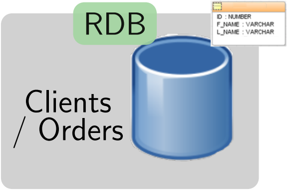
Integrating Heterogeneous Data by Extending Semantic Web Standards

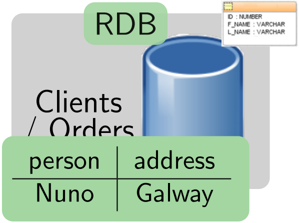
Nuno Lopes

26 November, 2012

Data Integration: Pizza Delivery Company



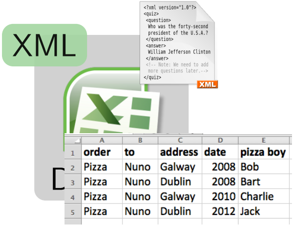
Data Integration: Pizza Delivery Company



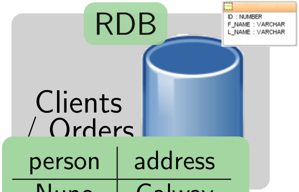
Data Integration: Pizza Delivery Company

person	address
Nuno	Galway

Data Integration: Pizza Delivery Company

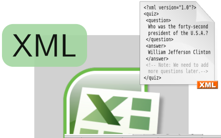


	A	B	C	D	E
1	order	to	address	date	pizza boy
2	Pizza	Nuno	Galway	2008	Bob
3	Pizza	Nuno	Dublin	2008	Bart
4	Pizza	Nuno	Galway	2010	Charlie
5	Pizza	Nuno	Dublin	2012	Jack

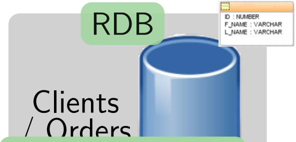


person	address
Nuno	Galway

Data Integration: Pizza Delivery Company



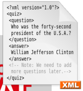
	A	B	C	D	E
1	**order**	**to**	**address**	**date**	**pizza boy**
2	Pizza	Nuno	Galway	2008	Bob
3	Pizza	Nuno	Dublin	2008	Bart
4	Pizza	Nuno	Galway	2010	Charlie
5	Pizza	Nuno	Dublin	2012	Jack



person	address
Nuno	Galway

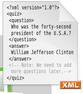
Data Integration: Pizza Delivery Company

XML




```
<person>
  <name>Nuno</name>
  <address>Dublin</address>
</person>
```

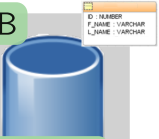
XML



D

	A	B	C	D	E
1	**order**	**to**	**address**	**date**	**pizza boy**
2	Pizza	Nuno	Galway	2008	Bob
3	Pizza	Nuno	Dublin	2008	Bart
4	Pizza	Nuno	Galway	2010	Charlie
5	Pizza	Nuno	Dublin	2012	Jack

RDB

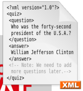


Clients / Orders

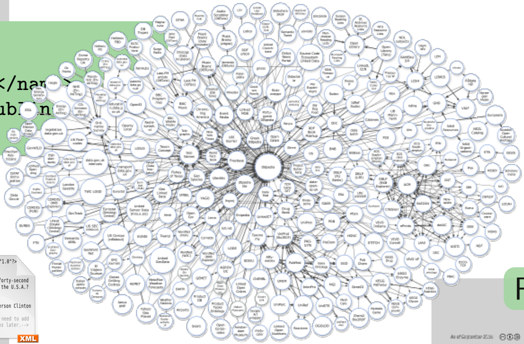
person	address
Nuno	Galway

Data Integration: Pizza Delivery Company

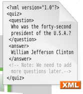
XML




```
<person>
  <name>Nuno</name>
  <address>Dublin</address>
</person>
```

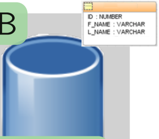


XML



	A	B	C	D	E
1	order	to	address	date	pizza boy
2	Pizza	Nuno	Galway	2008	Bob
3	Pizza	Nuno	Dublin	2008	Bart
4	Pizza	Nuno	Galway	2010	Charlie
5	Pizza	Nuno	Dublin	2012	Jack

RDB

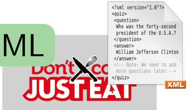


Clients / Orders

person	address
Nuno	Galway

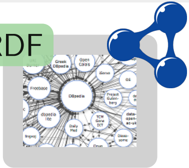
Data Integration: Pizza Delivery Company

XML




```
<person>
  <name>Nuno</name>
  <address>Dublin</address>
</person>
```

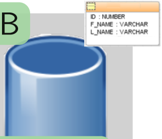
RDF



XML

	A	B	C	D	E
1	order	to	address	date	pizza boy
2	Pizza	Nuno	Galway	2008	Bob
3	Pizza	Nuno	Dublin	2008	Bart
4	Pizza	Nuno	Galway	2010	Charlie
5	Pizza	Nuno	Dublin	2012	Jack

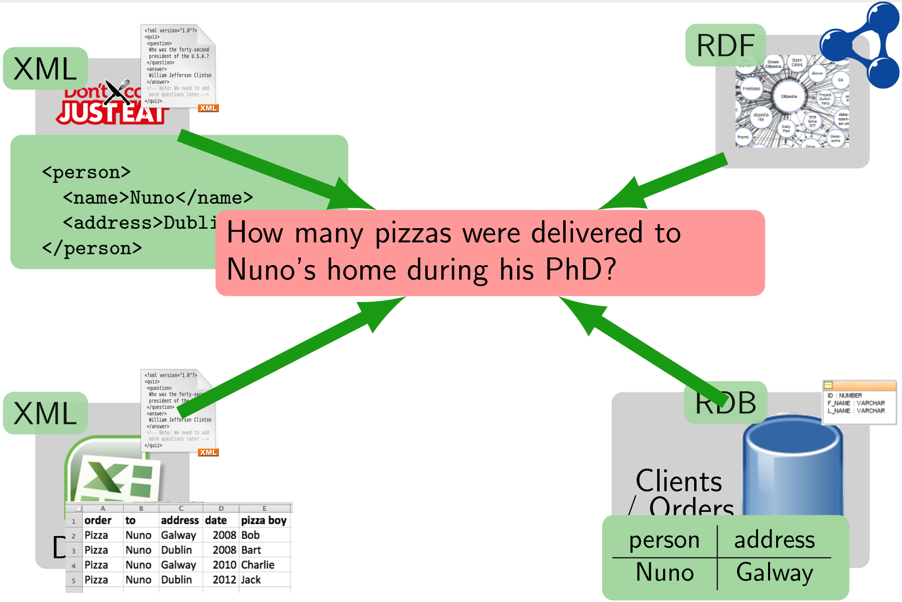
RDB



Clients / Orders

person	address
Nuno	Galway

Data Integration: Pizza Delivery Company

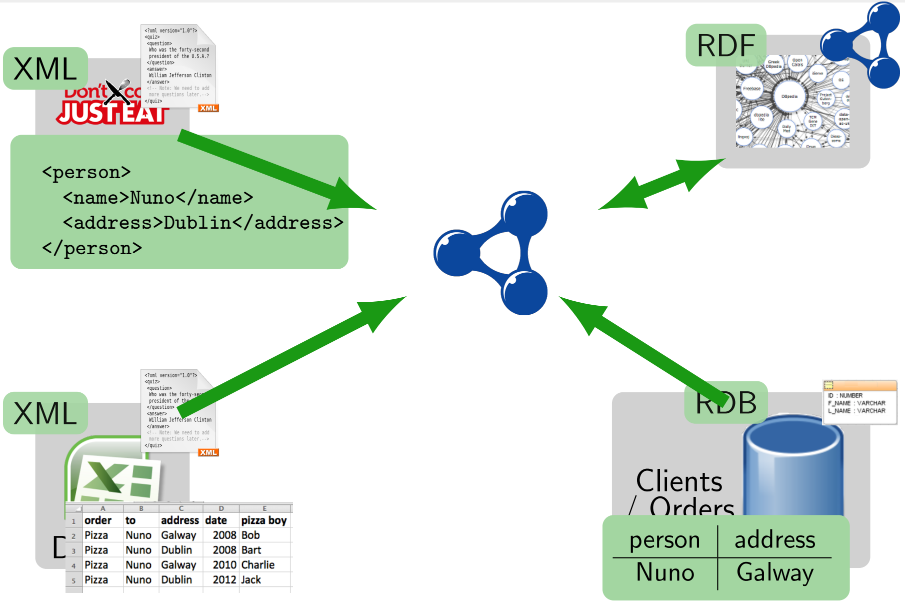


How many pizzas were delivered to Nuno's home during his PhD?

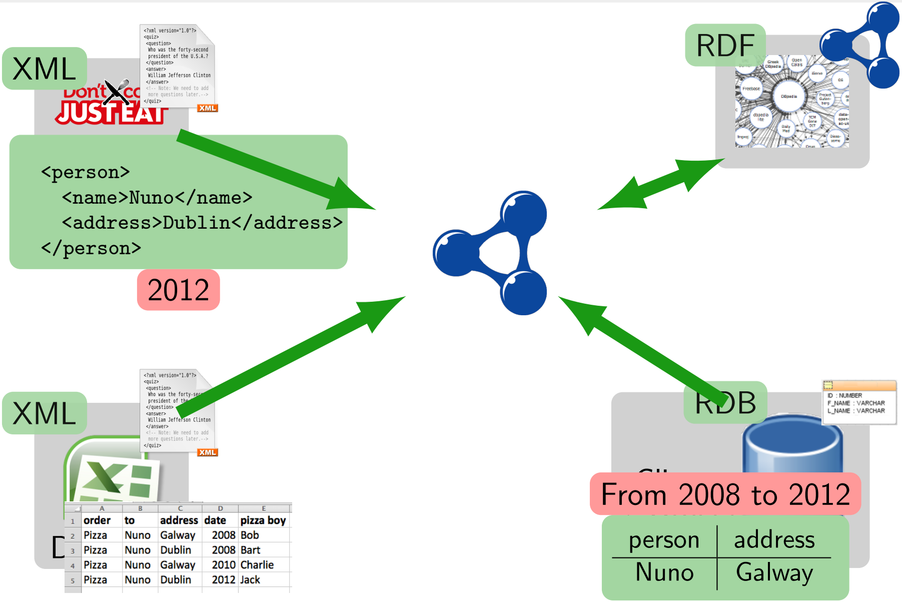
	A	B	C	D	E
1	order	to	address	date	pizza boy
2	Pizza	Nuno	Galway	2008	Bob
3	Pizza	Nuno	Dublin	2008	Bart
4	Pizza	Nuno	Galway	2010	Charlie
5	Pizza	Nuno	Dublin	2012	Jack

person	address
Nuno	Galway

Data Integration: Pizza Delivery Company



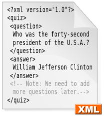
Data Integration: Pizza Delivery Company



	A	B	C	D	E
1	order	to	address	date	pizza boy
2	Pizza	Nuno	Galway	2008	Bob
3	Pizza	Nuno	Dublin	2008	Bart
4	Pizza	Nuno	Galway	2010	Charlie
5	Pizza	Nuno	Dublin	2012	Jack

person	address
Nuno	Galway

Using Query Languages for Data Integration



Using Query Languages for Data Integration

SQL

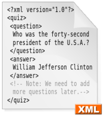
```
select address from clients
where person = "Nuno"
```

SPARQL

```
select $lat $long
from <geonames.org/..>
where { $point geo:lat $lat;
                geo:long $long }
```

XQuery

```
for $person in doc("eat.ie/..")
return $person//name
```



Using Query Languages for Data Integration

SQL

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

relation

SPARQL

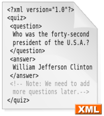
```
select $lat $long
from <geonames.org/..>
where { $point geo:lat $lat;
```

solution sequence

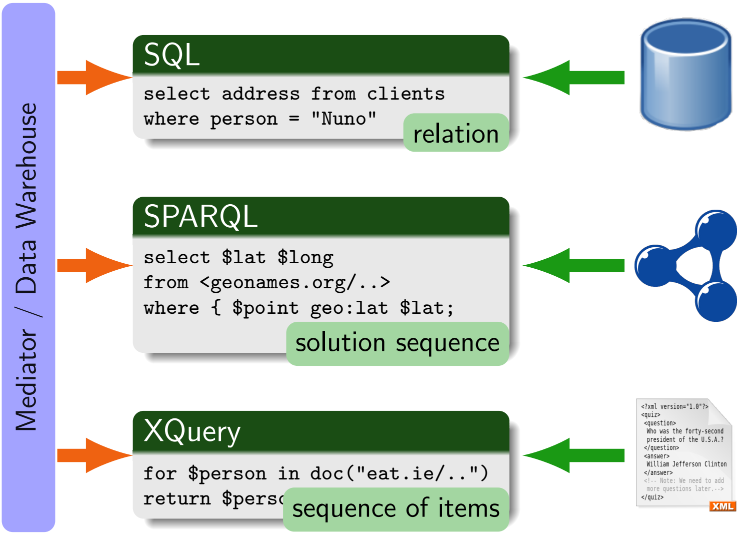
XQuery

```
for $person in doc("eat.ie/..")
return $perso
```

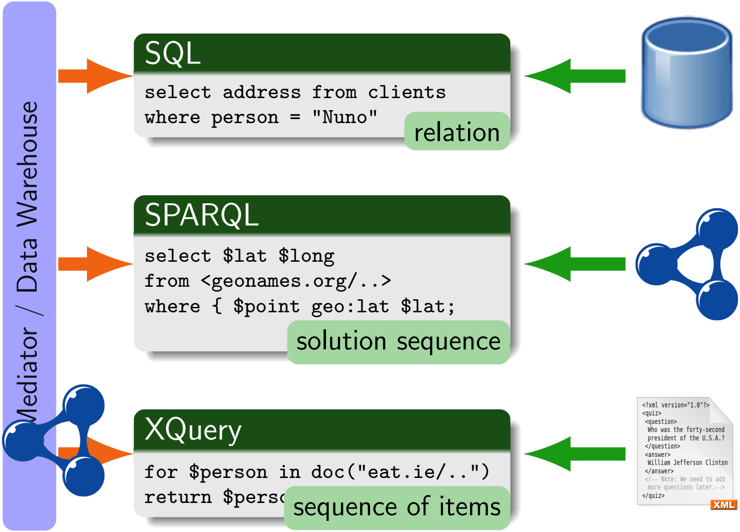
sequence of items



Using Query Languages for Data Integration



Using Query Languages for Data Integration



How many pizzas were delivered to Nuno's home during his PhD?

Mediator / Data Warehouse

SQL

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

relation

SPARQL

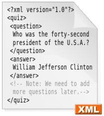
```
select $lat $long
from <geonames.org/..>
where { $point geo:lat $lat;
```

solution sequence

XQuery

```
for $person in doc("eat.ie/..")
return $perso
```

sequence of items



How to represent context information in RDF?

- RDF triples

```
:nuno :address :Galway .
:nuno :address :Dublin .
```

How to represent context information in RDF?

- RDF triples

```
:nuno :address :Galway .
:nuno :address :Dublin .
```

Not enough!

How to represent context information in RDF?

- RDF triples

```
:nuno :address :Galway .
:nuno :address :Dublin .
```

Not enough!

- Domain vocabulary/ontology

```
:address1 a          :AddressRecord;
          :person    :nuno;
          :address   :Galway;
          :start     "2008" ;
          :end       "2012" .
```

How to represent context information in RDF?

- RDF triples

```
:nuno :address :Galway .
:nuno :address :Dublin .
```

Not enough!

- Domain vocabulary/ontology

```
:address1 a           :AddressRecord;
          :person     :nuno;
          :address    :Galway;
          :start      "2008" ;
          :end        "2012" .
```

- Reification

```
:address1 rdf:type rdf:Statement
          rdf:subject :nuno;
          rdf:predicate :address ;
          rdf:object :Dublin ;
          :loc  <http://eat.ie/> .
```

How to represent context information in RDF?

- RDF triples

```
:nuno :address :Galway .
:nuno :address :Dublin .
```

Not enough!

- Domain vocabulary/ontology

```
:address1 a           :AddressRecord;
          :person     :nuno;
          :address    :Galway;
          :start      "2008" ;
          :end        "2012" .
```

- Reification

```
:address1 rdf:type rdf:Statement
          rdf:subject :nuno;
          rdf:predicate :address ;
          rdf:object :Dublin ;
          :loc  <http://eat.ie/> .
```

- Named Graphs

How to represent context information in RDF?

- RDF triples

```
:nuno :address :Galway .
:nuno :address :Dublin .
```

Not enough!

- Domain vocabulary/ontology

```
:address1 a          :AddressRecord;
          :person    :nuno;
          :address   :Galway;
          :start     "2008" ;
          :end       "2012" .
```

No defined semantics!

- Reification

```
:address1 rdf:type rdf:Statement
         rdf:subject :nuno;
         rdf:predicate :address ;
         rdf:object :Dublin ;
         :loc  <http://eat.ie/> .
```

No defined semantics!

- Named Graphs

Hypothesis and Research Questions

Hypothesis

Efficient data integration over heterogenous data sources can be achieved by

Hypothesis and Research Questions

Hypothesis

Efficient data integration over heterogenous data sources can be achieved by

- a combined query language that accesses heterogenous data in its original sources

Hypothesis and Research Questions

Hypothesis

Efficient data integration over heterogenous data sources can be achieved by

- a combined query language that accesses heterogenous data in its original sources
- optimisations for efficient query evaluation for this language

Hypothesis and Research Questions

Hypothesis

Efficient data integration over heterogenous data sources can be achieved by

- a combined query language that accesses heterogenous data in its original sources
- optimisations for efficient query evaluation for this language
- an RDF-based format with support for context information

Hypothesis and Research Questions

Hypothesis

Efficient data integration over heterogenous data sources can be achieved by

- a combined query language that accesses heterogenous data in its original sources
- optimisations for efficient query evaluation for this language
- an RDF-based format with support for context information

Research Questions

- How to design a query language that bridges the different formats?

Hypothesis and Research Questions

Hypothesis

Efficient data integration over heterogenous data sources can be achieved by

- a combined query language that accesses heterogenous data in its original sources
- optimisations for efficient query evaluation for this language
- an RDF-based format with support for context information

Research Questions

- How to design a query language that bridges the different formats?
- Can we reuse existing optimisations from other query languages?

Hypothesis and Research Questions

Hypothesis

Efficient data integration over heterogenous data sources can be achieved by

- a combined query language that accesses heterogenous data in its original sources
- optimisations for efficient query evaluation for this language
- an RDF-based format with support for context information

Research Questions

- How to design a query language that bridges the different formats?
- Can we reuse existing optimisations from other query languages?
- How to adapt RDF and SPARQL for context information?

Outline and Contributions

- XSPARQL
  - Syntax & Semantics
  - Implementation
  - Evaluation
- Annotated RDF(S)
  - Annotated RDF(S)
  - AnQL: Annotated SPARQL
  - Architecture

Outline and Contributions

- XSPARQL
  - Syntax & Semantics
  - Implementation
  - Evaluation

[JoDS2012], [EPIA2011]

- Annotated RDF(S)
  - Annotated RDF(S)
  - AnQL: Annotated SPARQL
  - Architecture

Outline and Contributions

- XSPARQL
  - Syntax & Semantics
  - Implementation
  - Evaluation
- Annotated RDF(S)
  - Annotated RDF(S)
  - AnQL: Annotated SPARQL
  - Architecture

[AAAI2010], [ISWC2010], [JWS2012]

XSPARQL

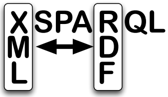
- Transformation language between RDB, XML, and RDF

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

XSPARQL

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

XSPARQL



- 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

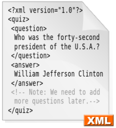
XSPARQL

- 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

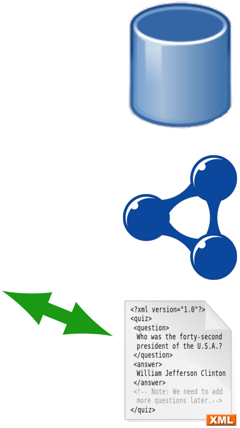


Same Language for each Format

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

XQuery

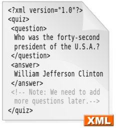
```
for $person in doc("eat.ie/..")
return $person//name
```



Same Language for each Format

XSPARQL

```
for $person in doc("eat.ie/..")
return $person//name
```

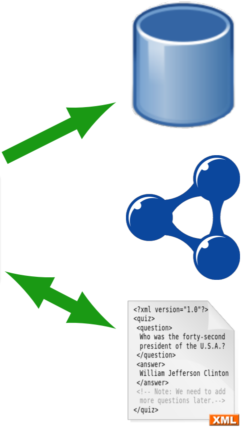


Same Language for each Format

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

XSPARQL

```
for address as $address from clients
where person = "Nuno"
return $address
```

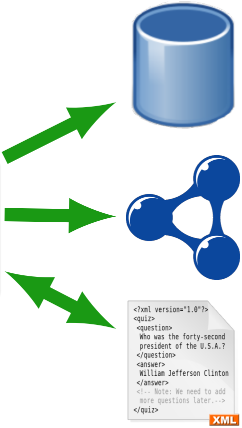


Same Language for each Format

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

XSPARQL

```
for $lat $long from <geonames.org/..>
where { $point geo:lat $lat;
              geo:long $long }
return $lat
```



Creating RDF with XSPARQL

Convert online orders into RDF

```
prefix : <http://pizza-vocab.ie/>

for $person in doc("eat.ie/..")//name
construct { :meal a :FoodOrder; :author {$person} }
```

Creating RDF with XSPARQL

Convert online orders into RDF

```
prefix : <http://pizza-vocab.ie/>

for $person in doc("eat.ie/..")//name
construct { :meal a :FoodOrder; :author {$person} }
```

`construct` clause generates RDF

Creating RDF with XSPARQL

Convert online orders into RDF

```
prefix : <http://pizza-vocab.ie/>

for $person in doc("eat.ie/..")//name
construct { :meal a :FoodOrder; :author {$person} }
```

Arbitrary XSPARQL expressions in subject, predicate, and object

Creating RDF with XSPARQL

Convert online orders into RDF

```
prefix : <http://pizza-vocab.ie/>

for $person in doc("eat.ie/..")//name
construct { :meal a :FoodOrder; :author {$person} }
```

Query result

```
@prefix : <http://pizza-vocab.ie/> .

:meal a :FoodOrder .
:meal :author "Nuno" .
```

Integration Query Example

"Display pizza deliveries in Google Maps using KML"

```
for $person in doc("eat.ie/...")//name
  for address as $address from clients
  where person = $person
    let $uri := fn:concat(
         "api.geonames.org/search?q=", $address)
    for $lat $long from $uri
    where { $point geo:lat $lat; geo:long $long }
    return <kml>
             <lat>{$lat}<lat>
             <long>{$long}</long>
           </kml>
```

Integration Query Example

“Display pizza deliveries in Google Maps using KML”


```
for $person in doc("eat.ie/...")//name
  for address as $address from clients
  where person = $person
    let $uri := fn:concat(
         "api.geonames.org/search?q=", $address)
    for $lat $long from $uri
    where { $point geo:lat $lat; geo:long $long }
    return <kml>
             <lat>{$lat}<lat>
             <long>{$long}</long>
           </kml>
```

Integration Query Example

"Display pizza deliveries in Google Maps using KML"

```
for $person in doc("eat.ie/...")//name
  for address as $address from clients
  where person = $person
    let $uri := fn:concat(
         "api.geonames.org/search?q=", $address)
    for $lat $long from $uri
    where { $point geo:lat $lat; geo:long $long }
    return <kml>
             <lat>{$lat}<lat>
             <long>{$long}</long>
           </kml>
```

Integration Query Example

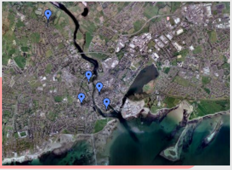
"Display pizza deliveries in Google Maps using KML"

```
for $person in doc("eat.ie/...")//name
  for address as $address from clients
  where person = $person
    let $uri := fn:concat(
         "api.geonames.org/search?q=", $address)
    for $lat $long from $uri
    where { $point geo:lat $lat; geo:long $long }
    return <kml>
             <lat>{$lat}<lat>
             <long>{$long}</long>
           </kml>
```

Integration Query Example

"Display pizza deliveries in Google Maps using KML"

```
for $person in doc("eat.ie/...")//name
  for address as $address from clients
  where person = $person
    let $uri := fn:concat(
         "api.geonames.org/search?q=", $address)
    for $lat $long from $uri
    where { $point geo:lat $lat; geo:long $long }
    return <kml>
             <lat>{$lat}<lat>
             <long>{$long}</long>
           </kml>
```



Integration Query Example

“Display pizza deliveries in Google Maps using KML”

```
for $person in doc("eat.ie/...")//name
  for address as $address from clients
  where person = $person
    let $uri := fn:concat(
         "api.geonames.org/search?q=", $address)
    for $lat $long from $uri
    where { $point geo:lat $lat; geo:long $long }
    return <kml>
             <lat>{$lat}<lat>
             <long>{$long}</long>
           </kml>
```

More involved XSPARQL queries: RDB2RDF

- Direct Mapping: ∼130 LOC
- R2RML: ∼290 LOC

Language Semantics

- Extension of the XQuery Evaluation Semantics

```
for $person in doc("eat.ie/...")//name
  for address as $address from clients
  where person = $person
    let $uri := fn:concat(
         "api.geonames.org/search?q=", $address)
    for $lat $long from $uri
    where { $point geo:lat $lat; geo:long $long }
    return <kml>
             <lat>{$lat}<lat>
             <long>{$long}</long>
           </kml>
```

Language Semantics

- Extension of the XQuery Evaluation Semantics
- `for` add variables and values to the dynamic environment

```
for $person in doc("eat.ie/...")//name
  for address as $address from clients
  where person = $person
    let $uri := fn:concat(
         "api.geonames.org/search?q=", $address)
    for $lat $long from $uri
    where { $point geo:lat $lat; geo:long $long }
    return <kml>
             <lat>{$lat}<lat>
             <long>{$long}</long>
           </kml>
```

Language Semantics

- Extension of the XQuery Evaluation Semantics
- `for` add variables and values to the dynamic environment
- Reuse semantics of original languages for the new expressions

```
for $person in doc("eat.ie/...")//name
  for address as $address from clients
  where person = $person
    let $uri := fn:concat(
         "api.geonames.org/search?q=", $address)
    for $lat $long from $uri
    where { $point geo:lat $lat; geo:long $long }
    return <kml>
             <lat>{$lat}<lat>
             <long>{$long}</long>
           </kml>
```

Language Semantics

- Extension of the XQuery Evaluation Semantics
- `for` add variables and values to the dynamic environment
- Reuse semantics of original languages for the new expressions

```
for $person in doc("eat.ie/...")//name
  for address as $address from clients
  where person = $person
    let $uri := fn:concat(
         "api.geonames.org/search?q=", $address
    for $lat $long from $uri
    where { $point geo:lat $lat; geo:long $long
    return <kml>
             <lat>{$lat}<lat>
             <long>{$long}</long>
           </kml>
```

dynEnv.varValue ⇒ relation

person
"nuno"

Language Semantics

- Extension of the XQuery Evaluation Semantics
- `for` add variables and values to the dynamic environment
- Reuse semantics of original languages for the new expressions

```
for $person in doc("eat.ie/...")//name
  for address as $address from clients
  where person = $person
    let $uri := fn:concat(
         "api.geonames.org/search?q=", $address)
    for $lat $long from $uri
    where { $point geo:lat $lat; geo:long $long }
    return <kml>
             <lat>{$lat}<lat>
             <long>{$long}</long>
           </kml>
```

eval(SQL SELECT)

$\bowtie$ dynEnv.varValue

Language Semantics

- Extension of the XQuery Evaluation Semantics
- `for` add variables and values to the dynamic environment
- Reuse semantics of original languages for the new expressions

```
for $person in doc("eat.ie/...")//name
  for address as $address from clients
  where person = $person
    let $uri := fn:concat(
         "api.geonames.org/search?q=", $address)
    for $lat $long from $uri
    where { $point geo:lat $lat; geo:long $long }
    return <kml>
             <lat>{$lat}<lat>
             <long>{$long}</long>
           </kml>
```

dynEnv.varValue ⇒ SPARQL sol. seq.
{{ person ⇒ "nuno" }}

Language Semantics

- Extension of the XQuery Evaluation Semantics
- `for` add variables and values to the dynamic environment
- Reuse semantics of original languages for the new expressions

```
for $person in doc("eat.ie/...")//name
  for address as $address from clients
  where person = $person
    let $uri := fn:concat(
         "api.geonames.org/search?q=", $address)
    for $lat $long from $uri
    where { $point geo:lat $lat; geo:long $long }
    return <kml>
             <lat>{$lat}<lat>
             <long>{$long}</long>
           </kml>
```

eval(SPARQL SELECT)
solutions compatible
dynEnv.varValue

Language Semantics

- Extension of the XQuery Evaluation Semantics
- `for` add variables and values to the dynamic environment
- Reuse semantics of original languages for the new expressions

$$
\frac{
\begin{array}{c}
\text{dynEnv.globalPosition} = (\mathit{Pos}_1, \cdots, \mathit{Pos}_j) \\
\text{dynEnv} \vdash \mathit{fs{:}dataset}(\mathit{DatasetClause}) \Rightarrow \mathit{Dataset} \\
\text{dynEnv} \;\vdash \mathit{fs{:}sparql}\left(\begin{array}{l}\mathit{Dataset}, \mathit{WhereClause}, \\ \mathit{SolutionModifier}\end{array}\right) \Rightarrow \mu_1, \ldots, \mu_m \\
\begin{array}{l}
\text{dynEnv} + \text{globalPosition}((\mathit{Pos}_1, \cdots, \mathit{Pos}_j, 1)) \;+\; \text{activeDataset}(\mathit{Dataset}) \\
\quad + \text{varValue}\left(\begin{array}{l}\mathit{Var}_1 \Rightarrow \mathit{fs{:}value}(\mu_1, \mathit{Var}_1); \\ \ldots; \\ \mathit{Var}_n \Rightarrow \mathit{fs{:}value}(\mu_1, \mathit{Var}_n)\end{array}\right) \vdash \mathit{ExprSingle} \Rightarrow \mathit{Value}_1
\end{array} \\
\vdots \\
\begin{array}{l}
\text{dynEnv} + \text{globalPosition}((\mathit{Pos}_1, \cdots, \mathit{Pos}_j, m)) \;+\; \text{activeDataset}(\mathit{Dataset}) \\
\quad + \text{varValue}\left(\begin{array}{l}\mathit{Var}_1 \Rightarrow \mathit{fs{:}value}(\mu_m, \mathit{Var}_1); \\ \ldots; \\ \mathit{Var}_n \Rightarrow \mathit{fs{:}value}(\mu_m, \mathit{Var}_n)\end{array}\right) \vdash \mathit{ExprSingle} \Rightarrow \mathit{Value}_m
\end{array}
\end{array}
}{
\text{dynEnv} \vdash \begin{array}{l}\textbf{for}\ \$\mathit{Var}_1 \cdots \$\mathit{Var}_n\ \ \mathit{DatasetClause} \\ \mathit{WhereClause}\ \ \mathit{SolutionModifier} \\ \textbf{return}\ \ \mathit{ExprSingle}\end{array} \Rightarrow \mathit{Value}_1, \ldots, \mathit{Value}_m
}
$$

- Extension of the XQuery Evaluation Semantics
- `for` add variables and values to the dynamic environment
- Reuse semantics of original languages for the new expressions

$$\frac{\begin{array}{c}
\mathrm{dynEnv.globalPosition} = (\mathit{Pos}_1, \cdots, \mathit{Pos}_j) \\
\mathrm{dynEnv} \vdash \mathit{fs{:}dataset}(\mathit{DatasetClause}) \Rightarrow \mathit{Dataset} \\
\mathrm{dynEnv} \vdash \mathit{fs{:}sparql}\left(\begin{array}{l}\mathit{Dataset}, \mathit{WhereClause}, \\ \mathit{SolutionModifier}\end{array}\right) \Rightarrow \mu_1, \ldots, \mu_m \\
\mathrm{dynEnv} + \mathrm{globalPosition}((\mathit{Pos}_1, \cdots, \mathit{Pos}_j, 1)) + \mathrm{activeDataset}(\mathit{Dataset}) \\
+ \mathrm{varValue}\left(\begin{array}{l}\mathit{Var}_1 \Rightarrow \mathit{fs{:}value}(\mu_1, \mathit{Var}_1); \\ \ldots; \\ \mathit{Var}_n \Rightarrow \mathit{fs{:}value}(\mu_1, \mathit{Var}_n)\end{array}\right) \vdash \mathit{ExprSingle} \Rightarrow \mathit{Value}_1 \\
\vdots \\
\mathrm{dynEnv} + \mathrm{globalPosition}((\mathit{Pos}_1, \cdots, \mathit{Pos}_j, m)) + \mathrm{activeDataset}(\mathit{Dataset}) \\
+ \mathrm{varValue}\left(\begin{array}{l}\mathit{Var}_1 \Rightarrow \mathit{fs{:}value}(\mu_m, \mathit{Var}_1); \\ \ldots; \\ \mathit{Var}_n \Rightarrow \mathit{fs{:}value}(\mu_m, \mathit{Var}_n)\end{array}\right) \vdash \mathit{ExprSingle} \Rightarrow \mathit{Value}_m
\end{array}}{\mathrm{dynEnv} \vdash \begin{array}{l}\textbf{for}\ \$\mathit{Var}_1 \cdots \$\mathit{Var}_n\ \ \mathit{DatasetClause} \\ \mathit{WhereClause}\ \ \mathit{SolutionModifier} \\ \textbf{return}\ \ \mathit{ExprSingle}\end{array} \Rightarrow \mathit{Value}_1, \ldots, \mathit{Value}_m}$$

Language Implementation

- Reuse components (SPARQL engine, Relational Database)

Language Implementation

- Reuse components (SPARQL engine, Relational Database)
- Implemented XSPARQL by rewriting to XQuery

Language Implementation

- Reuse components (SPARQL engine, Relational Database)
- Implemented XSPARQL by rewriting to XQuery
- Semantics implemented by substitution of bound variables

Language Implementation

- Reuse components (SPARQL engine, Relational Database)
- Implemented XSPARQL by rewriting to XQuery
- Semantics implemented by substitution of bound variables

Bound Variable Substitution

- Bound variables are replaced by their value at runtime

```
for $person in doc("eat.ie/...")//name
  for address as $address from clients
  where person = $person
```

Language Implementation

- Reuse components (SPARQL engine, Relational Database)
- Implemented XSPARQL by rewriting to XQuery
- Semantics implemented by substitution of bound variables

Bound Variable Substitution

- Bound variables are replaced by their value at runtime
- Implemented in the generated XQuery

```
for $person in doc("eat.ie/...")//name
  for address as $address from clients
  where person = $person
```

```
fn:concat(
  "SELECT address from clients
  where person = ", $person)
```

Language Implementation

- Reuse components (SPARQL engine, Relational Database)
- Implemented XSPARQL by rewriting to XQuery
- Semantics implemented by substitution of bound variables

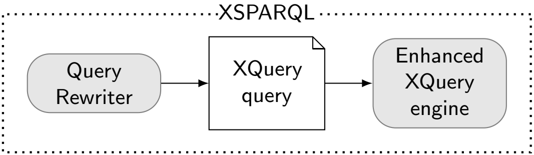
Bound Variable Substitution

- Bound variables are replaced by their value at runtime
- Implemented in the generated XQuery
- Pushing the variable bindings into the respective query engine

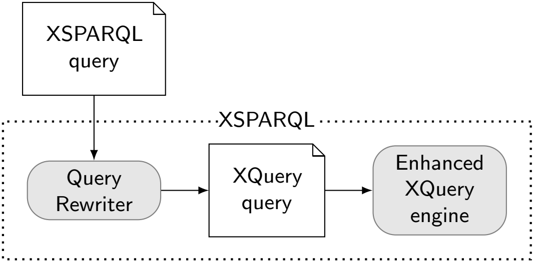
```
for $person in doc("eat.ie/...")//name
  for address as $address from clients
  where person = $person
```

```
fn:concat(
  "SELECT address from clients
  where person = ", $person)
```

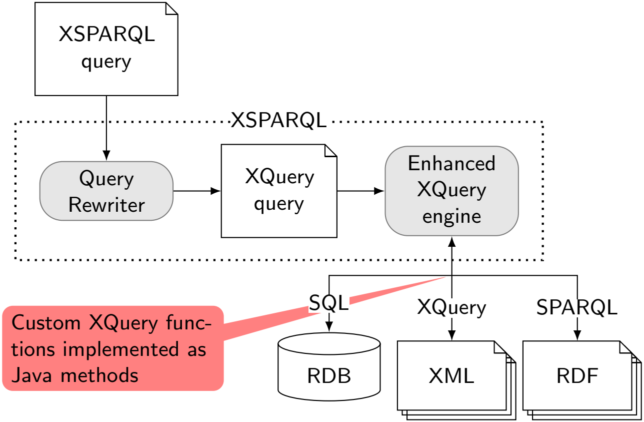
Implementation



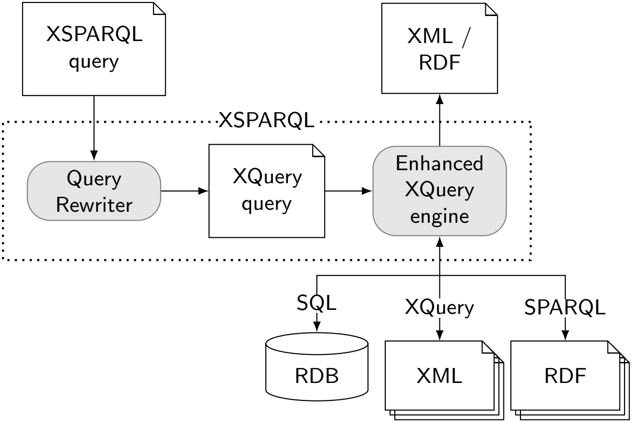
Implementation



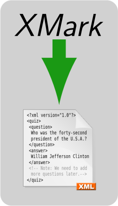
Implementation



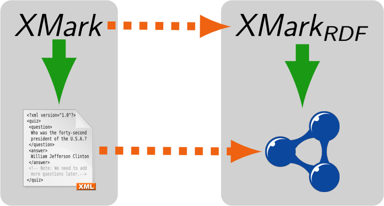
Implementation



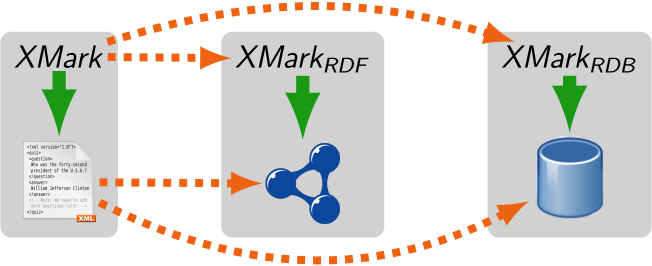
XSPARQL Evaluation

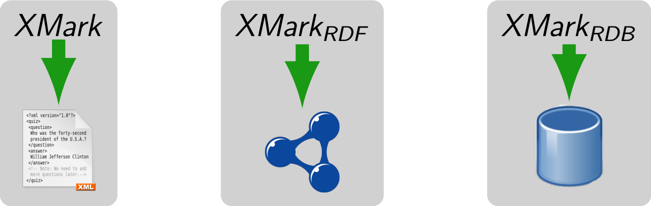


XSPARQL Evaluation



XSPARQL Evaluation



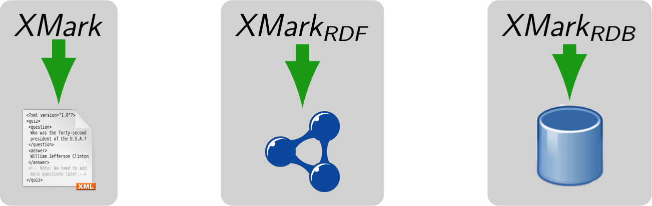


Experimental Results

- Compared to native XQuery (XMark)

  RDB and RDF: same order of magnitude for most queries

XSPARQL Evaluation



Experimental Results

- Compared to native XQuery (XMark)

  RDB and RDF: same order of magnitude for most queries
- Except on RDF nested queries (self joining data)

  several orders of magnitude slower
  due to the number of calls to the SPARQL engine

Rewriting techniques for Nested Queries

Q_8: "List the names of persons and the number of items they bought."

```
for $person $name from <input.rdf>
where { $person foaf:name $name }
return <item person="{$name}">
  {count(
     for * from <input.rdf>
     where { $ca :buyer $person .}
     return $ca
    )}
</item>
```

Rewriting techniques for Nested Queries

Q_8: "List the names of persons and the number of items they bought."

```
for $person $name from <input.rdf>
where { $person foaf:name $name }
return <item person="{$name}">
  {count(
     for * from <input.rdf>
     where { $ca :buyer $person .}
     return $ca
    )}
</item>
```

Returns $\left(\begin{array}{c} \texttt{nuno}, \\ \texttt{axel}, \\ \vdots \end{array} \right)$

Rewriting techniques for Nested Queries

Q_8: "List the names of persons and the number of items they bought."

```
for $person $name from <input.rdf>
where { $person foaf:name $name }
return <item person="{$name}">
  {count(
    for * from <input.rdf>
    where { $ca :buyer $person .}
    return $ca
   )}
</item>
```

Returns $\begin{pmatrix} \texttt{nuno}, \\ \texttt{axel}, \\ \vdots \end{pmatrix}$

```
sparqlQuery(
  fn:concat(
    "SELECT * from <input.rdf>
    where { $ca :buyer ", $person,
    " .}")
)
```

Q_8: "List the names of persons and the number of items they bought."

Returns
$$\begin{pmatrix} \texttt{nuno} \\ \texttt{axel,} \\ \vdots \end{pmatrix}$$

```
for $person $name from <input.rdf>
where { $person foaf:name $name }
return <item person="{$name}">
  {count(
     for * from <input.rdf>
     where { $ca :buyer $person .}
     return $ca
    )}
</item>
```

```
sparqlQuery(
  fn:concat(
    "SELECT * from <input.rdf>
    where { $ca :buyer ", nuno
    " .}")
)
```

Q_8: "List the names of persons and the number of items they bought."

Returns $\begin{pmatrix} \texttt{nuno,} \\ \texttt{axel} \\ \vdots \end{pmatrix}$

```
for $person $name from <input.rdf>
where { $person foaf:name $name }
return <item person="{$name}">
  {count(
     for * from <input.rdf>
     where { $ca :buyer $person .}
     return $ca
    )}
</item>
```

```
sparqlQuery(
  fn:concat(
    "SELECT * from <input.rdf>
    where { $ca :buyer ", axel
    " .}")
)
```

Rewriting techniques for Nested Queries

Q_8: "List the names of persons and the number of items they bought."

```
let $aux := sparqlQuery(
  fn:concat(
    "SELECT * from <input.rdf>
    where { $ca :buyer $person .}" )
)
```

```
for $person $name from <input.rdf>
where { $person foaf:name $name }
return <item person="{$name}">
  {count(
     for * from <input.rdf>
     where { $ca :buyer $person .}
     return $ca
    )}
</item>
```

Q_8: "List the names of persons and the number of items they bought."

```
let $aux := sparqlQuery(
  fn:concat(
    "SELECT * from <input.rdf>
    where { $ca :buyer $person .}" )
)
```

```
for $person $name from <input.rdf>
where { $person foaf:name $name }
return <item person="{$name}">
  {count(
```

Join in XQuery

```
     return $ca
    )}
</item>
```

Q_8: "List the names of persons and the number of items they bought."

```
let $aux := sparqlQuery(
  fn:concat(
    "SELECT * from <input.rdf>
    where { $ca :buyer $person .}" )
)

  for $person $name from <input.rdf>
  where { $person foaf:name $name }
  return <item person="{$name}">
    {count(
```

Join in XQuery

```
      return $ca
     )}
  </item>
```

Nested Loop rewriting

Rewriting techniques for Nested Queries

Q_8: "List the names of persons and the number of items they bought."

```
let $aux := sparqlQuery(
  fn:concat(
    "SELECT * from <input.rdf>
    where { $ca :buyer $person .}" )
)

  for $person $name from <input.rdf>
  where { $person foaf:name $name }
  return <item person="{$name}">
    {count(
```

Join in XQuery

```
      return $ca
     )}
  </item>
```

Applied to related language: SPARQL2XQuery [SAC2008]

Q_8: "List the names of persons and the number of items they bought."

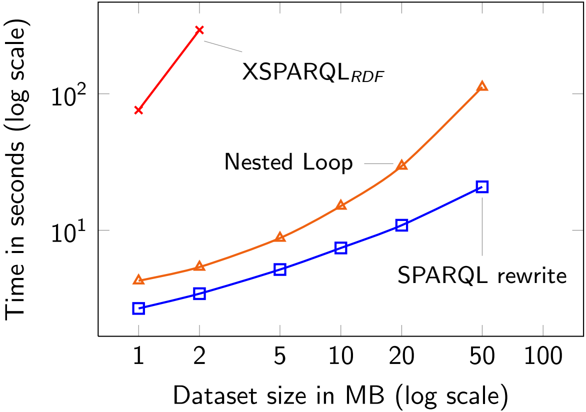
```
let $aux := sparqlQuery(
  fn:concat(
    "SELECT * from <input.rdf>
    where { $ca :buyer $person .}" )
)

  for $person $name from <input.rdf>
  where { $person foaf:name $name }
  return <item person="{$name}">
    {count(
```

Join in XQuery

```
      return $ca
     )}
  </item>
```

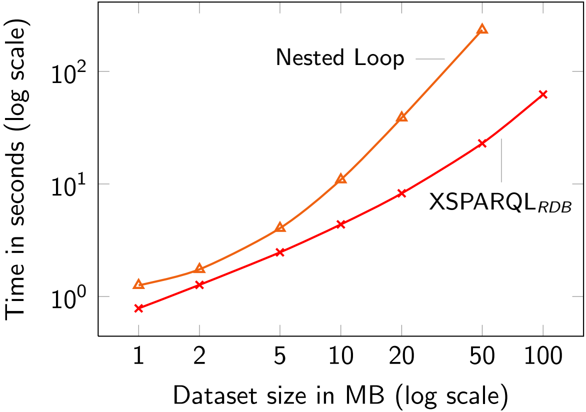
Other optimisations:
Join in SPARQL

Evaluation of different rewritings - $XMark_{RDF} Q_8$



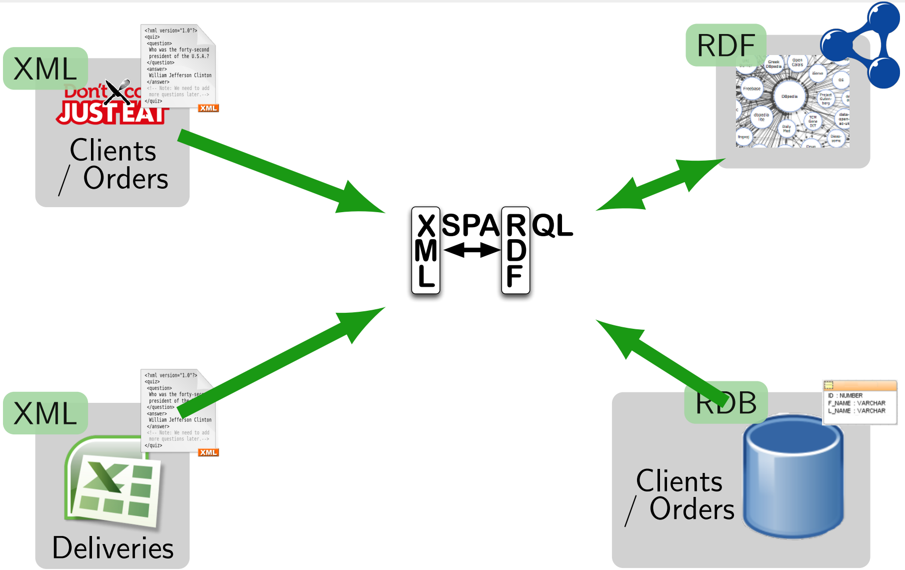
- Optimised rewritings show promising results
- Best Results: SPARQL-based
- Results included in [JoDS2012]

Evaluation of different rewritings - $XMark_{RDB}\,Q_8$

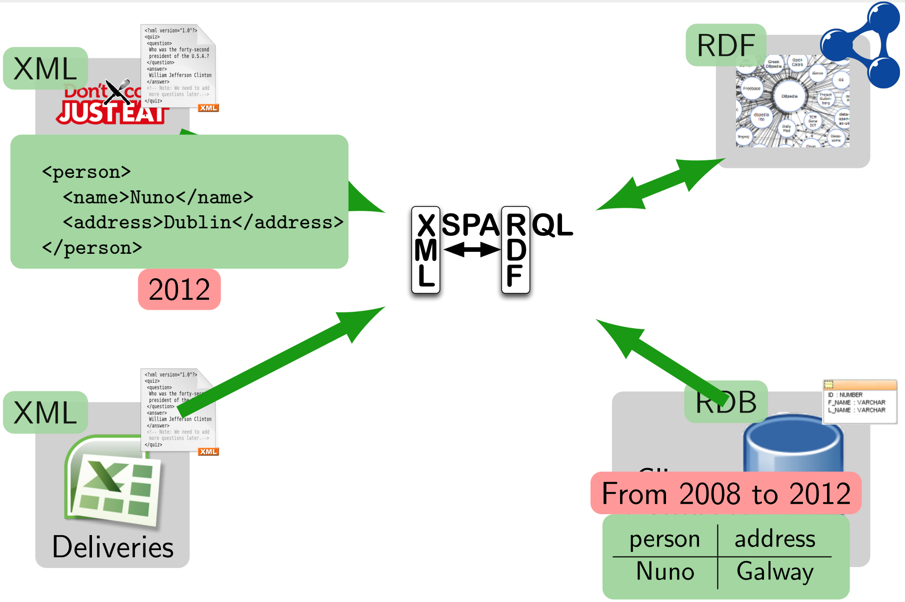


- Not so effective for RDB, requires different optimisations
- Due to schema representation?
- Additional Results for this thesis

Data Integration: Pizza Delivery Company



Data Integration: Pizza Delivery Company



Use Annotated RDF(S)!

Annotations refer to a specific **domain**

Use Annotated RDF(S)!

Annotations refer to a specific **domain**

Temporal:

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

Use Annotated RDF(S)!

Annotations refer to a specific **domain**

Temporal:

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

Fuzzy:

```
:nuno :address :Dublin .   0.9
```

Use Annotated RDF(S)!

Annotations refer to a specific **domain**

Temporal:

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

Fuzzy:

```
:nuno :address :Dublin .  0.9
```

Provenance:

```
:nuno :address :Dublin .  <http://eat.ie>
```

Use Annotated RDF(S)!

Annotations refer to a specific **domain**

Temporal:

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

Fuzzy:

```
:nuno :address :Dublin .  0.9
```

Provenance:

```
:nuno :address :Dublin .  <http://eat.ie>
```

Representation for the values of each annotation domain

Annotated RDF(S) Inference example

Inference rules are **independent** of the annotation domain

Annotated RDF(S) Inference example

Inference rules are **independent** of the annotation domain

RDFS subPropertyOf ("sp") rule:

```
   ?Prop1 sp ?Prop2 .
   ?x ?Prop1 ?y .
⇒ ?x ?Prop2 ?y .
```

```
   :address sp foaf:based_near .
   :nuno :address :Galway .
⇒ :nuno foaf:based_near :Galway .
```

Annotated RDF(S) Inference example

Inference rules are **independent** of the annotation domain

Annotated RDFS subPropertyOf ("sp") rule:

```
   ?Prop1 sp ?Prop2 .   ?v1
   ?x ?Prop1 ?y .       ?v2
⇒ ?x ?Prop2 ?y .
```

```
   :address sp foaf:based_near .      [2009,+∞]
   :nuno :address :Galway .           [2008,2012]
⇒ :nuno foaf:based_near :Galway .
```

Annotated RDF(S) Inference example

Inference rules are **independent** of the annotation domain

Annotated RDFS subPropertyOf ("sp") rule:

	?Prop1 sp ?Prop2 .	$?v1$
	?x ?Prop1 ?y .	$?v2$
$\Rightarrow$	?x ?Prop2 ?y .	$?v1 \otimes ?v2$

	:address sp foaf:based_near .	$[2009,+\infty]$
	:nuno :address :Galway .	$[2008,2012]$
$\Rightarrow$	:nuno foaf:based_near :Galway .	$[2009,+\infty] \otimes [2008,2012]$

Annotated RDF(S) Inference example

Inference rules are **independent** of the annotation domain

Annotated RDFS subPropertyOf ("sp") rule:

	?Prop1 sp ?Prop2 .	$?v1$
	?x ?Prop1 ?y .	$?v2$
$\Rightarrow$	?x ?Prop2 ?y .	$?v1 \otimes ?v2$

	:address sp foaf:based_near .	$[2009, +\infty]$
	:nuno :address :Galway .	$[2008, 2012]$
$\Rightarrow$	:nuno foaf:based_near :Galway .	$[2009, +\infty] \otimes [2008, 2012]$

Extra rule to group annotations triples ($\oplus$):

:nuno :address :Galway .	$[2009, +\infty]$
:nuno :address :Galway .	$[2008, 2012]$

Annotated RDF(S) Inference example

Inference rules are **independent** of the annotation domain

Annotated RDFS subPropertyOf ("sp") rule:

```
   ?Prop1 sp ?Prop2 .   ?v1
   ?x ?Prop1 ?y .       ?v2
⇒ ?x ?Prop2 ?y .       ?v1 ⊗ ?v2
```

```
   :address sp foaf:based_near .      [2009,+∞]
   :nuno :address :Galway .           [2008,2012]
⇒ :nuno foaf:based_near :Galway .    [2009,+∞]⊗[2008,2012]
```

Extra rule to group annotations triples (⊕):

```
   :nuno :address :Galway .    [2009,+∞]
   :nuno :address :Galway .    [2008,2012]
⇒ :nuno :address :Galway .    [2009,+∞]⊕[2008,2012]
```

Annotated RDF(S) Inference example

Inference rules are **independent** of the annotation domain

Annotated RDFS subPropertyOf ("sp") rule:

```
   ?Prop1 sp ?Prop2 .   ?v1
   ?x ?Prop1 ?y .       ?v2
⇒ ?x ?Prop2 ?y .       ?v1 ⊗ ?v2
```

```
   :address sp foaf:based_near .     [2009,+∞]
   :nuno :address :Galway .          [2008,2012]
⇒ :nuno foaf:based_near :Galway .   [2009,+∞]⊗[2008,2012]
```

Extra rule to group annotations triples (⊕):

```
⇒ :nuno :address :Galway .     [2009,+∞]⊕[2008,2012]
```

Operations on Annotations Elements

$[2009, +\infty]$ $[2008, 2012]$

Operations on Annotations Elements

$[2009, +\infty]$ $\qquad$ $[2008, 2012]$

$\otimes$

$[2009, 2012]$

$$[2009, +\infty] \qquad\qquad\qquad [2008, 2012]$$

$$\otimes$$

$$[2009, 2012]$$

$$[]$$

Operations on Annotations Elements

$$\begin{tikzcd}
 & {[2008, +\infty]} \arrow[dl, no head] \arrow[dr, no head] \arrow[d, phantom, "\oplus"] & \\
{[2009, +\infty]} \arrow[dr, no head] & & {[2008, 2012]} \arrow[dl, no head] \\
 & {[2009, 2012]} \arrow[u, phantom, "\otimes"] \arrow[d, no head, dashed] & \\
 & {[]} &
\end{tikzcd}$$

Operations on Annotations Elements

$$\begin{tikzcd}
& {[-\infty,+\infty]} \arrow[d, dashed, no head] & \\
& {[2008,+\infty]} \arrow[dl, no head] \arrow[dr, no head] \arrow[d, phantom, "\oplus"] & \\
{[2009,+\infty]} \arrow[dr, no head] & {} \arrow[d, phantom, "\otimes"] & {[2008,2012]} \arrow[dl, no head] \\
& {[2009,2012]} \arrow[d, dashed, no head] & \\
& {[]} &
\end{tikzcd}$$

Operations on Annotations Elements

$[2000, 2005]$ $[2008, 2012]$

Operations on Annotations Elements

$[2000, 2005]$ $\quad\quad$ $[2008, 2012]$

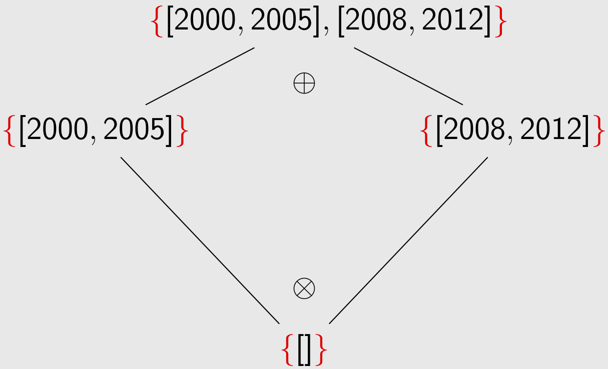
$\otimes$

$[]$

Operations on Annotations Elements

$$\begin{tikzcd}[column sep=small]
 & {[2000, 2005], [2008, 2012]} \arrow[dl, no head] \arrow[dr, no head] \arrow[d, phantom, "\oplus"] & \\
{[2000, 2005]} \arrow[dr, no head] & {} \arrow[d, phantom, "\otimes"] & {[2008, 2012]} \arrow[dl, no head] \\
 & {[]} &
\end{tikzcd}$$

Operations on Annotations Elements



AnQL: Annotated SPARQL

Extension of SPARQL Syntax

- triple pattern

Example

```
?person a foaf:Person .
```

AnQL: Annotated SPARQL

Extension of SPARQL Syntax

- triple pattern
- annotated triple pattern is a triple pattern plus
  - annotation term

Example

`?person a foaf:Person .` $[-\infty, +\infty]$

AnQL: Annotated SPARQL

Extension of SPARQL Syntax

- triple pattern
- annotated triple pattern is a triple pattern plus
  - annotation term; or
  - annotation variable

Example

```
?person :address ?address . ?l
```

AnQL: Annotated SPARQL

Extension of SPARQL Syntax

- triple pattern
- annotated triple pattern is a triple pattern plus
  - annotation term; or
  - annotation variable
- Basic Annotated Patterns (BAP) are sets of annotated triple patterns

Example

```
{ ?person a foaf:Person . [−∞, +∞]
  ?person :address ?address . ?l
}
```

AnQL: Annotated SPARQL

Extension of SPARQL Syntax

- triple pattern
- annotated triple pattern is a triple pattern plus
  - annotation term; or
  - annotation variable
- Basic Annotated Patterns (BAP) are sets of annotated triple patterns

Example

```
{ ?person a foaf:Person . [−∞, +∞]
  ?person :address ?address . ?l
}
```

Combine BAPs using AND(.), OPTIONAL, UNION, FILTER

Evaluation of SPARQL OPTIONALs

“List my address, time interval and optionally people living in the same city at the same time.”

```
SELECT   ?city ?t ?person
WHERE    { :nuno :address ?city . ?t
           OPTIONAL { ?person :address ?city . ?t } }
```

Sample input:

:nuno	:address	:Galway	.	$[2008, 2012]$
:axel	:address	:Galway	.	$[2005, 2010]$

Evaluation of SPARQL OPTIONALs

"List my address, time interval and optionally people living in the same city at the same time."

```
SELECT   ?city ?t ?person
WHERE    { :nuno :address ?city . ?t
           OPTIONAL { ?person :address ?city . ?t } }
```

Sample input:

:nuno	:address	:Galway	.	$[2008, 2012]$
:axel	:address	:Galway	.	$[2005, 2010]$

Answers:

$$S_1 = \{?city \rightarrow \text{:Galway}, ?t \rightarrow [2008, 2012]\}$$

Evaluation of SPARQL OPTIONALs

"List my address, time interval and optionally people living in the same city at the same time."

```
SELECT   ?city ?t ?person
WHERE    { :nuno :address ?city . ?t
           OPTIONAL { ?person :address ?city . ?t } }
```

Sample input:

:nuno	:address	:Galway	.	$[2008, 2012]$
:axel	:address	:Galway	.	$[2005, 2010]$

Answers:

$$S_1 = \{?city \rightarrow \text{:Galway}, ?t \rightarrow [2008, 2012]\}$$

$$S_2 = \{?city \rightarrow \text{:Galway}, ?t \rightarrow [2008, 2010],\ ?person \rightarrow \text{:axel}\}$$

Evaluation of SPARQL OPTIONALs

"List my address, time interval and optionally people living in the same city at the same time."

```
SELECT   ?city ?t ?person
WHERE    { :nuno :address ?city . ?t
           OPTIONAL { ?person :address ?city . ?t } }
```

Sample input:

:nuno	:address	:Galway	.	$[2008, 2012]$
:axel	:address	:Galway	.	$[2005, 2010]$

Answers:

$$S_1 = \{?city \rightarrow \text{:Galway}, ?t \rightarrow [2008, 2012]\}$$

$$S_2 = \{?city \rightarrow \text{:Galway}, ?t \rightarrow [2008, 2010],\ ?person \rightarrow \text{:axel}\}$$

OPTIONAL provide more information maybe restricting annotation values

Annotated RDF(S) Domains

Temporal:

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

Fuzzy:

```
:nuno :address :Galway .   0.9
```

Provenance:

```
:nuno :address :Dublin .   http://eat.ie
```

Annotated RDF(S) Domains

Temporal:

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

Fuzzy:

```
:nuno :address :Galway .   0.9
```

Provenance:

```
:nuno :address :Dublin .   http://eat.ie
```

Combining domains: Included in [JWS2012]

```
:nuno :address :Dublin .   (http://eat.ie, [2012,2012])
```

Annotated RDF(S) Domains

Temporal:

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

Fuzzy:

```
:nuno :address :Galway .  0.9
```

Provenance:

```
:nuno :address :Dublin .  http://eat.ie
```

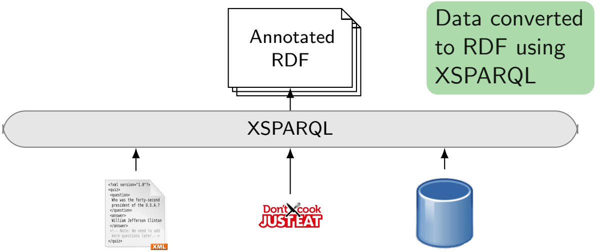
Combining domains: Included in [JWS2012]

```
:nuno :address :Dublin .  (http://eat.ie, [2012,2012])
```

Access Control: Presented at [ICLP2012]

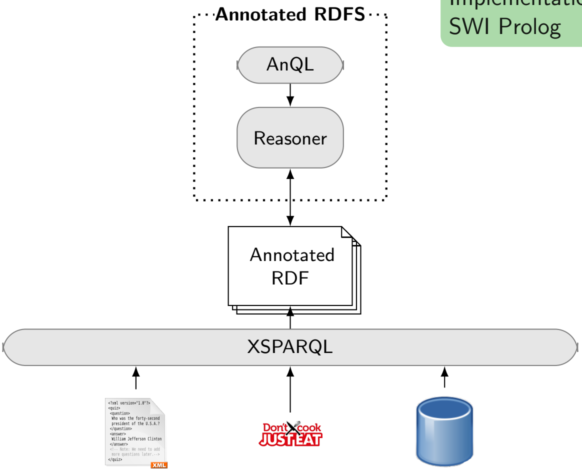
```
:nuno :address :Galway .  [nl]
```

Architecture combining XSPARQL and AnQL



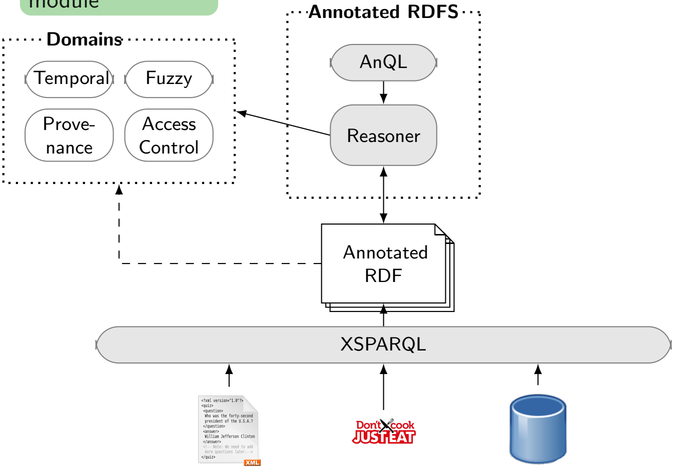
Architecture combining XSPARQL and AnQL

Implementation in SWI Prolog

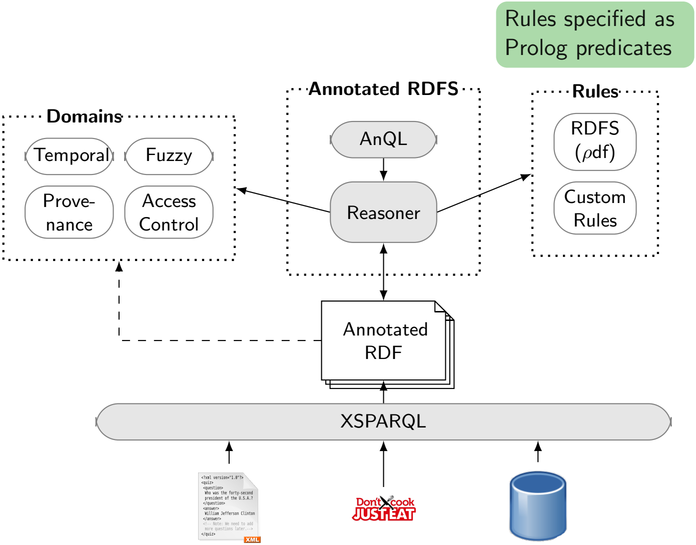


Architecture combining XSPARQL and AnQL

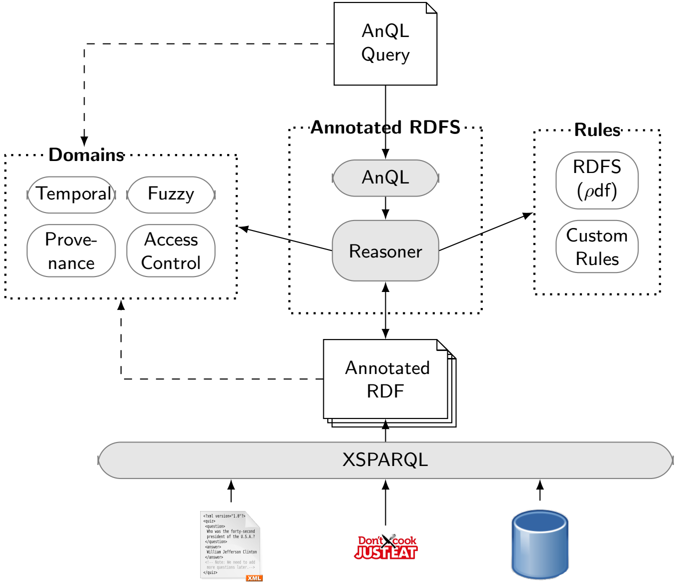
Each domain is a different Prolog module



Architecture combining XSPARQL and AnQL



Architecture combining XSPARQL and AnQL



Conclusions

Efficient data integration over heterogenous data sources can be achieved by

Conclusions

Efficient data integration over heterogenous data sources can be achieved by

1. a combined query language that accesses heterogenous data in its original sources
2. optimisations for efficient query evaluation for this language
3. an RDF-based format with support for context information

Conclusions

Efficient data integration over heterogenous data sources can be achieved by

1. a combined query language that accesses heterogenous data in its original sources
    - XSPARQL can integrate heterogeneous sources
2. optimisations for efficient query evaluation for this language
3. an RDF-based format with support for context information

Conclusions

Efficient data integration over heterogenous data sources can be achieved by

1. a combined query language that accesses heterogenous data in its original sources
   - XSPARQL can integrate heterogeneous sources
2. optimisations for efficient query evaluation for this language
   - rewriting techniques for nested queries for our implementation of XSPARQL
3. an RDF-based format with support for context information

Conclusions

Efficient data integration over heterogenous data sources can be achieved by

1. a combined query language that accesses heterogenous data in its original sources
   - XSPARQL can integrate heterogeneous sources
2. optimisations for efficient query evaluation for this language
   - rewriting techniques for nested queries for our implementation of XSPARQL
3. an RDF-based format with support for context information
   - Annotated RDFS: inferences and query over context information
   - Use XSPARQL to create Annotated RDF representing the integrated data

Conclusions

Efficient data integration over heterogenous data sources can be achieved by

1. a combined query language that accesses heterogenous data in its original sources
   - XSPARQL can integrate heterogeneous sources
2. optimisations for efficient query evaluation for this language
   - rewriting techniques for nested queries for our implementation of XSPARQL
3. an RDF-based format with support for context information
   - Annotated RDFS: inferences and query over context information
   - Use XSPARQL to create Annotated RDF representing the integrated data

Thank you! Questions?

Bibliography I

- Stefan Bischof, Stefan Decker, Thomas Krennwallner, Nuno Lopes, and Axel Polleres.
  Mapping between RDF and XML with XSPARQL.
  Journal on Data Semantics, 1:147–185, 2012.

- Sven Groppe, Jinghua Groppe, Volker Linnemann, Dirk Kukulenz, Nils Hoeller, and Christoph Reinke.
  Embedding SPARQL into XQuery/XSLT.
  In Roger L. Wainwright and Hisham Haddad, editors, *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, March 16-20, 2008*, pages 2271–2278. ACM, 2008.

- Nuno Lopes, Stefan Bischof, Stefan Decker, and Axel Polleres.
  On the Semantics of Heterogeneous Querying of Relational, XML and RDF Data with XSPARQL.
  In Paulo Moura and Vitor Beires Nogueira, editors, *Proceedings of the 15th Portuguese Conference on Artificial Intelligence (EPIA2011) – Computational Logic with Applications Track*, Lisbon, Portugal, October 2011.

- Nuno Lopes, Sabrina Kirrane, Antoine Zimmermann, Axel Polleres, and Alessandra Mileo.
  A Logic Programming approach for Access Control over RDF.
  In Agostino Dovier and Vítor Santos Costa, editors, *Technical Communications of the 28th International Conference on Logic Programming (ICLP'12)*, volume 17 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 381–392, Dagstuhl,

Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Nuno Lopes, Axel Polleres, Umberto Straccia, and Antoine Zimmermann.
AnQL: SPARQLing Up Annotated RDFS.
In *International Semantic Web Conference (1)*, pages 518–533, 2010.

Umberto Straccia, Nuno Lopes, Gergely Lukácsy, and Axel Polleres.
A General Framework for Representing and Reasoning with Annotated Semantic Web Data.
In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, July 2010.

Antoine Zimmermann, Nuno Lopes, Axel Polleres, and Umberto Straccia.
A General Framework for Representing, Reasoning and Querying with Annotated Semantic Web Data.
Web Semantics: Science, Services and Agents on the World Wide Web, 11(0):72–95, 2012.