved
- Two separate, independent SPARQL calls don’t work anymore

**Solution:**
- We need to add Dataset to the dynamic environment in the semantics.
- We need a special SPARQL implementation that allows several calls to the same active graph.
Different “type systems” of RDF/XML (e.g. sequences)...

- Social Graph queries a la [1]:

  Give me all pairs of co-authors and their joint publications.

  ```sparql
  prefix foaf: <http://xmlns.com/foaf/0.1/>
  prefix dc:     <http://purl.org/dc/elements/1.1/>

  let $ds := for *
  from <http://dblp.l3s.de/d2r/resource/authors/Axel_Polleres>
  where { $pub dc:creator [] } 
  construct {
    { for * from $pub where { $p dc:creator $o . }
      construct {$p dc:creator <{$o}>} }
  }

  let $allauthors :=
  distinct-values(for $o from $ds where {p dc:creator $o})
  order by $o
  return $o

  for $auth at $auth_pos in $allauthors
  for $coauth in $allauthors[position() > $auth_pos]
  let $commonPubs := count(
    { for $pub from $ds where { $pub dc:creator $auth, $coauth } return $pub }
  )
  where ($commonPubs > 0)
  construct { [ :author1 $auth; :author2 $coauth; :commonPubs $commonPubs ] }
  ```

1. Mauro San Martín, Claudio Gutierrez: Representing, Querying and Transforming Social Networks with RDF/SPARQL. ESWC 2009: 293-307
Use: SIOC-2-RSS

- XSPARQL+SIOC enables customised RSS export:

```
<channel>
<title>
  {for $name
  where { [a sioc:Forum] sioc:name $name }
  return $name}
</title>
{for $seeAlso
  where { [a sioc:Forum] sioc:container_of [rdfs:seeAlso $seeAlso] }
  return <item>
    {for $title $descr $date
      from $seeAlso
      where { [a sioc:Post] dc:title $title ;
        sioc:content $descr;
        dcterms:created $date }
      return <title>$title</title>
      <description>$descr</description>
      <pubDate>$date</pubDate>}
</item>
```

“Great stuff,... I have not seen any SIOC to RSS xslt examples or vice versa” (John Breslin, creator of SIOC)
Use: Generate KML from FOAF+Geo:

- XSPARQL+FOAF and GEO data enables KML map data:

```
<kml xmlns="http://www.opengis.net/kml/2.2">
  {
    for $person $name $long $lat
    from <http://www.deri.ie/Stefan_Decker>
    where { $person; foaf:name $name;
      foaf:based_near [ geo:long $long; geo:lat $lat ] }
    return <Placemark>
      <name>{fn:concat("Location of ", $name)}</name>
      <Point>
        <coordinates>{fn:concat($long, ", ", $lat, ", ,0")}</coordinates>
      </Point>
    </Placemark>
  }
</kml>
```