XSPARQL

Traveling between the XML and RDF Worlds – and Avoiding the XSLT Pilgrimage

Waseem Akhtar\textsuperscript{1} \hspace{1em} Jacek Kopecký\textsuperscript{3}
Thomas Krennwallner\textsuperscript{1,2} \hspace{1em} Axel Polleres\textsuperscript{1}

\textsuperscript{1}Digital Enterprise Research Institute, National University of Ireland, Galway
\textsuperscript{2}Knowledge-Based Systems Group, Institute for Information Systems, TU Wien
\textsuperscript{3}STI Innsbruck, University of Innsbruck, Austria
Alice, Bob, and Charles

knows

knows

knows

knows
Motivation

relations.xml

relations.rdf

relations

person

name

knows

Bob

Alice

Charles

person

name

knows

Bob

Alice

Charles

person

name

knows

Bob

Alice

Charles

relations

relations.rdf

@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".

relations.xml

relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type

_relations rdf type
Motivation

```
relations.xml

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

relations.rdf

@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".
```
Motivation

- Two standards for different types of data: XML and RDF
- XML data model differs from RDF data model
  - XML and RDF query languages have different objectives
    ... but not so different at all.
Mapping between XML and RDF

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>XML</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td></td>
<td>XSLT, XQuery</td>
<td></td>
</tr>
<tr>
<td>RDF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDF + XML</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Solved
## Mapping between XML and RDF

A table summarizes the mapping processes:

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>XML</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery</td>
<td></td>
</tr>
<tr>
<td>RDF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDF + XML</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The process is described as **tedious**.
## Mapping between XML and RDF

![Diagram showing mapping from XML to RDF](image)

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>XML</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery</td>
<td>?</td>
</tr>
<tr>
<td>RDF</td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery</td>
<td>?</td>
</tr>
<tr>
<td>RDF + XML</td>
<td></td>
<td>XSLT, XQuery</td>
<td>?</td>
</tr>
</tbody>
</table>

**Issue:** ambiguous XML representation of RDF
Mapping between XML and RDF

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>XML</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td></td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery</td>
</tr>
<tr>
<td>RDF</td>
<td></td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery</td>
</tr>
<tr>
<td>RDF + XML</td>
<td></td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery</td>
</tr>
</tbody>
</table>

Issue: extract the whole RDF store as XML
Mapping between XML and RDF

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>XML</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery ?</td>
<td></td>
</tr>
<tr>
<td>RDF</td>
<td>XSLT, XQuery ?</td>
<td>XSLT, XQuery ?</td>
<td></td>
</tr>
<tr>
<td>RDF + XML</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Issue: higher entailment regime
## Mapping between XML and RDF

**Diagram:**

- From XML to RDF
- From RDF to XML

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>XML</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>XML</td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery</td>
</tr>
<tr>
<td>RDF</td>
<td>RDF</td>
<td>XSLT, XQuery</td>
<td>SPARQL</td>
</tr>
<tr>
<td>RDF + XML</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Mapping between XML and RDF

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>XML</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>XML</td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery</td>
</tr>
<tr>
<td>RDF</td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery</td>
<td>SPARQL</td>
</tr>
<tr>
<td>RDF + XML</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

construct { _:b foaf:name ?FN }
from <vcard.rdf>
where { ?P vc:FN ?FN }
Mapping between XML and RDF

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>XML</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery</td>
</tr>
<tr>
<td>RDF</td>
<td>XSLT, XQuery ?</td>
<td></td>
<td>SPARQL ?</td>
</tr>
<tr>
<td>RDF + XML</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
construct { _:b foaf:name ?? }
from <vcard.rdf>
```
Mapping between XML and RDF

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>XML</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery</td>
</tr>
<tr>
<td>RDF</td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery</td>
<td>SPARQL</td>
</tr>
<tr>
<td>RDF + XML</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

construct { _:b foaf:name ?FN }
from <vcard.rdf>
filter(?FN = concat(""",?N," ",?F,"""")) }
Mapping between XML and RDF

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>XML</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>XML</td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery</td>
</tr>
<tr>
<td>RDF</td>
<td>RDF</td>
<td>XSLT, XQuery</td>
<td>SPARQL</td>
</tr>
<tr>
<td>RDF + XML</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
construct { _:b foaf:name fn:concat("","",?N," ",?F,"""") }
from <vcard.rdf>
```
Mapping between XML and RDF

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>XML</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td></td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery ?</td>
</tr>
<tr>
<td>RDF</td>
<td></td>
<td>XSLT, XQuery ?</td>
<td>SPARQL ?</td>
</tr>
<tr>
<td>RDF + XML</td>
<td></td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Mapping between XML and RDF

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>XML</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>to XML</td>
<td>XSLT, XQuery</td>
<td>XSLT, XQuery</td>
</tr>
<tr>
<td>RDF + XML</td>
<td>to RDF</td>
<td>?</td>
<td>SPARQL ?</td>
</tr>
</tbody>
</table>
### XQuery and SPARQL – A comparison

#### Schematic view on XQuery

<table>
<thead>
<tr>
<th>Prolog:</th>
<th>P</th>
<th>declare namespace prefix=&quot;namespace-URI&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body:</td>
<td>F</td>
<td>for var in XPath-expression</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>let var := XPath-expression</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>where XPath-expression</td>
</tr>
<tr>
<td></td>
<td>O</td>
<td>order by XPath-expression</td>
</tr>
<tr>
<td>Head:</td>
<td>R</td>
<td>return XML + nested XQuery</td>
</tr>
</tbody>
</table>

#### Schematic view on SPARQL

<table>
<thead>
<tr>
<th>Prolog:</th>
<th>P</th>
<th>prefix prefix: &lt;namespace-URI&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head:</td>
<td>C</td>
<td>construct { template }</td>
</tr>
<tr>
<td>Body:</td>
<td>D</td>
<td>from / from named &lt;dataset-URI&gt;</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>where { pattern }</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>order by expression</td>
</tr>
<tr>
<td></td>
<td></td>
<td>limit integer &gt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>offset integer &gt; 0</td>
</tr>
</tbody>
</table>
### XSPARQL: Combining XQuery with SPARQL

| 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 |
Mapping RDF to RDF

Generate fullname from first and last name:

\[
\text{construct} \ \{ \ _:b \ \text{foaf:name} \ \{ \text{fn:concat}("","$N"," ","$F",""") \} \ \} \\
\text{from} \ \text{<vcard.rdf>}
\]

\text{where} \ \{ \\
\quad \ P \ \text{vc:Given} \ \ N \ . \\
\quad \ P \ \text{vc:Family} \ \ F \ .
\}

Mapping RDF to RDF

Generate fullname from first and last name:

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

_:b1 foaf:name "Waseem Akhtar"
_:b2 foaf:name "Jacek Kopecky"
_:b3 foaf:name "Axel Polleres"
Mapping RDF to 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>

<relations>
  <person name="Alice">
    <knows>Bob</knows>
    <knows>Charles</knows>
  </person>
  <person name="Bob">
    <knows>Charles</knows>
  </person>
  <person name="Charles"/>
</relations>
Mapping RDF to 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>

<relations>
  <person name="Alice">
    <knows>Bob</knows>
    <knows>Charles</knows>
  </person>
  <person name="Bob">
    <knows>Charles</knows>
  </person>
  <person name="Charles"/>
</relations>
Mapping RDF to 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>

<relations>
<person name="Alice">
  <knows>Bob</knows>
  <knows>Charles</knows>
</person>
<person name="Bob">
  <knows>Charles</knows>
</person>
<person name="Charles"/>
</relations>
Mapping RDF to 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>

<relations>
  <person name="Alice">
    <knows>Bob</knows>
    <knows>Charles</knows>
  </person>
  <person name="Bob">
    <knows>Charles</knows>
  </person>
  <person name="Charles"/>
</relations>
Mapping RDF to 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>

<relations>
<person name="Alice">
  <knows>Bob</knows>
  <knows>Charles</knows>
</person>
<person name="Bob">
  <knows>Charles</knows>
</person>
<person name="Charles"/>
</relations>
Mapping RDF to 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>

<relations>
<person name="Alice">
  <knows>Bob</knows>
  <knows>Charles</knows>
</person>
<person name="Bob">
  <knows>Charles</knows>
</person>
<person name="Charles"/>
</relations>
XSPARQL Semantics + Implementation

- Formal semantics of XSPARQL: extension of the XQuery semantics by plugging in SPARQL semantics in a modular way
XSPARQL Semantics + Implementation

- Formal semantics of XSPARQL: extension of the XQuery semantics by plugging in SPARQL semantics in a modular way

Diagram:

1. XML or RDF
2. XSPARQL
3. Query Rewriter
4. SPARQL Engine
5. XQuery Engine
6. XML or RDF
XSPARQL Semantics + Implementation

- Formal semantics of XSPARQL: 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
XSPARQL Semantics + Implementation

- Formal semantics of XSPARQL: 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

- Benefits: rely on off-the-shelf components
XSPARQL Semantics + Implementation

- Formal semantics of XSPARQL: 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

- Benefits: rely on off-the-shelf components

- Ongoing work: Optimization
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)

return ( fn:concat("b",$aux_result_pos," foaf:name ",
fn:concat("","",$N_RDFTerm," ",F_RDFTerm,""") ) )
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.)}")
    )
```
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).

return ( fn:concat(":_:b",$aux_result_pos," foaf:name "),
(fn:concat(""",$N_RDFTerm," ",F_RDFTerm,"""")))

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)
...
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)

return ( fn:concat("&_b","$aux_result_pos," foaf:name "),
    ( fn:concat(""",$N_RDFTerm," ",F_RDFTerm,"""")) ), "." )
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,$N)
. . .
return ( fn:concat(":_b",$aux_result_pos," foaf:name "),
  ( fn:concat("""",$N_RDFTerm," ","F_RDFTerm,"""""") ), "." )
Conclusion and Outlook

- Querying XML and RDF at one shot
- Producing XML or RDF from XML and/or RDF
- Extending SPARQL by expressive XQuery constructs
- Extending XQuery by SPARQL graph patterns

Ongoing work:
- Optimization
- extend XSPARQL with nested FLWOR in \texttt{where} part
- XSPARUL

http://xsparql.deri.org/