

XML Data Mediation using XSPARQL

Nuno Lopes, Laleh Kazemzadeh

October, 2013

Why use RDF for data integration (I)



Why use RDF for data integration (I)

Flexibility



Flexibility

- 1 Representation
 - Reuse any vocabularies
 - No schema required



Flexibility

- ① Representation
 - Reuse any vocabularies
 - No schema required
- ② Combining
 - Easily combine different datasets
 - RDF merge is simple



Flexibility

- ① Representation
 - Reuse any vocabularies
 - No schema required
- ② Combining
 - Easily combine different datasets
 - RDF merge is simple
- ③ Sharing
 - Linked Data
 - Built on web technologies (HTTP, URIs)



Why use RDF for data integration (II)

Global Identifiers
Schema-less
Self-Describing
Graph-Based

Why use RDF for data integration (II)



Global Identifiers	×
Schema-less	×
Self-Describing	×
Graph-Based	×

Why use RDF for data integration (II)



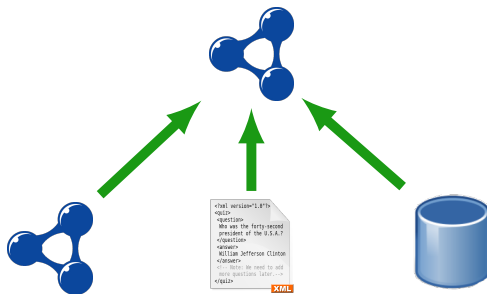
Global Identifiers	×	×
Schema-less	×	✓
Self-Describing	×	× / ✓
Graph-Based	×	× / ✓

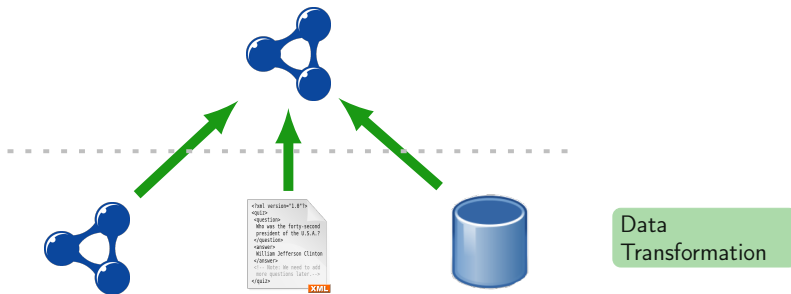
Why use RDF for data integration (II)

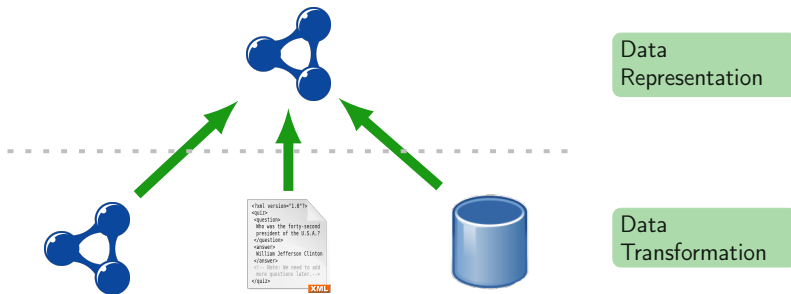


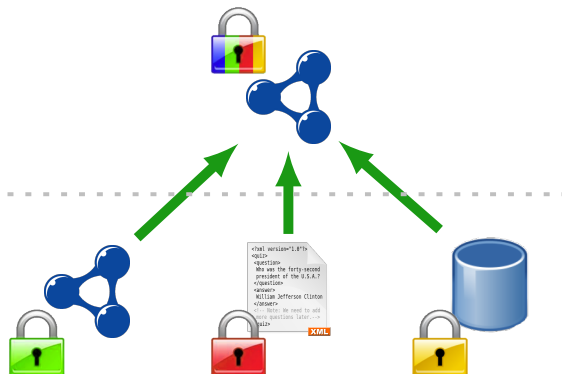
Global Identifiers	×	×	✓
Schema-less	×	✓	✓
Self-Describing	×	×	✓
Graph-Based	×	×	✓











Data
Representation

Data
Transformation



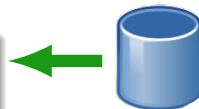
```
<?xml version="1.0"?>
<quiz>
  <question>
    Who was the forty-second
    president of the U.S.A.?
  </question>
  <answer>
    William Jefferson Clinton
  </answer>
  <!-- Note: We need to add
  more questions later.-->
</quiz>
```

XML



SQL

```
select address from person  
where name = "Nuno"
```



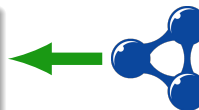
XQuery

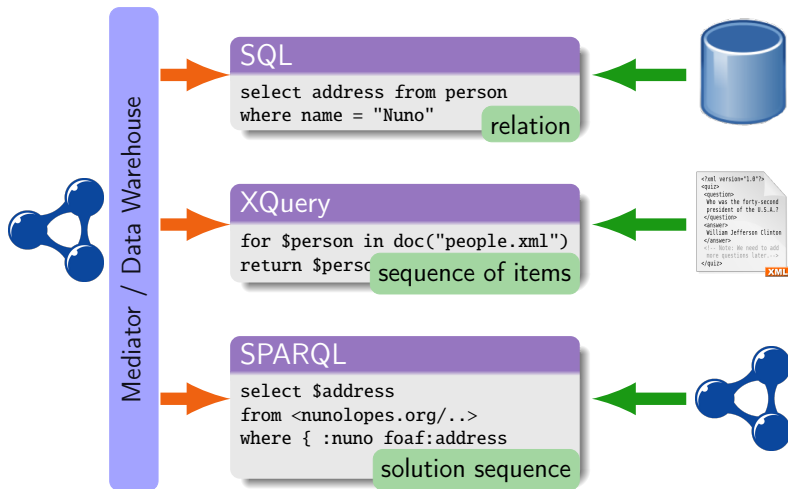
```
for $person in doc("people.xml")  
return $person//address
```

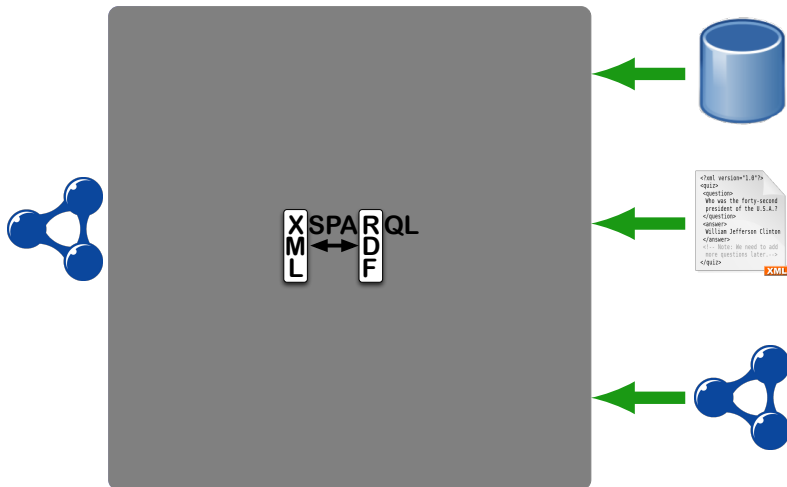


SPARQL

```
select $address  
from <nunolopes.org/..>  
where { :nuno foaf:address  
        $address }
```









- Transformation language between RDB, XML, and RDF



- Transformation language between RDB, XML, and RDF
- **Syntactic** extension of XQuery



- Transformation language between RDB, XML, and RDF
- Syntactic extension of XQuery
- **Semantics** based on XQuery's semantics



- Transformation language between RDB, XML, and RDF
- Syntactic extension of XQuery
- Semantics based on XQuery's semantics

Why based on XQuery?

- Expressive language
- Use as scripting language



- Transformation language between RDB, XML, and RDF
- Syntactic extension of XQuery
- Semantics based on XQuery's semantics

Why based on XQuery?

- Expressive language
- Use as scripting language
- **Arbitrary Nesting of expressions**

Same Language for each Format



```
<?xml version="1.0"?>
<quiz>
  <question>
    Who was the forty-second
    president of the U.S.A.?
  </question>
  <answer>
    William Jefferson Clinton
  </answer>
  <!-- Note: We need to add
  more questions later.-->
</quiz>
```

XML



Same Language for each Format

```
for var in Expr  
let var := Expr  
where Expr  
order by Expr  
return Expr
```



XQuery

```
for $person in doc("people.xml")  
return $person//address
```



```
<?xml version="1.0"?>  
<quiz>  
  <question>  
    Who was the forty-second  
    president of the U.S.A.?  
  </question>  
  <answer>  
    William Jefferson Clinton  
  </answer>  
  <!-- Note: We need to add  
  more questions later.-->  
</quiz>
```

XML



Same Language for each Format

```
for var in Expr  
let var := Expr  
where Expr  
order by Expr  
return Expr
```



XSPARQL

```
for $person in doc("people.xml")  
return $person//address
```



```
<?xml version="1.0"?>  
<quiz>  
  <question>  
    Who was the forty-second  
    president of the U.S.A.?  
  </question>  
  <answer>  
    William Jefferson Clinton  
  </answer>  
  <!-- Note: We need to add  
  more questions later.-->  
</quiz>
```

XML



Same Language for each Format

```
for SelectSpec  
from RelationList  
where WhereSpecList  
return Expr
```

XSPARQL

```
for address as $address from people  
where name = "Nuno"  
return $address
```



```
<?xml version="1.0"?>  
<quiz>  
  <question>  
    Who was the forty-second  
    president of the U.S.A.?  
  </question>  
  <answer>  
    William Jefferson Clinton  
  </answer>  
  <!-- Note: We need to add  
  more questions later.-->  
</quiz>
```

XML



Same Language for each Format

```
for varlist  
from DatasetClause  
where { pattern }  
return Expr
```

XSPARQL

```
for $address from <nunolopes.org/..>  
where { :nuno foaf:address $address }  
return $address
```



```
<?xml version="1.0"?>  
<quiz>  
  <question>  
    Who was the forty-second  
    president of the U.S.A.?  
  </question>  
  <answer>  
    William Jefferson Clinton  
  </answer>  
  <!-- Note: We need to add  
  more questions later.-->  
</quiz>
```

XML





[Home](#) | [Browse](#) | [Gene search](#) | [Text search](#) | [Blast](#) | [Downloads](#) | [Previous version](#) | [Summary](#) | [FAQ](#) | [Contact us](#) | [Subscribe](#)

InParanoid: Eukaryotic Ortholog Groups

100 organisms: 1687023 sequences

Version 7.0, Updated June 2009 ([release notes](#))

BROWSE the database - Select two species and view all their orthologs
SEARCH BY SEQUENCE IDs - View orthologs of a specific gene or protein
TEXT SEARCH - Query InParanoid by keywords
BLAST SEARCH - Find orthologs in InParanoid similar to your protein sequence
DOWNLOAD DATA - Obtain tables, html, [orthoXML](#), sequences and core data
SUMMARY OF INPARANOID - Statistics of the database and genomes used
ORTHOPHYLOGRAM - Phylogenetic tree based on the average fraction of InParanoid orthologs between species.
MAILING LIST - Subscribe to the InParanoid mailing list

Stand-alone InParanoid Program

InParanoid Version 4.1 is available [here](#)

Browse the database

Choose two species

Homo Sapiens Mus musculus 50

Cluster 1					
Protein ID	Species	Score <input type="text"/>	Bootstrap <input type="text"/>	Description	Alternative ID
ENSP00000364178	Homo Sapiens	1	100%	titin isoform novex-3 [Source:RefSeq peptide;Acc:NP_596870]	
ENSMUSP00000097561	Mus musculus	1	100%	titin Gene [Source:MGI (curated);Acc:Ttn-019]	MGI:98864 (MGI ID) Q8BIH3 (UniProt/TrEMBL Accession) Q8BUJ0 (UniProt/TrEMBL Accession) Q3UT48 (UniProt/TrEMBL Accession) A2AT59 (UniProt/TrEMBL Accession) A2AT63 (UniProt/TrEMBL Accession)

Usecase: Inparanoid

Browse the database

Choose two species

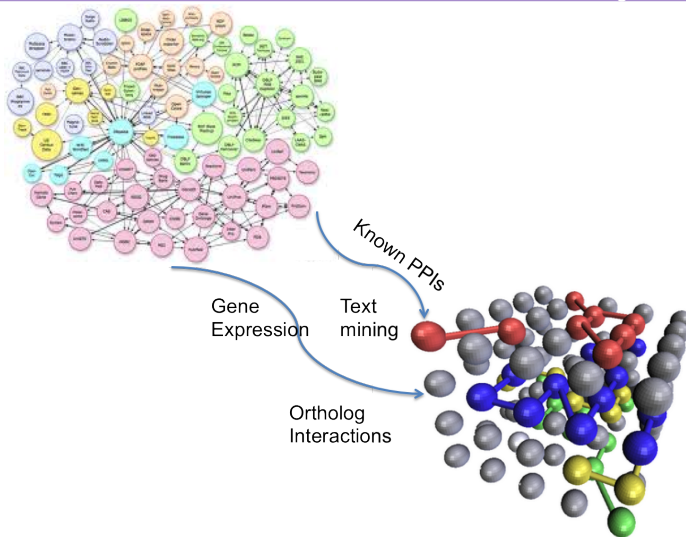
Homo Sapiens Mus musculus 50 Submit

Cluster 1					
Protein ID	Species	Score <input type="text"/>	Bootstrap <input type="text"/>	Description	Alternative ID
ENSP00000364178	Homo Sapiens	1	100%	titin isoform novex-3 [Source:RefSeq peptide;Acc:NP_596870]	
ENSMUSP00000097561	Mus musculus	1	100%	titin Gene [Source:MGI (curated);Acc:Ttn-019]	MGI:98864 (MGI ID) Q8BIH3 (UniProt/TrEMBL Accession) Q8BUJ0 (UniProt/TrEMBL Accession) Q3UT48 (UniProt/TrEMBL Accession) A2AT59 (UniProt/TrEMBL Accession) A2AT63 (UniProt/TrEMBL Accession)

```
<?xml version="1.0"?>
<quiz>
  <question>
    Who was the forty-second
    president of the U.S.A.?
  </question>
  <answer>
    William Jefferson Clinton
  </answer>
  <!-- Note: We need to add
  more questions later.-->
</quiz>
```

XML

Usecase: Predicting Protein-Protein Interactions



```
for $gene in doc("H.sapiens-M.musculus.xml")//gene
  let $id := fn:data($gene/@id)
  let $geneId :=fn:data($gene/@geneId)
  let $protId :=fn:data($gene/@protId)
  let $uri := fn:concat("http://bioinfo.deri.ie/inparanoid/", $id)
  construct { <{$uri}> purl:identifier {$id};
              :geneID {$geneId};
              :protID {$protId} . }
```

```
for $gene in doc("H.sapiens-M.musculus.xml")//gene
  let $id := fn:data($gene/@id)
  let $geneId :=fn:data($gene/@geneId)
  let $protId :=fn:data($gene/@protId)
  let $uri := fn:concat("http://bioinfo.deri.ie/inparanoid/" . $id)
  construct { <{$uri}> purl:identifier {$id};
               :geneID {$geneId};
               :protID {$protId} . }
```

construct
clause generates
RDF

```
for $gene in doc("H.sapiens-M.musculu
  let $id := fn:data($gene/@id)
  let $geneId :=fn:data($gene/@geneId)
  let $protId :=fn:data($gene/@protId)
  let $uri := fn:concat("http://bioinfo.berli.ie/inparanoid/", $id)
  construct { <{$uri}> purl:identifier {$id};
              :geneID {$geneId};
              :protID {$protId} . }
```

Arbitrary XSPARQL
expressions in subject,
predicate, and object

```
for $gene in doc("H.sapiens-M.musculus.xml")//gene
  let $id := fn:data($gene/@id)
  let $geneID :=fn:data($gene/@geneID)
  let $protID :=fn:data($gene/@protID)
  let $uri := fn:concat("http://bioinfo.deri.ie/inparanoid/", $id)
  construct { <{$uri}> purl:identifier {$id};
              :geneID {$geneID};
              :protID {$protID} . }
```

Query Result

```
:1 dct:identifier "1" .
:1 bioinfo:geneID "ENSG000000155657" .
gene:ENSG000000155657 rdf:type bioinfo:Gene .
:1 bioinfo:protID "ENSP000000364178" .
protein:ENSP000000364178 rdf:type bioinfo:protein .
gene:ENSG000000155657 bioinfo:source_database "Ensembl" .
protein:ENSG000000155657 bioinfo:organism <http://purl.uniprot.org/taxonomy/9606>
```

Usecase: Combining Inparanoid with BridgeDB

Browse the database

Choose two species

Homo Sapiens

Mus musculus

50

Submit

Cluster 1					
Protein ID	Species	Score	Bootstrap	Description	Alternative ID
ENSP00000364178	Homo Sapiens	1	100%	titin isoform novex-3 [Source:RefSeq peptide;Acc:NP_596870]	
ENSMUSP00000097561	Mus musculus	1	100%	titin Gene [Source:MGI (curated);Acc:Ttn-019]	MGI:98864 (MGI ID) Q8BIH3 (UniProt/TrEMBL Accession) Q8BUJ0 (UniProt/TrEMBL Accession) Q3UT48 (UniProt/TrEMBL Accession) A2AT59 (UniProt/TrEMBL Accession) A2AT63 (UniProt/TrEMBL Accession)





[Login](#) | [Help/Guide](#) | [About Trac](#) | [Preferences](#)

	Wiki	Timeline	Roadmap	Browse Source	View Tickets	Search
--	----------------------	--------------------------	-------------------------	-------------------------------	------------------------------	------------------------

[Start Page](#) | [Index](#) | [History](#) | [Last Change](#)

Welcome to BridgeDb

BridgeDb is an id mapping framework for bioinformatics applications. BridgeDb lets you add the following capabilities quickly and easily:

- translate identifiers from one system to another
- search references by id or symbol
- link out to online information for an identifier

BridgeDb is not tied to a specific source of mapping information. Instead it provides an abstraction layer so you can switch easily between flat files, relational databases and several different web services. The following applications make use of BridgeDb?: [PathVisio pathway analysis tool](#), [WikiPathways](#), [CyThesaurus Cytoscape plugin](#), [NetworkMerge Cytoscape plugin](#), [BatchMapper](#), a command line tool and the [HOMECAAT Cytoscape plugin](#).

News

- Aug 29, 2013 BridgeDb 1.2.0 is planned to be release this September
- Feb 24, 2011 **BridgeDb 1.1.0 released** This is a development preview.
- Feb 24, 2011 **BridgeDb 1.0.3 released** This is a stable release.
- Apr 29, 2010 **BridgeDb 1.0.1 released** with a couple of small bugfixes
- Mar 5, 2010 **BridgeDb 1.0 released**



[Login](#) | [Help/Guide](#) | [About Trac](#) | [Preferences](#)

	Wiki	Timeline	Roadmap	Browse Source	View Tickets	Search
--	----------------------	--------------------------	-------------------------	-------------------------------	------------------------------	------------------------

[Start Page](#) | [Index](#) | [History](#) | [Last Change](#)

Welcome to BridgeDb

BridgeDb is an id mapping framework for bioinformatics applications. BridgeDb lets you add the following capabilities quickly and easily:

- translate identifiers from one system to another
- search references by id or symbol
- link out to online information for an identifier

BridgeDb is not tied to a specific source of mapping information. Instead it provides an abstraction layer so you can switch easily between flat files, relational databases and several different web services. The following applications make use of BridgeDb?: [PathVisio pathway analysis tool](#), [WikiPathways](#), [CyThesaurus Cytoscape plugin](#), [NetworkMerge Cytoscape plugin](#), [BatchMapper](#), a command line tool and the [HOMECAAT Cytoscape plugin](#).

News

- Aug 29, 2013 BridgeDb 1.2.0 is planned to be release this September
- Feb 24, 2011 **BridgeDb 1.1.0 released** This is a development preview.
- Feb 24, 2011 **BridgeDb 1.0.3 released** This is a stable release.
- Apr 29, 2010 **BridgeDb 1.0.1 released** with a couple of small bugfixes
- Mar 5, 2010 **BridgeDb 1.0 released**




```
for $gene in doc("H.sapiens-M.musculus.xml")//gene
let $id := fn:data($gene/@id)
let $geneId :=fn:data($gene/@geneId)
let $protId :=fn:data($gene/@protId)
let $uri := fn:concat("http://bioinfo.deri.ie/inparanoid/", $id)
  for $link $idRight from link
  where idRight = $geneId
  construct { <{$uri}> purl:identifier {$id};
              :geneID {$geneId};
              :protID {$protId};
              :link {$idRight} . }
```

```
for $gene in doc("H.sapiens-M.musculus.xml")//gene
let $id := fn:data($gene/@id)
let $geneId :=fn:data($gene/@geneId)
let $protId :=fn:data($gene/@protId)
let $uri := fn:concat("http://bioinfo.deri.ie/inparanoid/", $id)
for $link $idRight from link
where idRight = $geneId
construct { <{$uri}> purl:identifier {$id};
            :geneID {$geneId};
            :protID {$protId};
            :link {$idRight} . }
```



```
<?xml version="1.0"?>
<quiz>
  <question>
    Who was the forty-second
    president of the U.S.A.?
  </question>
  <answer>
    William Jefferson Clinton
  </answer>
  ... More! We need to add
  more questions later...
</quiz>
```

XML

```
for $gene in doc("H.sapiens-M.musculus.xml")//gene
let $id := fn:data($gene/@id)
let $geneId :=fn:data($gene/@geneId)
let $protId :=fn:data($gene/@protId)
let $uri := fn:concat("http://bioinfo.deri.ie/inparam", $id)
for $link $idRight from link
where idRight = $geneId
construct { <{$uri}> purl:identifier {$id};
           :geneID {$geneId};
           :protID {$protId};
           :link {$idRight} . }
```



```
for $gene in doc("H.sapiens-M.musculus.xml")//gene
let $id := fn:data($gene/@id)
let $geneId :=fn:data($gene/@geneId)
let $protId :=fn:data($gene/@protId)
let $uri := fn:concat("http://bioinfo.deri.ie/inparanoid/", $id)
for $link $idRight from link
where idRight = $geneId
construct { <{$uri}> purl:identifier {$id};
              :geneID {$geneId};
              :protID {$protId};
              :link {$idRight} . }
```



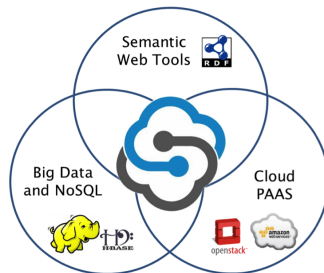
```
for $gene in doc("H.sapiens-M.musculus.xml")//gene
let $id := fn:data($gene/@id)
let $geneId :=fn:data($gene/@geneId)
let $protId :=fn:data($gene/@protId)
let $uri := fn:concat("http://bioinfo.deri.ie/inparanoid/", $id)
  for $link $idRight from link
  where idRight = $geneId
  construct { <{$uri}> purl:identifier {$id};
              :geneID {$geneId};
              :protID {$protId};
              :link {$idRight} . }
```

More involved XSPARQL queries: RDB2RDF

- Direct Mapping: ~130 LoC
- R2RML: ~290 LoC

CloudSpace Platform offers
Infrastructure for the intersection of:

- Linked Data
- Big Data
- Cloud Computing



Cloudspaces

A collection of tool to help users **Extract, Transform & Load big data sets.**

Cloudspaces is based on **Sindice**, allowing user-defined Linked Data pipelines to be used in the Cloud

Cloudspaces

A collection of tool to help users Extract, Transform & Load big data sets.

Cloudspaces is based on Sindice, allowing user-defined Linked Data pipelines to be used in the Cloud

Sindice

- Linked Data "Search Engine"
- High availability Sparql Endpoint:
 - 12 Billion Triples
 - Can load 100 million triples a day (updated daily)

Overview

XSPARQL

Usecases & Other features

Beyond XSPARQL

Conclusions

Data integration features in SPARQL 1.1

- Aggregates
- Subqueries
- Federation Extensions
- Negation
- Expressions in the SELECT clause
- Property Paths
- Assignment

Data integration features in SPARQL 1.1

- Aggregates
- Subqueries
- Federation Extensions
- Negation
- Expressions in the SELECT clause
- Property Paths
- Assignment

```
PREFIX dbpedia2: <http://dbpedia.org/property/>  
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
SELECT ?N ?MyB  
FROM <http://polleres.net/foaf.rdf>  
WHERE { [ foaf:birthday ?MyB ].
```

```
    SERVICE <http://dbpedia.org/sparql> {  
        SELECT ?N WHERE { [ dbpedia2:born ?B; foaf:name ?N ]. }  
    }  
    FILTER ( Regex(Str(?B),str(?MyB)) )  
}
```

```
PREFIX dbpedia2: <http://dbpedia.org/property/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?N ?MyB
FROM <http://polleres.net/foaf.rdf>
WHERE { [ foaf:birthday ?MyB ].

    SERVICE <http://dbpedia.org/sparql> {
        SELECT ?N WHERE { [ dbpedia2:born ?B; foaf:name ?N ]. }
    }
    FILTER ( Regex(Str(?B),str(?MyB)) )
}
```

```
PREFIX dbpedia2: <http://dbpedia.org/property/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?N ?MyB
FROM <http://polleres.net/foaf.rdf>
WHERE { [ foaf:birthday ?MyB ].

    SERVICE <http://dbpedia.org/sparql> {
        SELECT ?N WHERE { [ dbpedia2:born ?B; foaf:name ?N ]. }
    }
    FILTER ( Regex(Str(?B),str(?MyB)) )
}
```

Problem

- limits from SPARQL endpoints prevent this query from working!

XSPARQL endpoint

```
prefix dbprop: <http://dbpedia.org/property/>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix : <http://xsparql.deri.org/bday#>

let $MyB := for * from <http://polleres.net/foaf.rdf>
where { [ foaf:birthday $B ]. }
return $B

for * from <http://dbpedia.org/>
endpoint <http://dbpedia.org/sparql>
where { [ dbprop:born $B; foaf:name $N ].
        filter ( regex(str($B),str($MyB)) ) }
construct { :axel :sameBirthDayAs $N }
```

XSPARQL endpoint

```
prefix dbprop: <http://dbpedia.org/property/>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix : <http://xsparql.deri.org/bday#>

let $MyB := for * from <http://polleres.net/foaf.rdf>
where { [ foaf:birthday $B ]. }
return $B

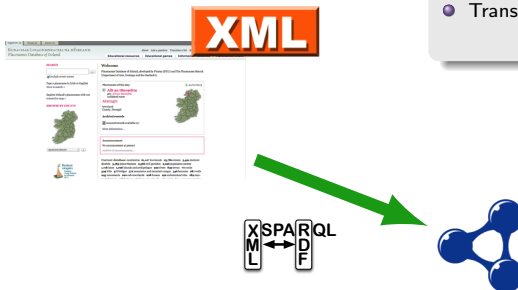
for * from <http://dbpedia.org/>
endpoint <http://dbpedia.org/sparql>
where { [ dbprop:born $B; foaf:name $N ].
        filter ( regex(str($B),str($MyB)) ) }
construct { :axel :sameBirthDayAs $N }
```


- Data provided in XML



Converting Logainm dump to RDF

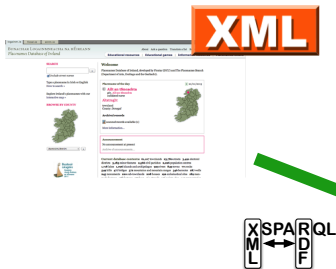
- Data provided in XML
- Translated to RDF using XSPARQL



Converting Logainm dump to RDF

- Data provided in XML
- Translated to RDF using XSPARQL
- Exposed using Openlink Virtuoso

~ 1.3M triples



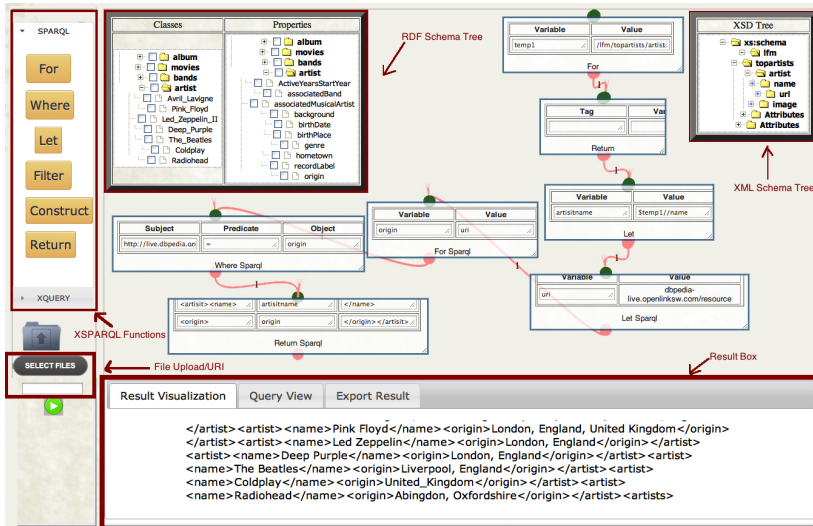
Overview

XSPARQL

Usecases & Other features

Beyond XSPARQL

Conclusions



The interface displays a SPARQL query and its execution steps. The query is:

```

  SELECT ?origin
  WHERE {
    ?artist dbpedia:origin ?origin
  }

```

The query is visualized as a graph with nodes for variables and values. The graph shows the following structure:

- Variable**: `temp1`, `uri`
- Value**: `//fm/topartists/artist:`, `dbpedia-live.openlinksw.com/resource`
- For**: `For Sparql`
- Let**: `Let Sparql`
- Return**: `Return Sparql`

The **XML Schema Tree** shows the structure of the query results, including elements like `xs:schema`, `ifm`, `topartists`, `artist`, `name`, `uri`, `image`, and `Attributes`.

The **Result Visualization** shows the query results in a table format:

Artist	Origin
Pink Floyd	London, England, United Kingdom
Led Zeppelin	London, England
Deep Purple	London, England
The Beatles	Liverpool, England
Coldplay	United Kingdom
Radiohead	Abingdon, Oxfordshire

Annotations refer to a specific **domain**

Temporal

:nuno :address :Galway . [2008,2012]

Annotations refer to a specific **domain**

Temporal

:nuno :address :Galway . [2008,2012]

Fuzzy

:nuno :address :Dublin . 0.9

Annotations refer to a specific **domain**

Temporal

:nuno :address :Galway . [2008,2012]

Fuzzy

:nuno :address :Dublin . 0.9

Access Control

:nuno :address :Galway . [n]


```
ap address Galway.  
ap birthday "24/03" .  
nl address Galway.  
nl birthday "23/12" .
```

AnQL AC Query

```
SELECT * WHERE { $person :birthday $birthday . }
```

<i>\$person</i>	<i>\$birthday</i>
ap	"24/03"
nl	"23/12"

```
ap address Galway.  
ap birthday "24/03" : [[ap]].  
nl address Galway.  
nl birthday "23/12" : [[nl]].
```

AnQL AC Query

```
SELECT * WHERE { $person :birthday $birthday : [[nl]] . }
```

<i>\$person</i>	<i>\$birthday</i>
ap	"24/03"
nl	"23/12"

```
ap address Galway.  
ap birthday "24/03" : [[ap]].  
nl address Galway.  
nl birthday "23/12" : [[nl]].
```

AnQL AC Query

```
SELECT * WHERE { $person :birthday $birthday :[[nl]] . }
```

<u>\$person</u>	<u>\$birthday</u>
ap	"24/03"
nl	"23/12"

- You can use **XSPARQL** to easily merge data from different sources in a common language

- You can use XSPARQL to easily merge data from different sources in a common language
- **Annotated RDF** can provide Access Control over RDF data

- You can use XSPARQL to easily merge data from different sources in a common language
- Annotated RDF can provide Access Control over RDF data

Useful links

XSPARQL <http://xsparql.deri.org/>

XSPARQLViz <http://deri-srvgal33.nuig.ie:8080/XsparqlViz/>

logainm <http://data.logainm.ie/>

- You can use XSPARQL to easily merge data from different sources in a common language
- Annotated RDF can provide Access Control over RDF data

Useful links

XSPARQL <http://xsparql.deri.org/>

XSPARQLViz <http://deri-srvgal33.nuig.ie:8080/XsparqlViz/>

logainm <http://data.logainm.ie/>

Thank you! Questions?