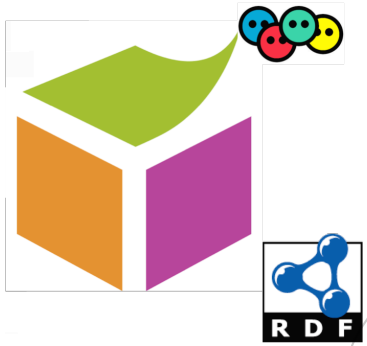
On the Semantics of Heterogeneous Querying of Relational, XML and RDF Data with XSPARQL

Nuno Lopes, Stefan Bischof, Stefan Decker, Axel Polleres

Integration of Heterogeneous Sources

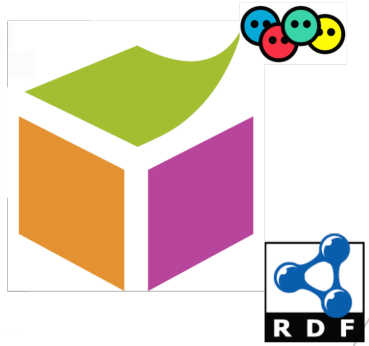
Integration of Heterogeneous Sources

Integration of Heterogeneous Sources



Integration of Heterogeneous Sources

XSLT | XQuery

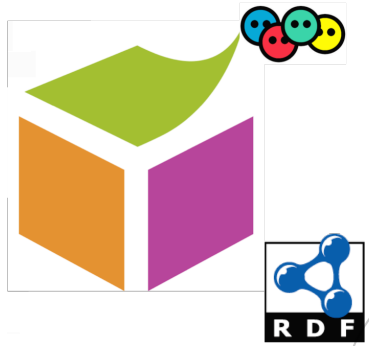


SPARQL

SQL

Integration of Heterogeneous Sources

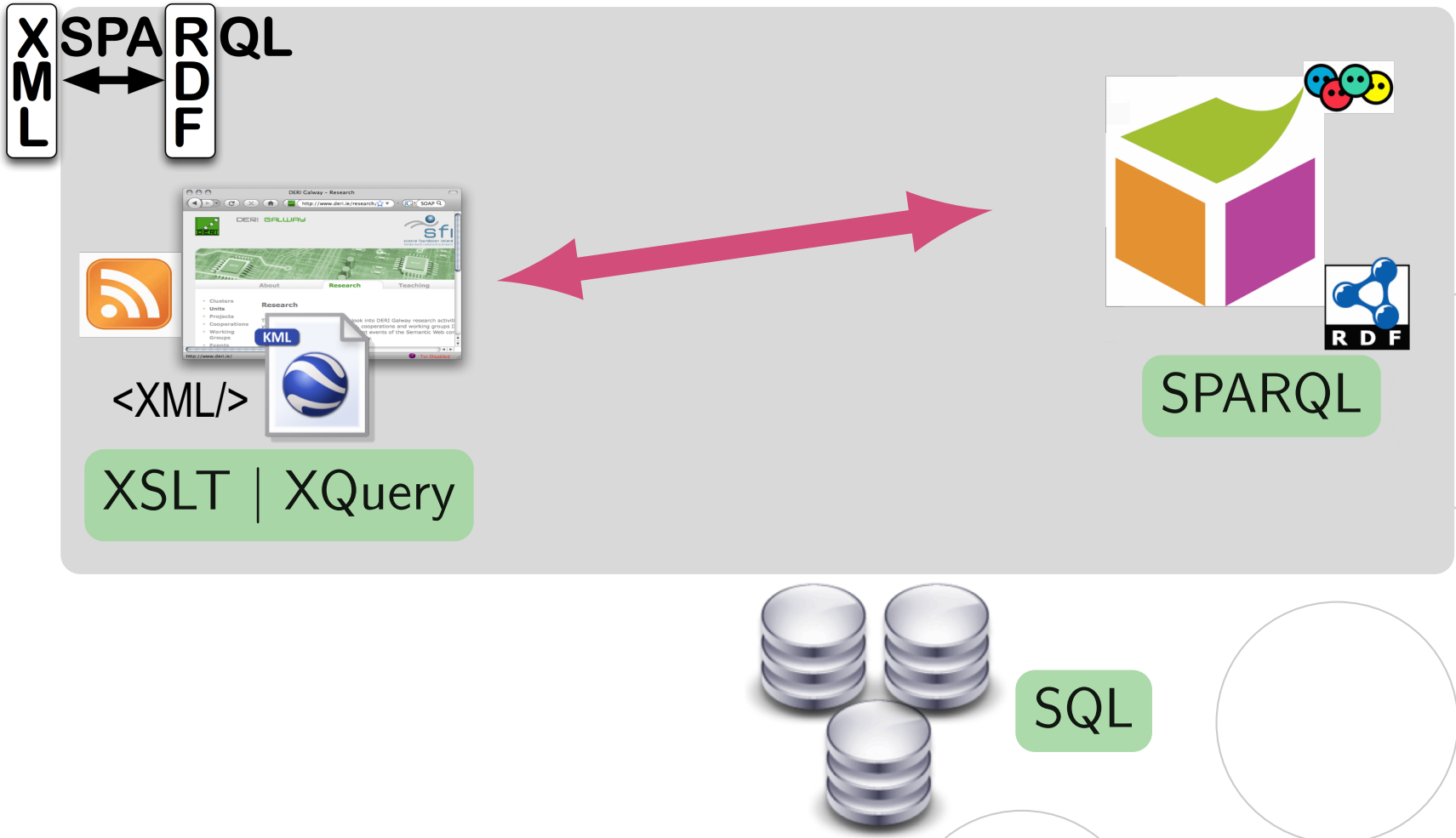
XSLT | XQuery



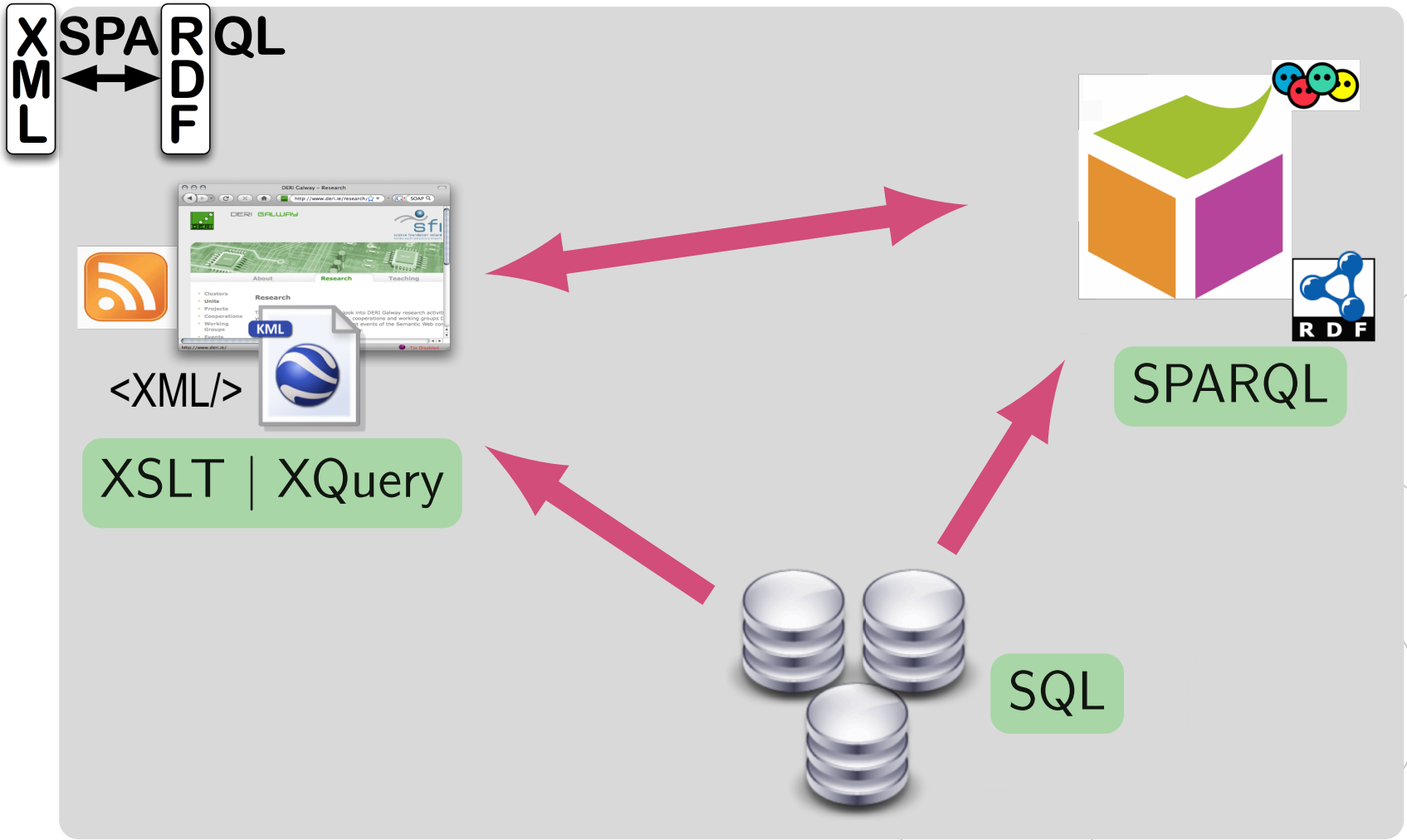
SPARQL

SQL

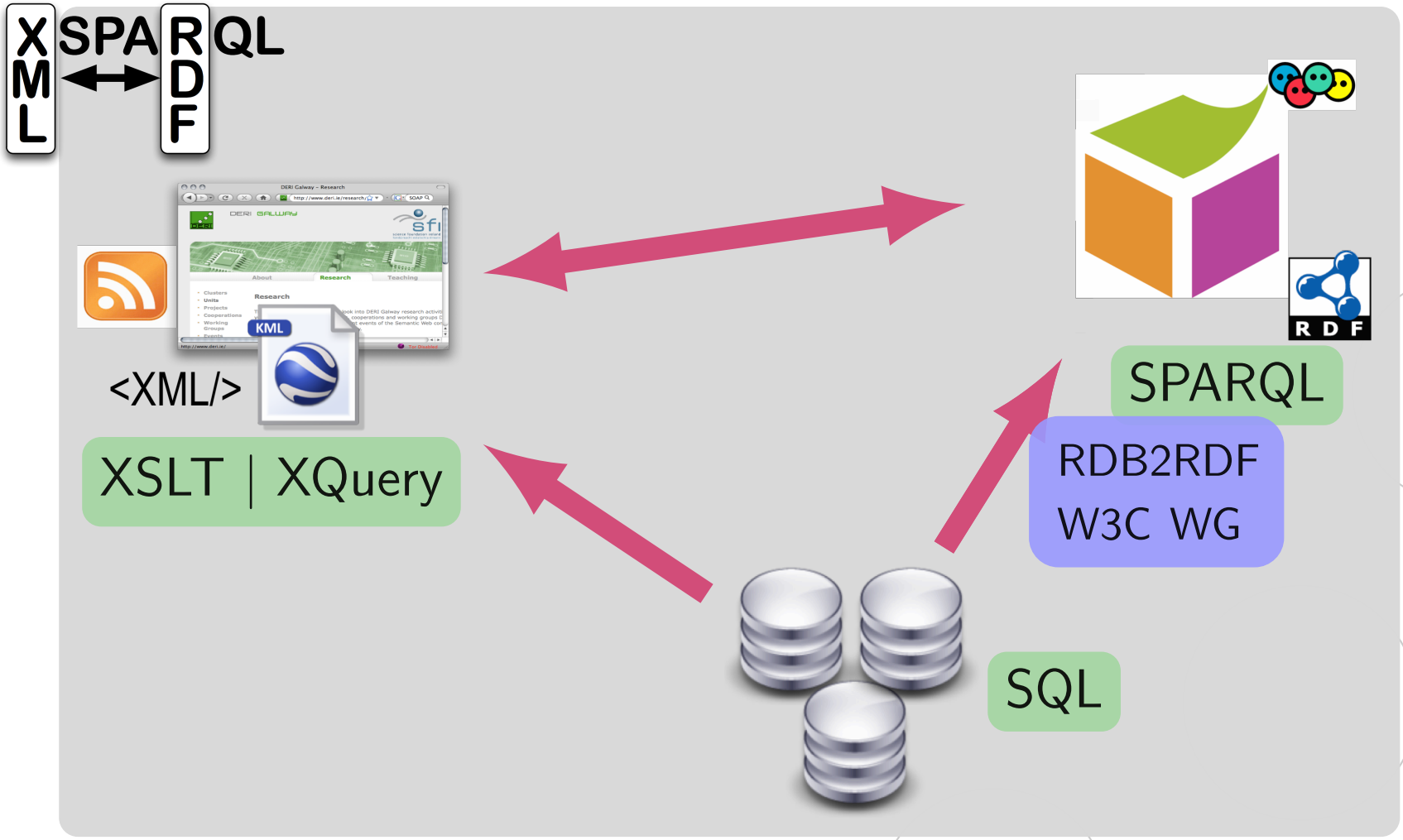
Integration of Heterogeneous Sources



Integration of Heterogeneous Sources



Integration of Heterogeneous Sources



RDB schema and use case

person

ssn	name	*memberOf*
123	“Bono”	1

band

id	name	origin
1	“U2”	“Dublin”

RDB schema and use case

person

ssn	name	*memberOf*
123	“Bono”	1

band

id	name	origin
1	“U2”	“Dublin”

?

RDB schema and use case

person

ssn	name	*memberOf*
123	"Bono"	1

band

id	name	origin
1	"U2"	"Dublin"

RDB schema and use case

person

ssn	name	*memberOf*
123	"Bono"	1

band

id	name	origin
1	"U2"	"Dublin"

```
<Placemark>
  <name>Hometown of U2</name>
  <Point>
    <coordinates>-6.259722232818604,
                 53.3477783203125,0
    </coordinates>
  </Point>
</Placemark>
```

This talk

Overview

Background
- RDF
- XQuery
- SPARQL

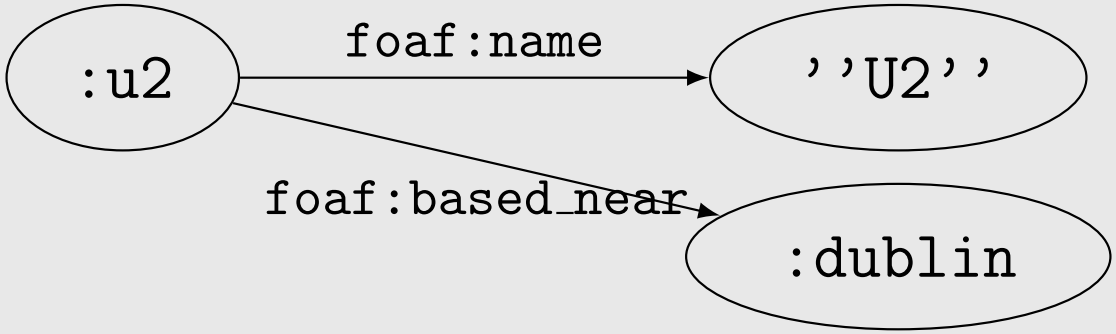
RDB in XSPARQL

RDB2RDF

Conclusions

RDF simple example

band.rdf

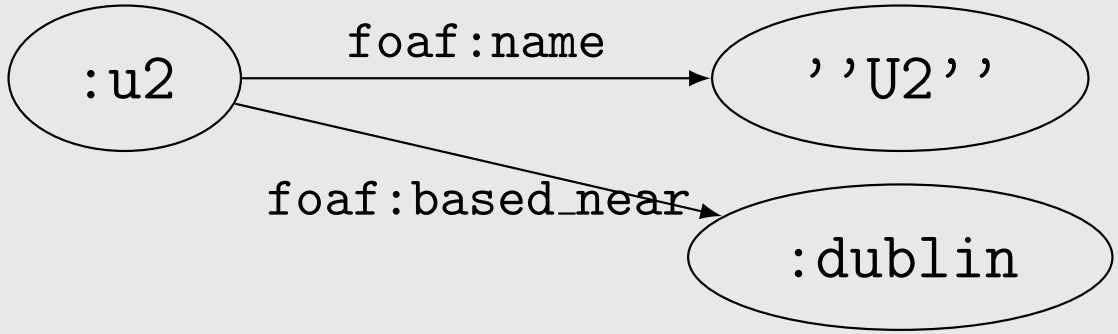


RDF: Resource Description Framework

- Disjoint sets of URI $\mathcal{U}$, blank nodes $\mathcal{B}$, and literals $\mathcal{L}$
- *RDF triple:* $(s, p, o) \in \mathcal{UB} \times \mathcal{U} \times \mathcal{UBL}$
- *RDF graphs* are sets of RDF triples

RDF simple example

band.rdf



Example in Turtle

```
@prefix : <http://xsparql.deri.org/band#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

:u2 foaf:name "U2";
    foaf:based_near :dublin .
```

Overview

- Query language for XML
  - functional language
  - typed language
- Superset of XPath

Query example (using band.rdf serialised as RDF/XML)

```
declare namespace foaf = "http://xmlns.com/foaf/0.1/";

for $x in doc("bands.rdf")//foaf:name/text()
return $x
```

SPARQL - Query language for RDF

- *triple pattern:* $(s, p, o) \in \mathcal{UBLV} \times \mathcal{UV} \times \mathcal{UBLV}$
- SPARQL queries RDF data by *pattern matching*
- Operators `optional`, `union`, `order by`, `offset`

SPARQL `select` query

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

select $name
from <bands.rdf>
where { $band foaf:name $name}
```

SPARQL - Query language for RDF

- *triple pattern:* $(s, p, o) \in \mathcal{UBLV} \times \mathcal{UV} \times \mathcal{UBLV}$
- SPARQL queries RDF data by *pattern matching*
- Operators `optional`, `union`, `order by`, `offset`

SPARQL `construct` query

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

construct { [] :bandURI $band }
from <bands.rdf>
where { $band foaf:name $name}
```

XSPARQL: XQuery + SPARQL

- Consume and generate XML and RDF data
- Adds new expressions to XQuery: `SparqlForClause` and `ConstructClause`
- Extends SPARQL with generating values, nested queries

XSPARQL example

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

for $name
from <bands.rdf>
where { $band foaf:name $name }
return <band name={$name}/>
```

Adding RDB to XSPARQL

- Another expression in XSPARQL: `SQLForClause`
- Allows SQL conjunctive queries
- Formal semantics based on XQuery semantics
  - Normalisation rules
  - Static Typing
  - Dynamic Evaluation

XSPARQL(DB) by example (I)

Query 1

```
for band.name from band
return <name>{$band.name}</name>
```

band

id	name	origin
1	“U2”	“Dublin”

XSPARQL(DB) by example (I)

Query 1

```
for band.name from band
return <name>{$band.name}</name>
```

Query 2

```
for * from band
return <name>{$band.name}</name>
```

band

id	name	origin
1	“U2”	“Dublin”

Output 1 & 2

```
<name>U2</name>
```

XSPARQL(DB) by example (II)

RDB 2 RDF example

```
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix band: <http://example.org/bands#>

for person.name as $name, band.name as $bandName
from person, band
where person.memberOf = band.id
construct { [] foaf:name $name;
                band:memberOf $bandName }
```

band

id	name	origin
1	“U2”	“Dublin”

person

ssn	name	*memberOf*
123	“Bono”	1

XSPARQL(DB) by example (II)

RDB 2 RDF example

```
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix band: <http://example.org/bands#>

for person.name as $name, band.name as $bandName
from person, band
where person.memberOf = band.id
construct { [] foaf:name $name;
                band:memberOf $bandName }
```

band

id	name	origin
1	"U2"	"Dublin"

Output

```
@prefix band: <http://example.org/bands#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

[ foaf:name "Bono" ;  band:memberOf "U2" ] .
```

person

ssn	name	*memberOf*
123	"Bono"	1

XSPARQL(DB) by example (III)

Retrieve data from DBPedia

```
prefix dbprop: <http://dbpedia.org/property/>
prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>

<kml><Document>{
for name,origin from band
return
    let $uri := fn:concat("http://dbpedia.org/resource/", $origin)
    for $lat $long
    from $uri
    where {$city geo:lat $lat; geo:long $long }
    return <Placemark>
            <name>{fn:concat("Hometown of ", $name)}</name>
            <Point><coordinates>{fn:concat($long, ",", $lat, ",0")}
            </coordinates></Point>
           </Placemark>
}</Document></kml>
```

XSPARQL(DB) by example (III)

Retrieve data from DBPedia

```
prefix dbprop: <http://dbpedia.org/property/>
prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>

<kml><Document>{
for name,origin from band
return
    let $uri := fn:concat("http://dbpedia.org/resource/", $origin)
    for $lat $long
    from $uri
    where {$city geo:lat $lat; geo:long $long }
    return <Placemark>
            <name>{fn:concat("Hometown of ", $name)}</name>
            <Point><coordinates>{fn:concat($long, ",", $lat, ",0")}
            </coordinates></Point>
           </Placemark>
}</Document></kml>
```

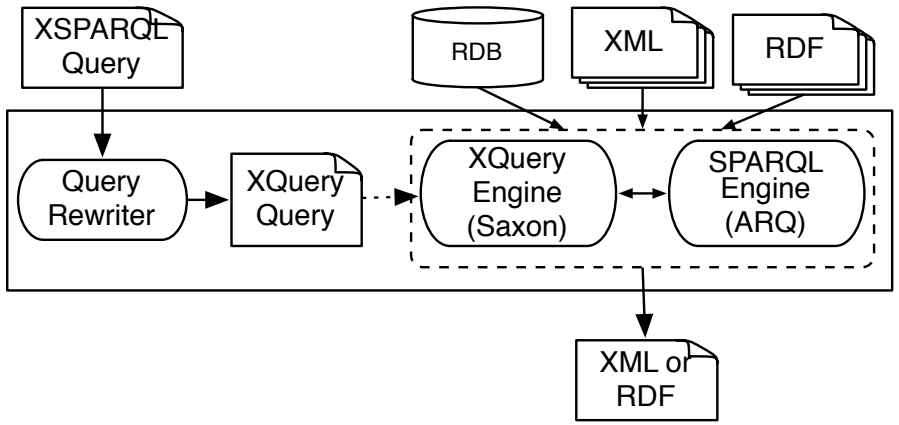
XSPARQL(DB) by example (III)

Retrieve data from DBPedia

```
prefix dbprop: <http://dbpedia.org/property/>
prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>

<kml><Document>{
for name,origin from band
return
    let $uri := fn:concat("http://dbpedia.org/resource/", $origin)
    for $lat $long
    from $uri
    where {$city geo:lat $lat; geo:long $long }
    return <Placemark>
            <name>{fn:concat("Hometown of ", $name)}</name>
            <Point><coordinates>{fn:concat($long, ",", $lat, ",0")}
            </coordinates></Point>
           </Placemark>
}</Document></kml>
```

Implementation



- Each XSPARQL query is rewritten into an XQuery
- `SQLForClauses` are translated into SQL queries
- `selected` attributes are made accessible as XQuery variables during query rewriting

Relation Name as a variable


```
for $relation in ("band", "person")
for * from $relation
return $attribute
```

Relation Name as a variable


```
for $relation in ("band", "person")
for row $row from $relation
return $row//name
```

XML representation of the result

```
<SQLResult>
  <result>
    <id>1</id>
    <name>U2</name>
    <origin>Dublin</origin>
  </result>
</SQLResult>
```

Standardised mapping from relational databases to RDF

Direct Mapping (DM): default mapping without any user input

R2RML: vocabulary (RDF) that allows to define custom mappings

Last Call

- DM http://www.w3.org/TR/rdb-direct-mapping/
- R2RML http://www.w3.org/TR/r2rml/

send comments to public-rdb2rdf-comments@w3.org

Direct Mapping

- base URI: `http://ex.org/bands#`

band		
id	name	origin
1	“U2”	“Dublin”

DM graph

```
@base <http://ex.org/bands#>

<band/id-1> rdf:type <band> .
<band/id-1> <band#id> 1 .
<band/id-1> <band#name> "U2" .
<band/id-1> <band#origin> "Dublin" .
```

Direct Mapping

- base URI: `http://ex.org/bands#`

band

id	name	origin
1	"U2"	"Dublin"

person

ssn	name	*memberOf*
123	"Bono"	1

DM graph

```
@base <http://ex.org/bands#>

<band/id-1> rdf:type <band> .
<band/id-1> <band#id> 1 .
<band/id-1> <band#name> "U2" .
<band/id-1> <band#origin> "Dublin" .

<person/ssn-123> rdf:type <person> .
<person/ssn-123> <person#ssn> 123 .
<person/ssn-123> <band#name> "Bono" .
<person/ssn-123> <band#memberOf> <band/id-1> .
```

R2RML

- Mapping represented in Turtle
- TriplesMap maps a "logical" table
- SubjectMap generates the subject
- Similar for predicate and object
- RefObjectMap for foreign key references

R2RML in Turtle


```
<#TriplesMap1> a rr:TriplesMap;
   rr:logicalTable [ rr:sqlQuery  """Select * from band""" ];

   rr:subjectMap [ rr:template "http://example.com/band/{id}" ;
                   rr:graphMap [ rr:graph rr:defaultGraph ]
                 ];

   rr:predicateObjectMap [
     rr:predicateMap [ rr:predicate foaf:name ];
     rr:objectMap    [ rr:column "name" ]
   ];

   rr:predicateObjectMap [
     rr:predicateMap [ rr:predicate foaf:homepage ];
     rr:objectMap    [ rr:template "http://example.com/band/{name}";
                       rr:termType "IRI" ]
   ];
```

RDB2RDF implementation in XSPARQL

XSPARQL RDB2RDF transformation

- DM and R2RML implemented as XSPARQL queries
- Query is executed with access to the database. Connection parameters and R2RML mapping as input.

Prototype: http://xsparql.deri.org/rdb2rdf

Conclusions

Overview

- XSPARQL:
  - Extension of XQuery
  - Transform between RDB, XML and RDF

Conclusions

Overview

- XSPARQL:
  - Extension of XQuery
  - Transform between RDB, XML and RDF
- RDB2RDF spec in XSPARQL:
  - DM implementation 100 lines approx.
  - R2RML implementation 200 lines approx.

Conclusions

Overview

- XSPARQL:
  - Extension of XQuery
  - Transform between RDB, XML and RDF
- RDB2RDF spec in XSPARQL:
  - DM implementation 100 lines approx.
  - R2RML implementation 200 lines approx.

Future Work

- Complexity of fragment of the language
- Algebra for the heterogeneous languages/input formats
- Support UPDATE