Towards Extending and using SPARQL for Modular Document Generation

Faisal Alkhateeb and Sébastien Laborie

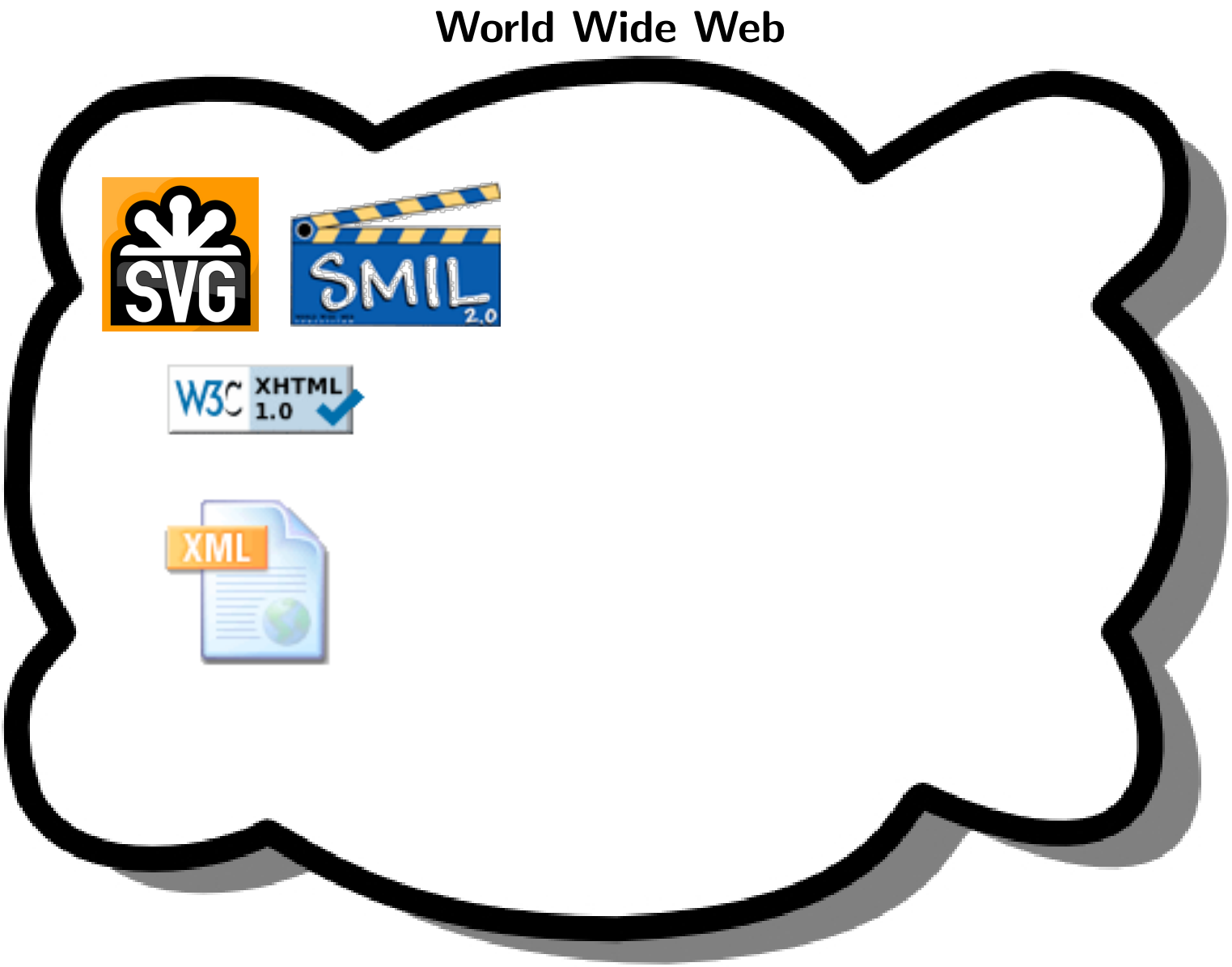
DocEng'08

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

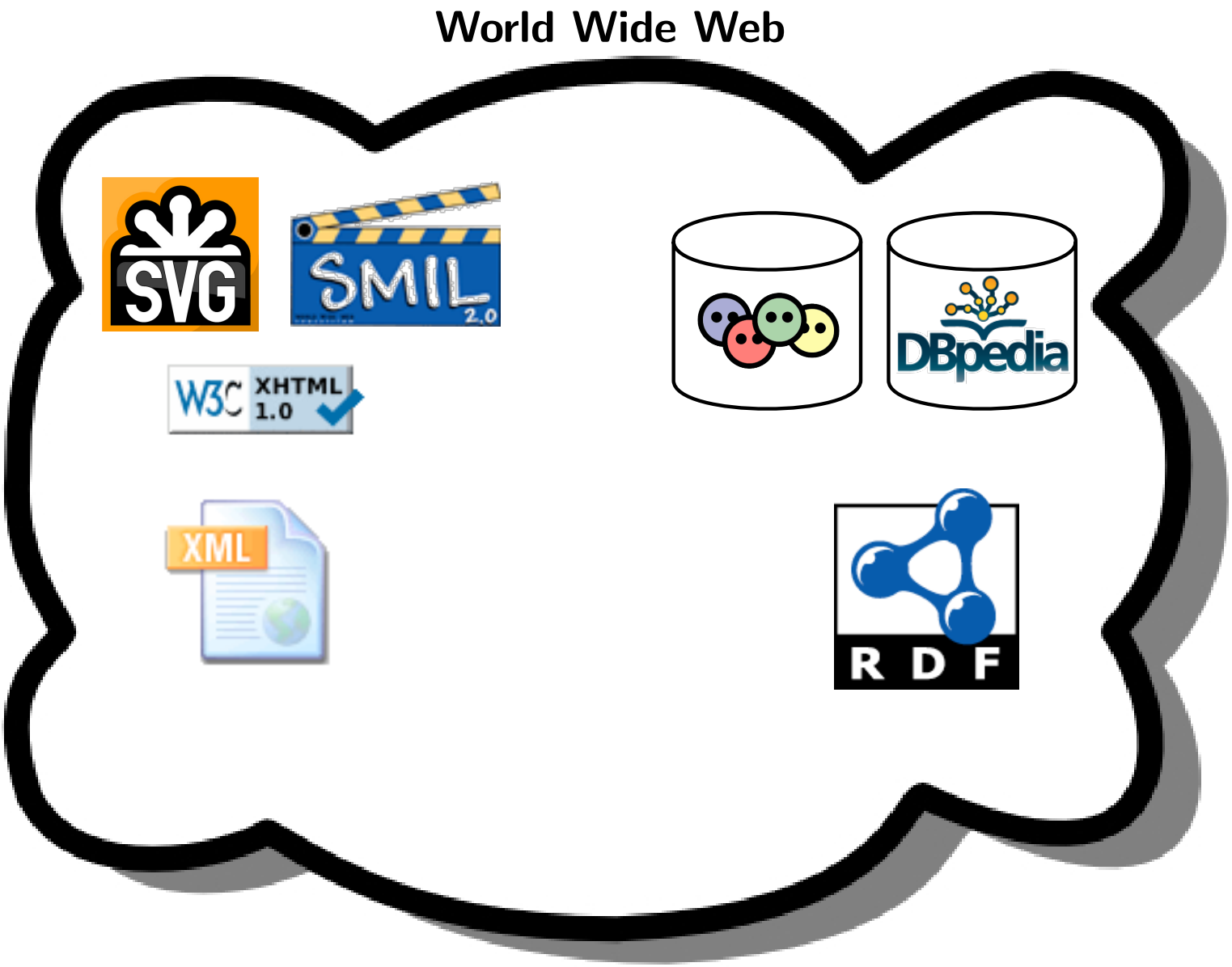
INRIA

centre de recherche
GRENOBLE - RHÔNE-ALPES

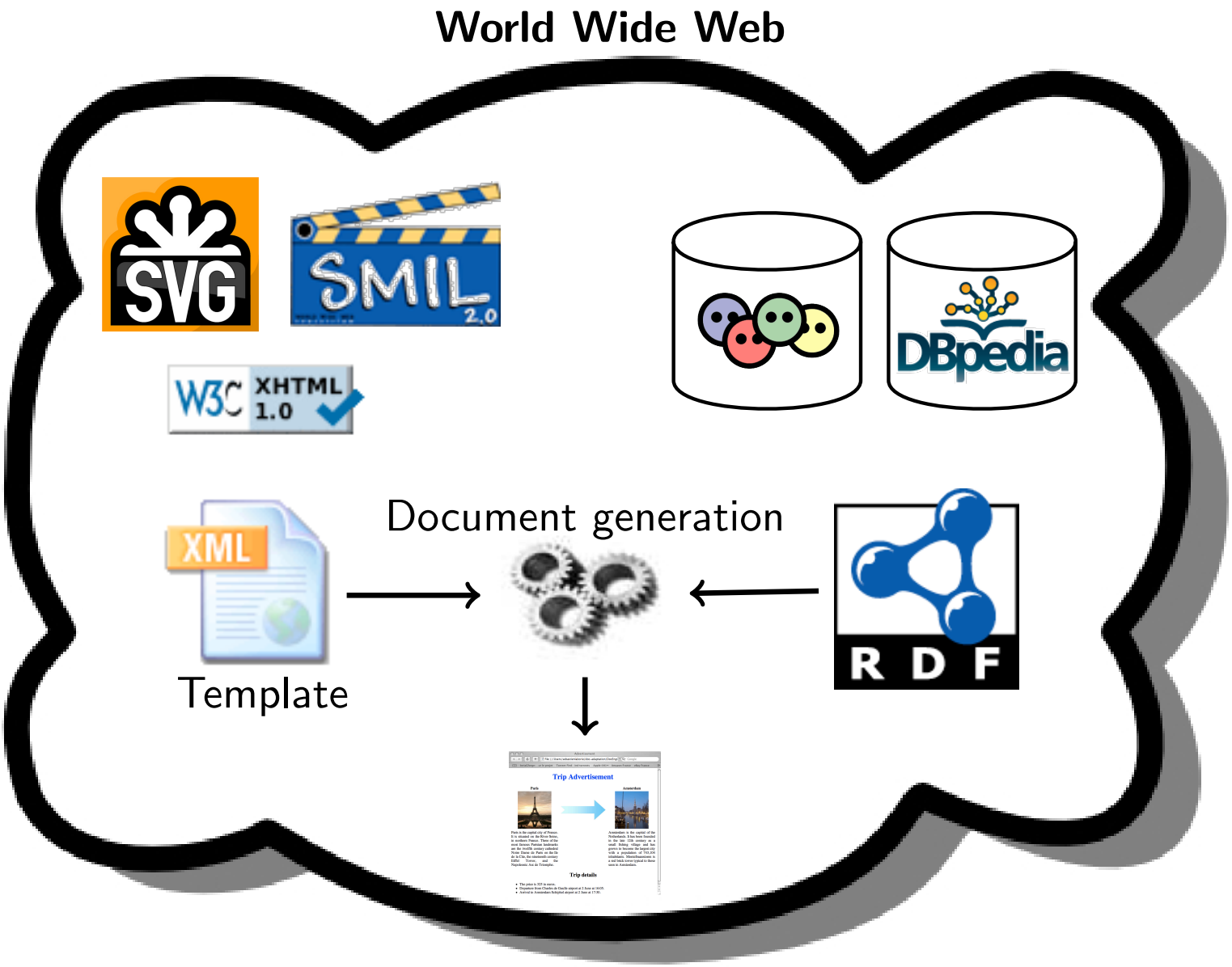
The context



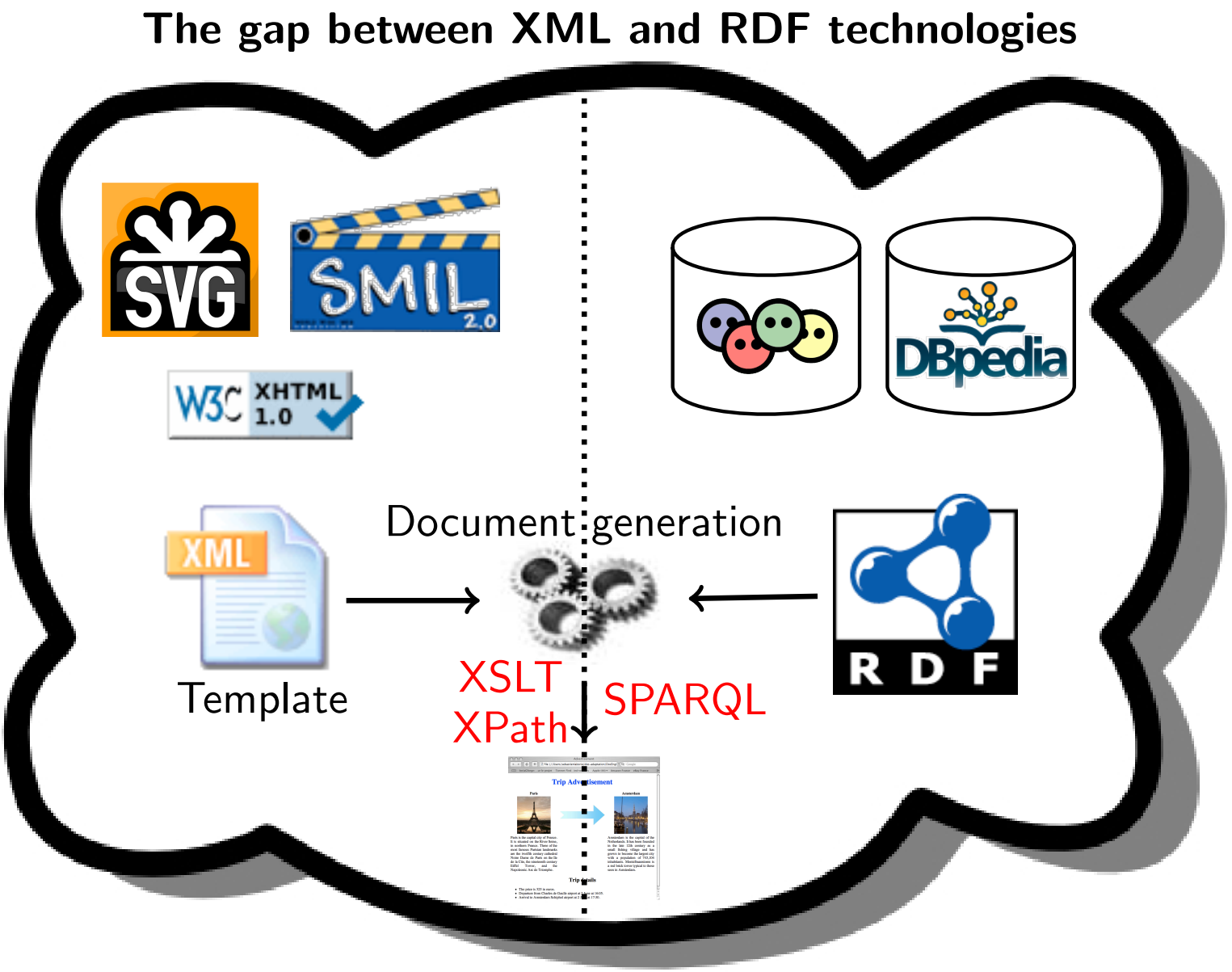
The context



The context

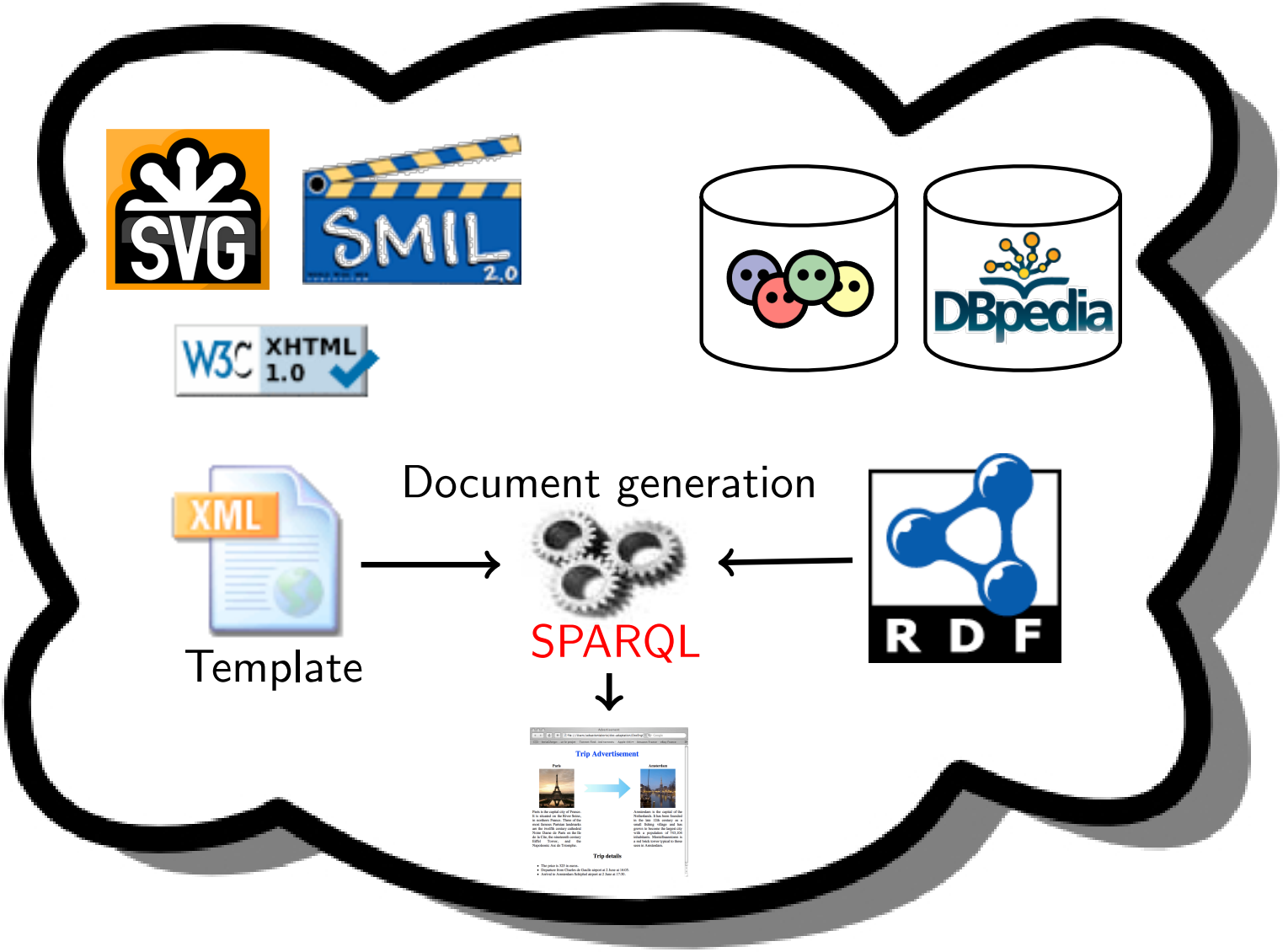

World Wide Web
SVG
SMIL 2.0
W3C XHTML 1.0
DBpedia
XML
Template
Document generation
RDF
Trip Advertisement
Paris
Amsterdam
Trip details

The problem


The gap between XML and RDF technologies
SVG
SMIL 2.0
W3C XHTML 1.0
DBpedia
XML
Template
Document generation
XSLT
XPath
SPARQL
RDF

Our contribution

Bridge the gap between XML and RDF technologies



Our contribution

Contribution:

- **A SPARQL extension** which supports the generation of XML documents based on templates
- **Definition of a template language** based on our SPARQL extension

Advantages:

- Templates modularity
- Templates reusability
- Templates readability

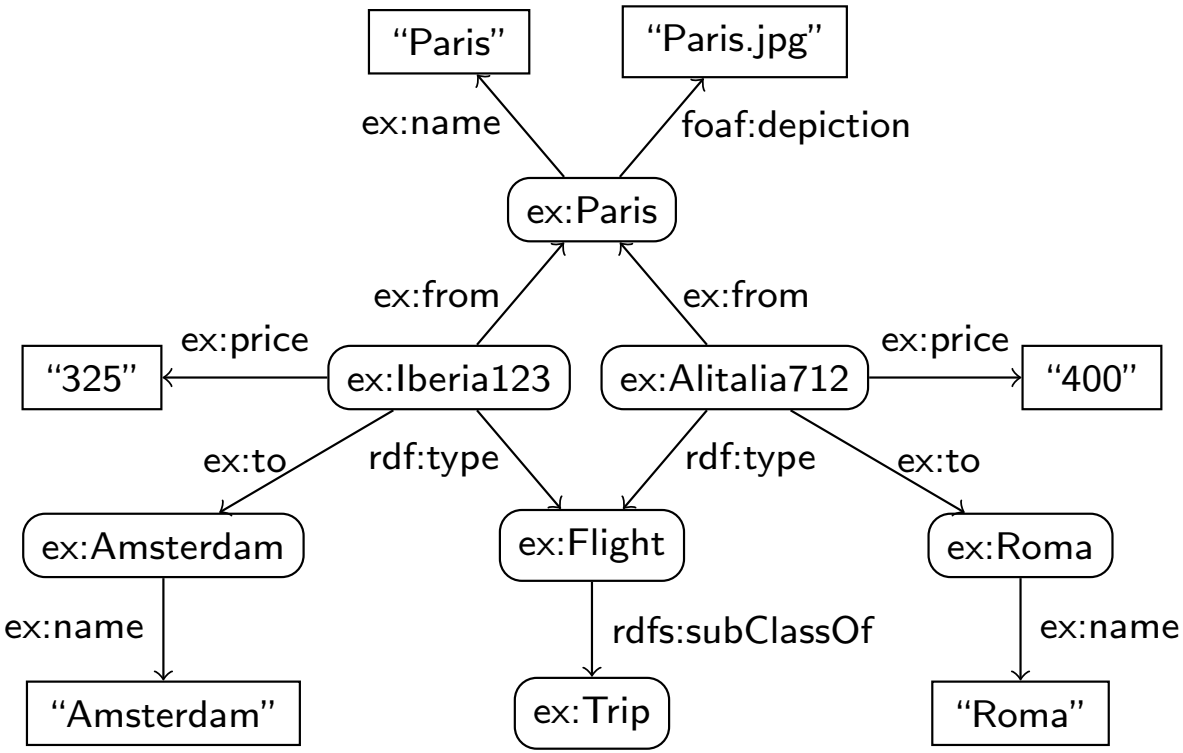
Outline

Outline

The SPARQL Language

Query: Look for flights from Paris which cost a certain price

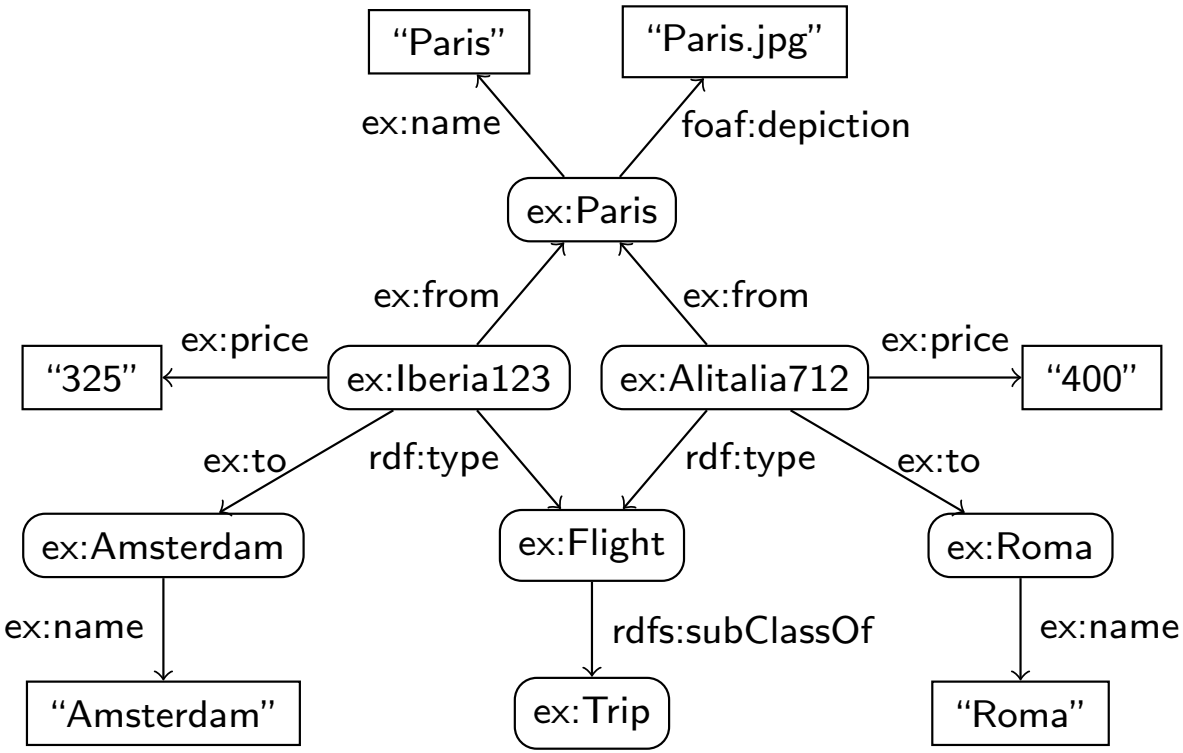
RDF Graph



The SPARQL Language

Query: Look for flights from Paris which cost a certain price

RDF Graph



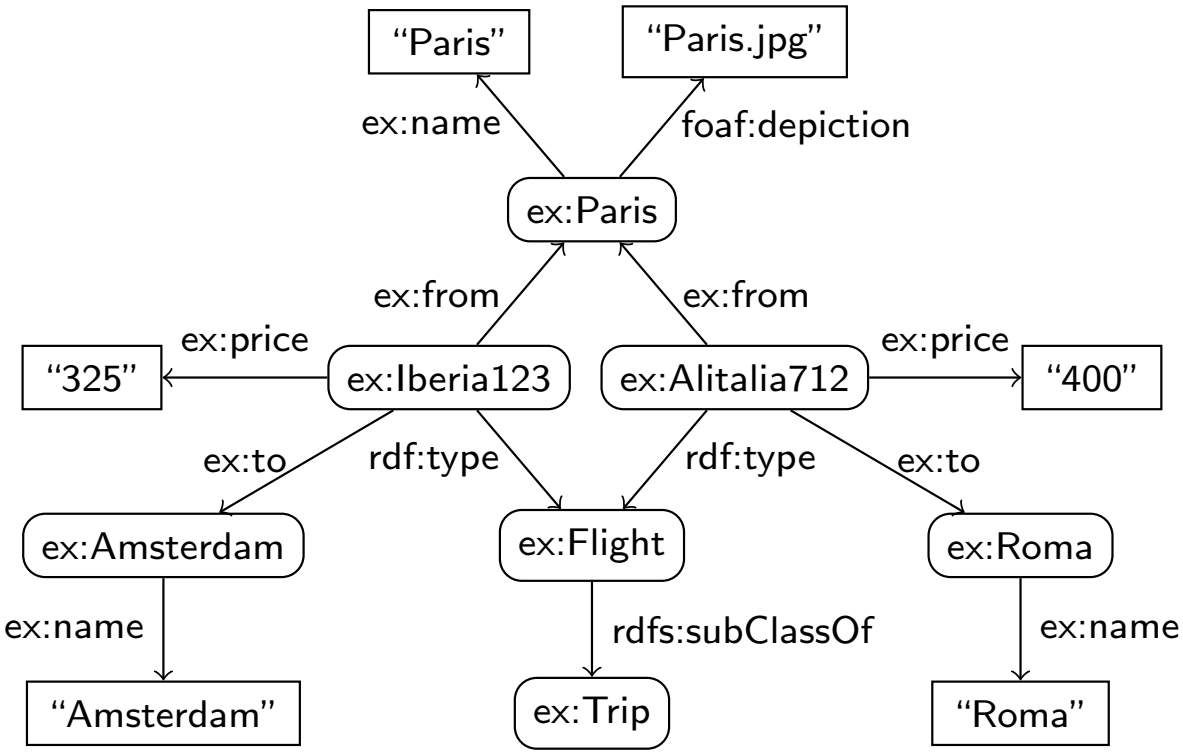
SPARQL Query

```
SELECT *
FROM RDF Graph
WHERE {
    ?Flight rdf:type ex:Flight .
    ?Flight ex:from ex:Paris .
    ?Flight ex:to ?CityArrival .
    ?Flight ex:price ?Price .
}
```

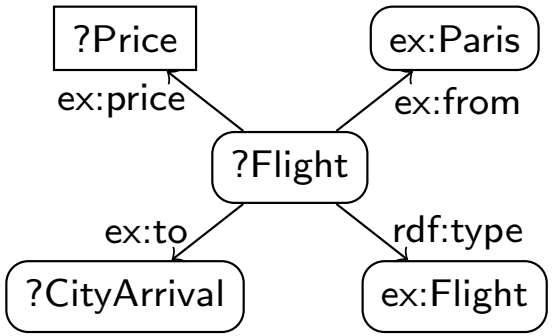
The SPARQL Language

Query: Look for flights from Paris which cost a certain price

RDF Graph



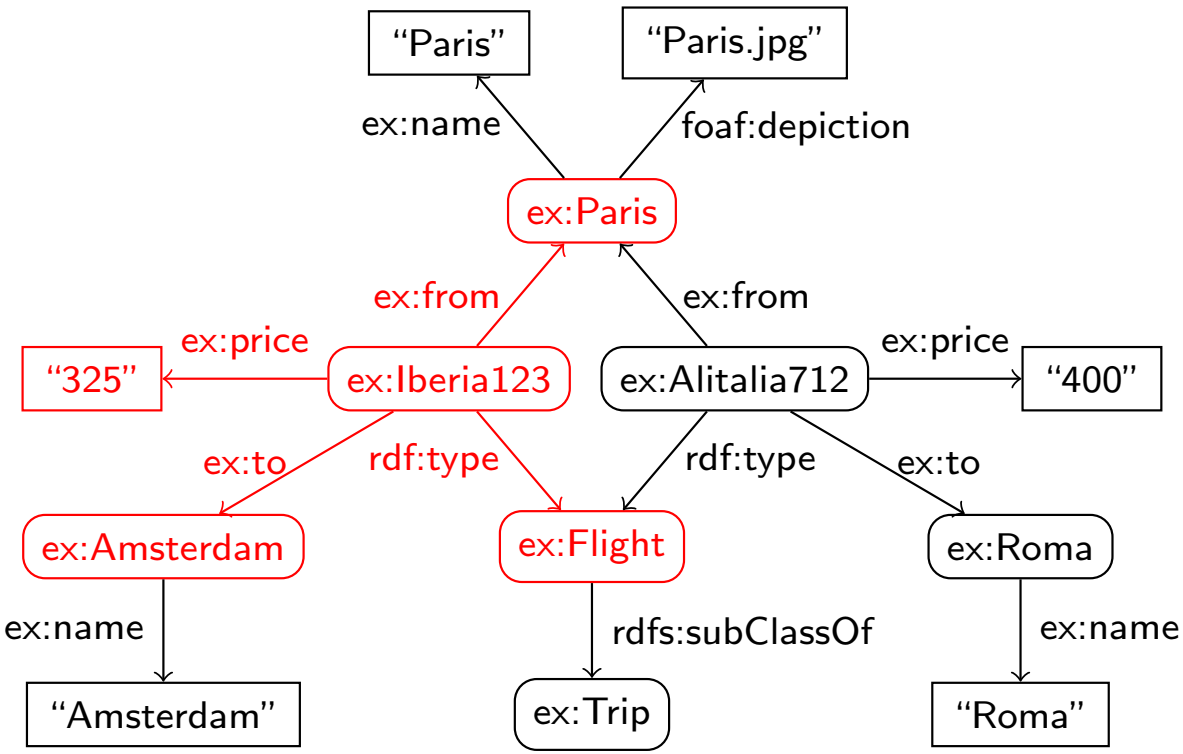
SPARQL Query



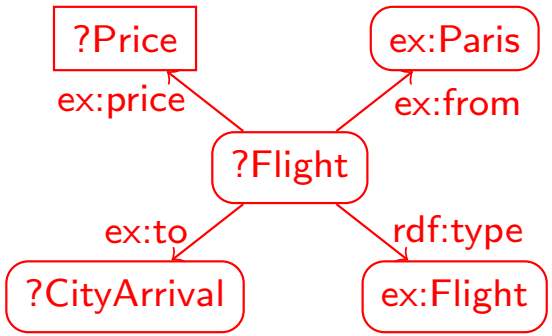
The SPARQL Language

Query: Look for flights from Paris which cost a certain price

RDF Graph



SPARQL Query

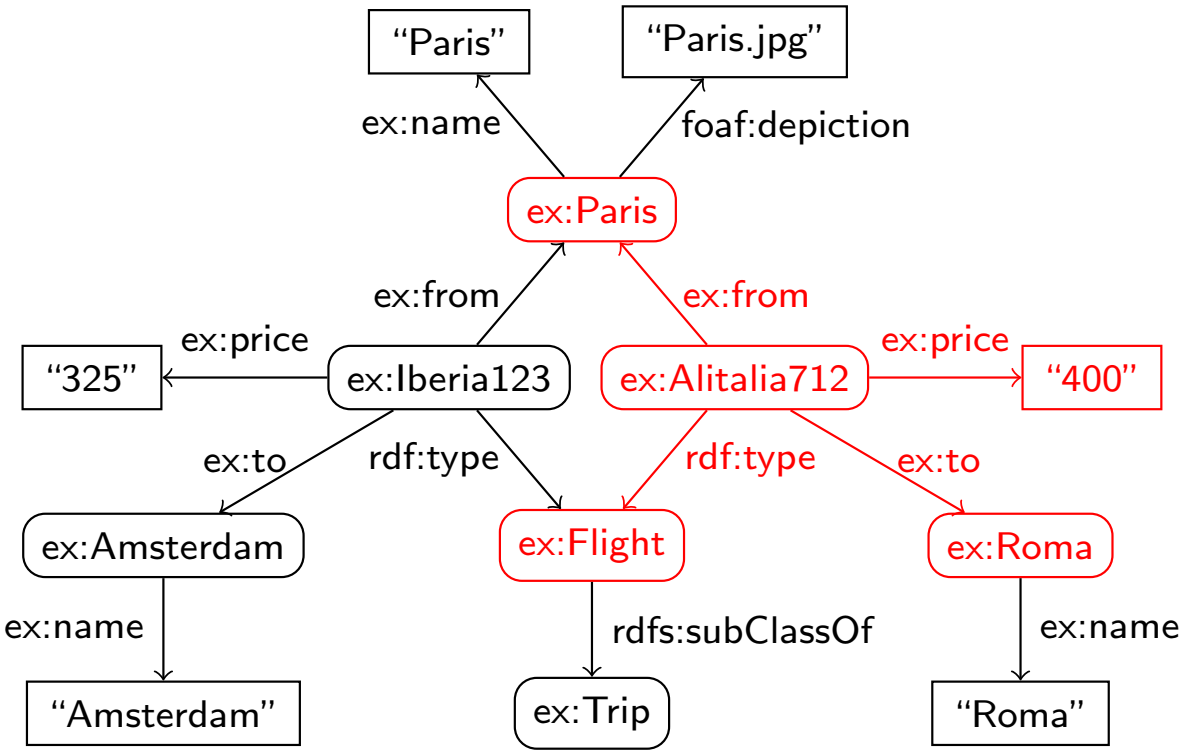


Solutions	*?Flight*	*?CityArrival*	*?Price*
1	ex:Iberia123	ex:Amsterdam	"325"

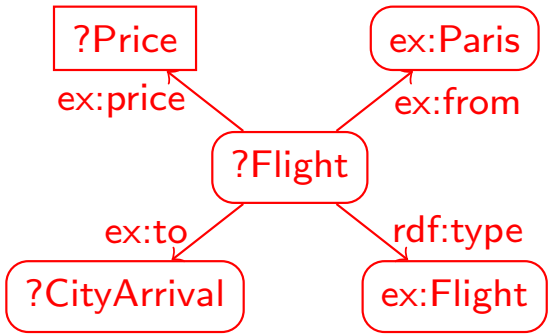
The SPARQL Language

Query: Look for flights from Paris which cost a certain price

RDF Graph



SPARQL Query

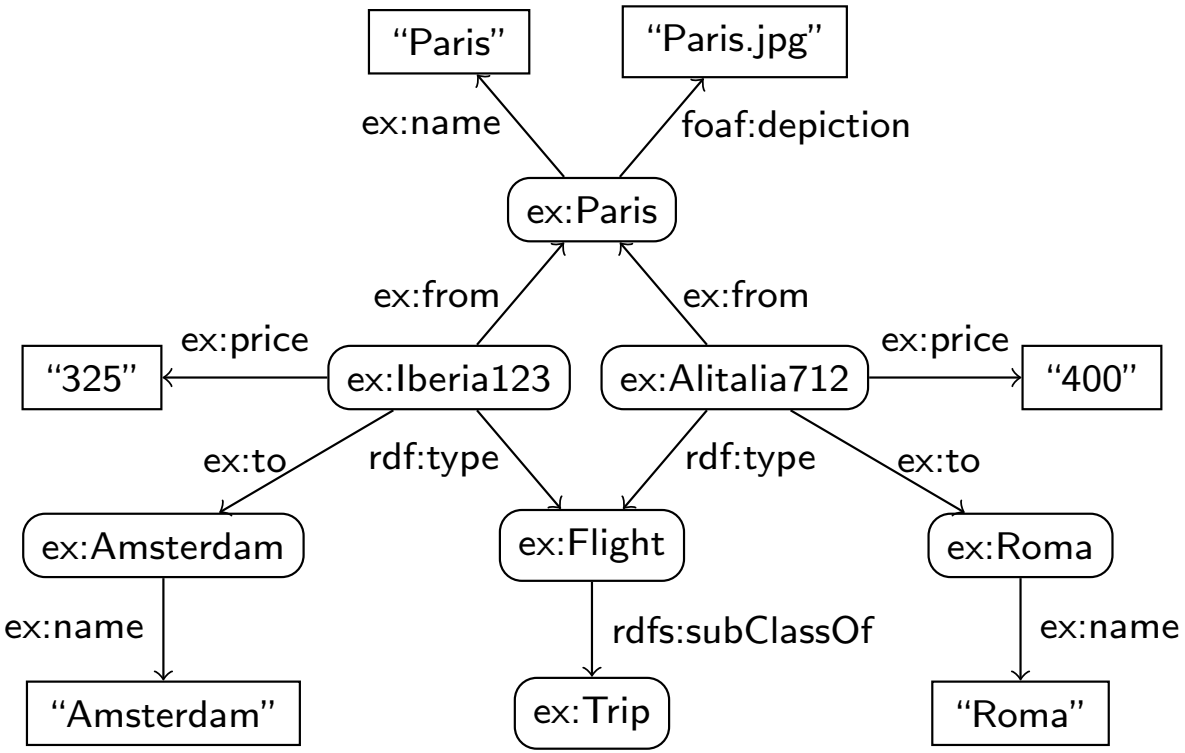


Solutions	*?Flight*	*?CityArrival*	*?Price*
1	ex:Iberia123	ex:Amsterdam	"325"
2	ex:Alitalia712	ex:Roma	"400"

The SPARQL Language with RDFS

Query: Look for trips from Paris which cost a certain price

RDF Graph



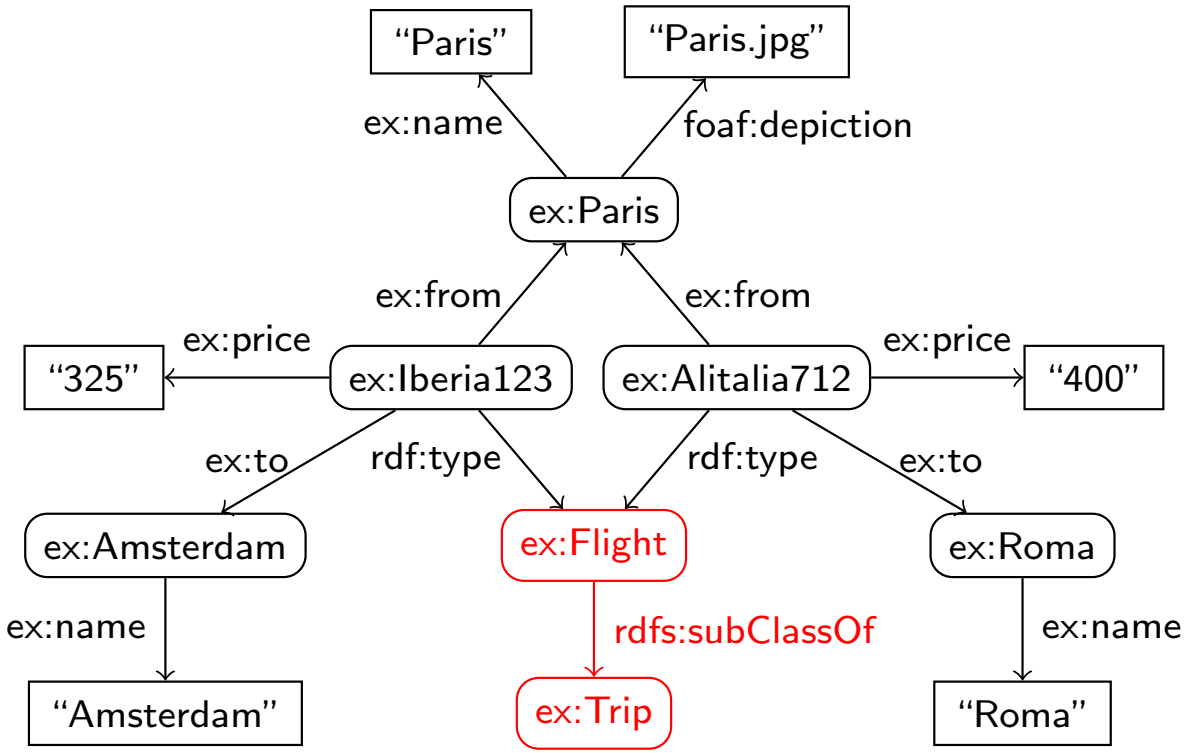
SPARQL Query

```
SELECT *
FROM RDF Graph
WHERE {
        ?Trip rdf:type ex:Trip .
        ?Trip ex:from ex:Paris .
        ?Trip ex:to ?CityArrival .
        ?Trip ex:price ?Price .
}
```

The SPARQL Language with RDFS

Query: Look for trips from Paris which cost a certain price

RDF Graph



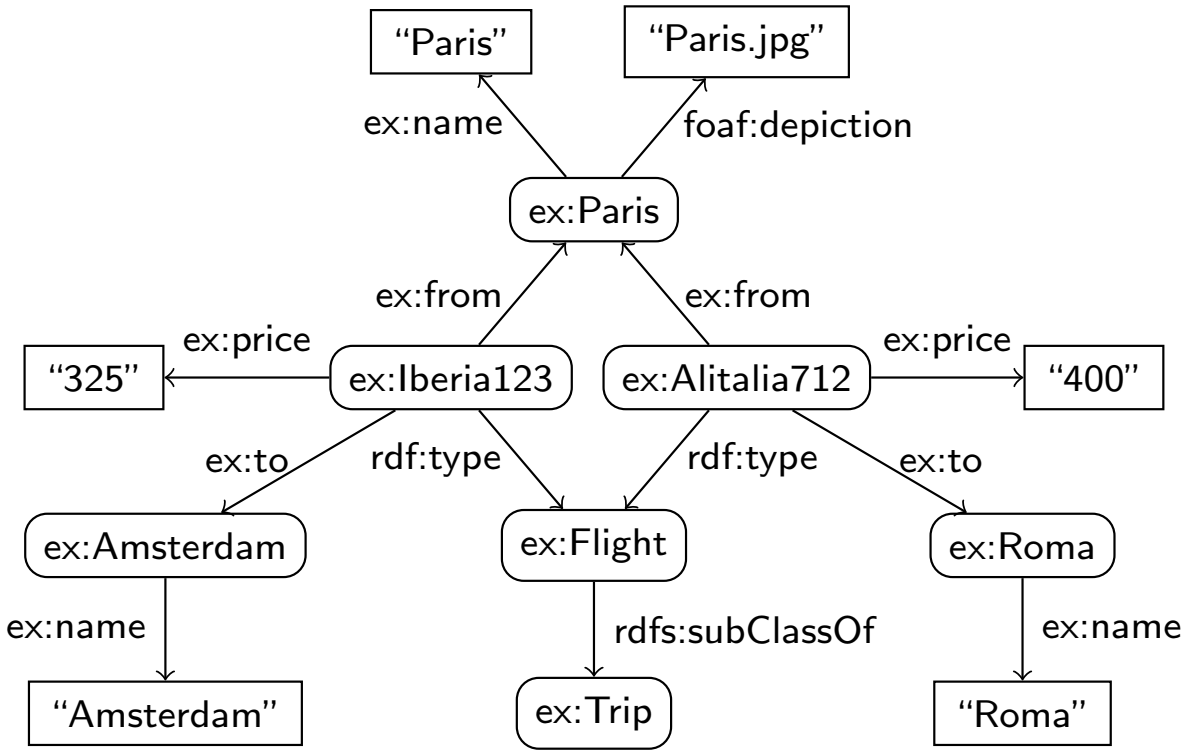
SPARQL Query

```
SELECT *
FROM RDF Graph
WHERE {
        ?Trip rdf:type ex:Trip .
        ?Trip ex:from ex:Paris .
        ?Trip ex:to ?CityArrival .
        ?Trip ex:price ?Price .
}
```

The SPARQL Language with RDFS

Query: Look for trips from Paris which cost a certain price

RDF Graph



SPARQL Query

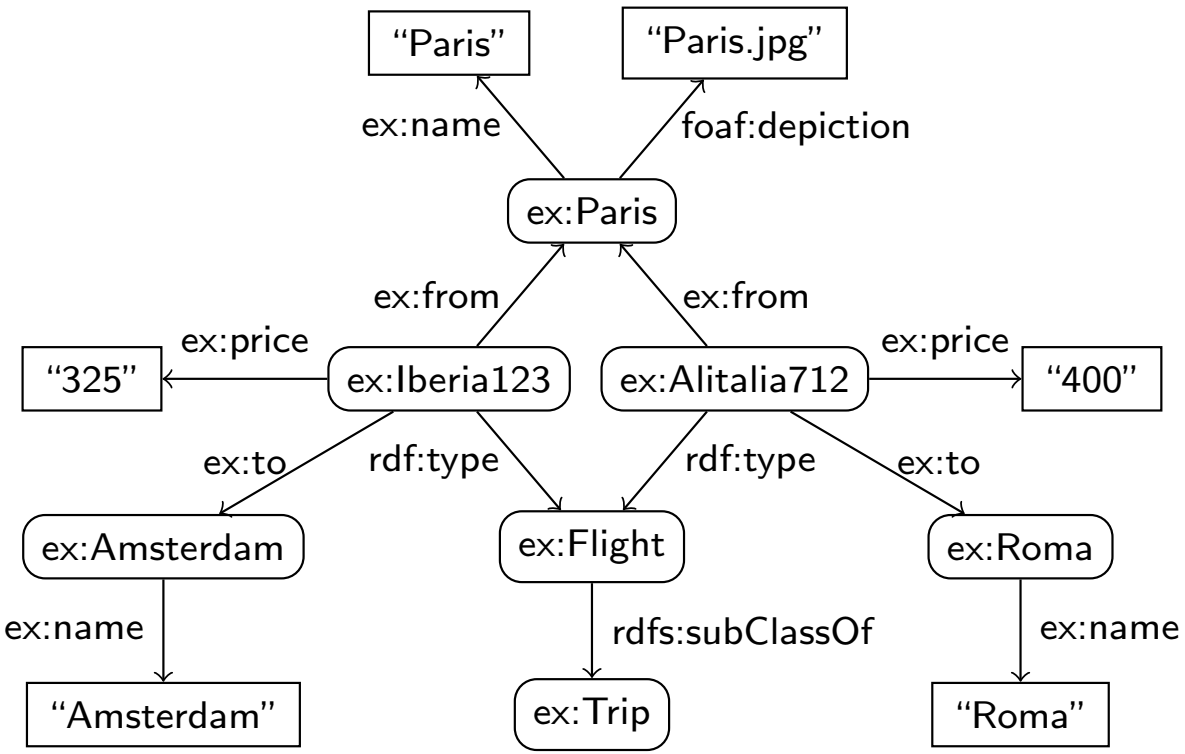
```
SELECT *
FROM RDF Graph
WHERE {
      ?Trip rdf:type ex:Trip .
      ?Trip ex:from ex:Paris .
      ?Trip ex:to ?CityArrival .
      ?Trip ex:price ?Price .
}
```

Solutions	*?Trip*	*?CityArrival*	*?Price*
1	ex:Iberia123	ex:Amsterdam	"325"
2	ex:Alitalia712	ex:Roma	"400"

The SPARQL Language with a CONSTRUCT clause

Query: Look for trips from Paris which cost a certain price

RDF Graph



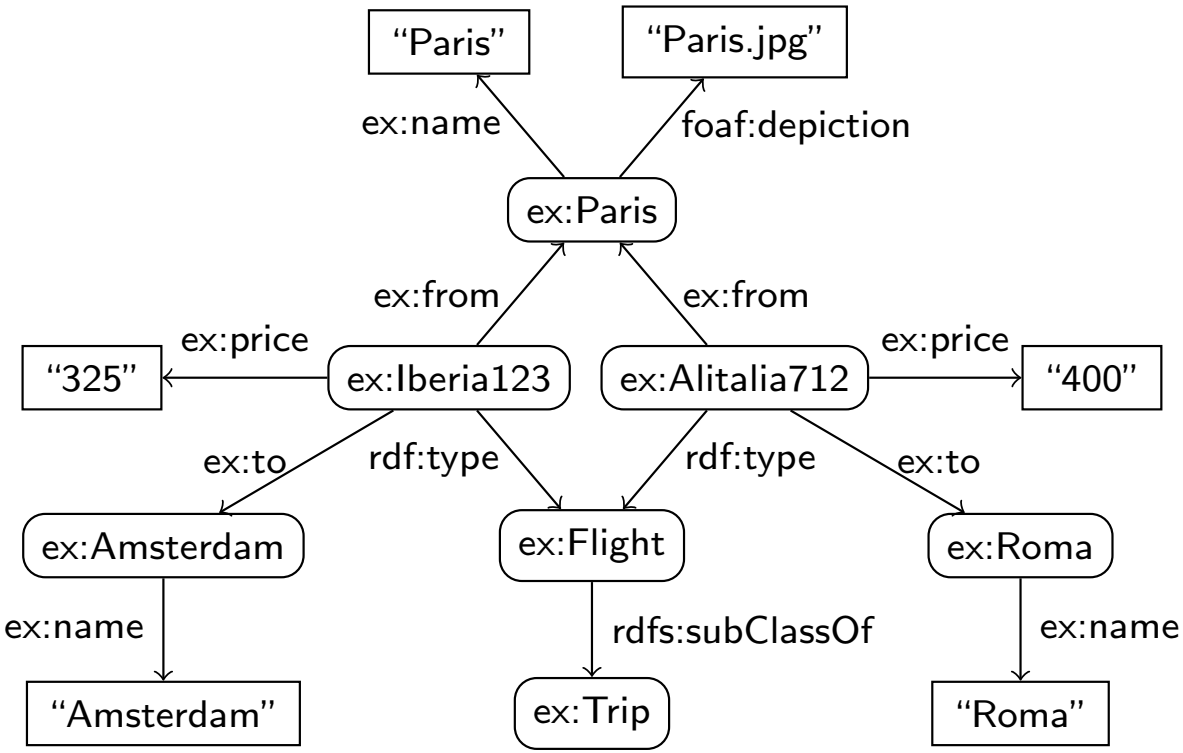
SPARQL Query

```
CONSTRUCT
    {ex:Paris ex:reachable ?CityArrival .}
FROM RDF Graph
WHERE {
      ?Trip rdf:type ex:Trip .
      ?Trip ex:from ex:Paris .
      ?Trip ex:to ?CityArrival .
      ?Trip ex:price ?Price .
}
```

The SPARQL Language with a CONSTRUCT clause

Query: Look for trips from Paris which cost a certain price

RDF Graph



SPARQL Query

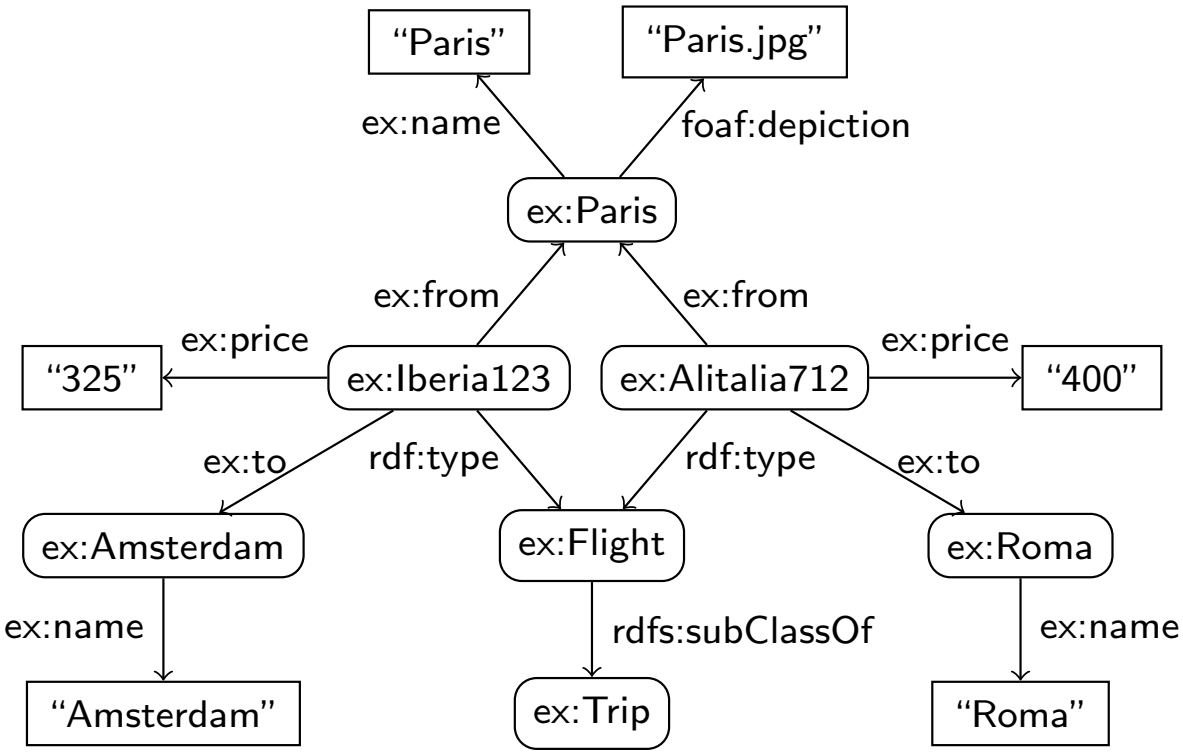
```
CONSTRUCT
    {ex:Paris ex:reachable ?CityArrival .}
FROM RDF Graph
WHERE {
      ?Trip rdf:type ex:Trip .
      ?Trip ex:from ex:Paris .
      ?Trip ex:to ?CityArrival .
      ?Trip ex:price ?Price .
}
```

Query answer

The SPARQL Language with a CONSTRUCT clause

Query: Look for trips from Paris which cost a certain price

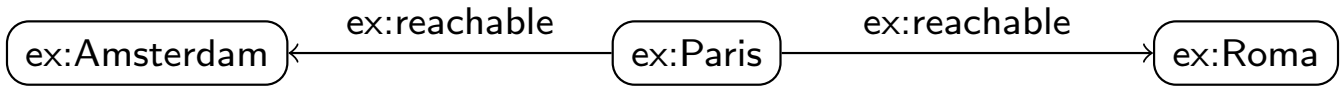
RDF Graph



SPARQL Query

```
CONSTRUCT
    {ex:Paris ex:reachable ?CityArrival .}
FROM RDF Graph
WHERE {
      ?Trip rdf:type ex:Trip .
      ?Trip ex:from ex:Paris .
      ?Trip ex:to ?CityArrival .
      ?Trip ex:price ?Price .
}
```

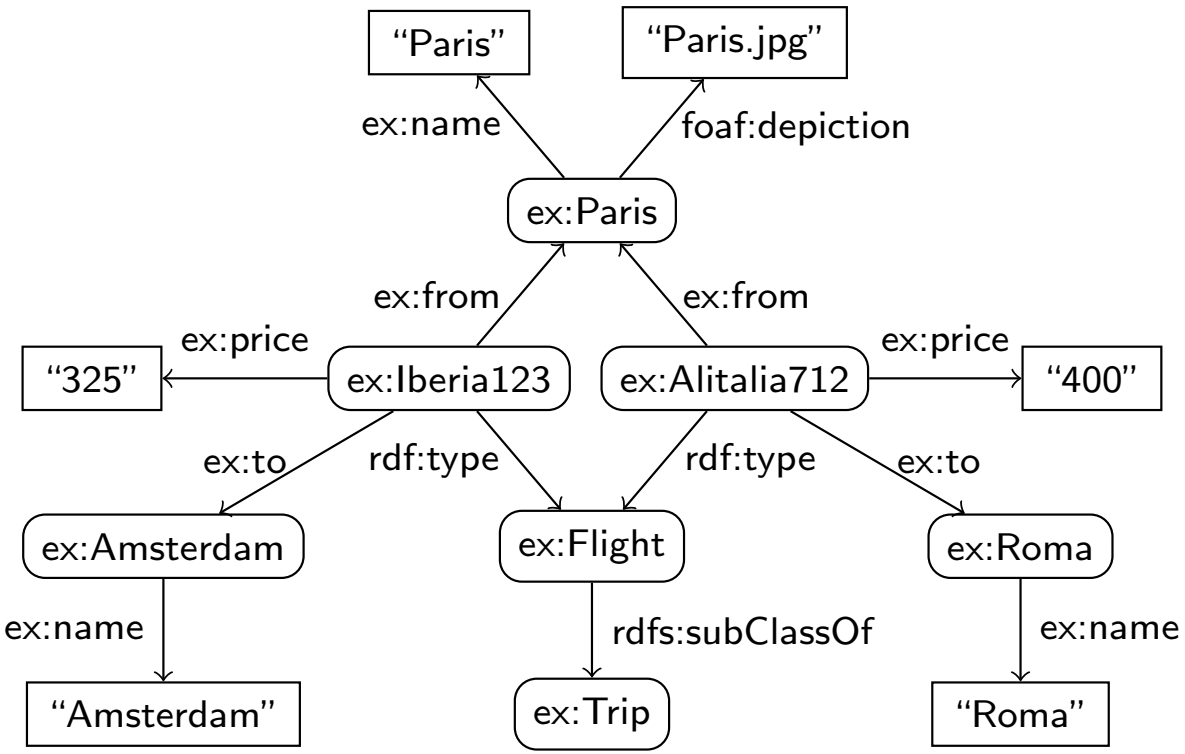
Query answer



The SPARQL Language with a CONSTRUCT clause

Query: Look for trips from Paris which cost a certain price

RDF Graph



SPARQL Query

```
CONSTRUCT
    {ex:Paris ex:reachable ?CityArrival .}
FROM RDF Graph
WHERE {
      ?Trip rdf:type ex:Trip .
      ?Trip ex:from ex:Paris .
      ?Trip ex:to ?CityArrival .
      ?Trip ex:price ?Price .
}
```

Limitation: SPARQL generates only RDF Graphs

Outline

Our SPARQL Extension

We propose an extension to SPARQL that:

- can be used to generate XML documents;
- preserves the compatibility with SPARQL syntax.

Extension of the CONSTRUCT clause

```
CONSTRUCT ⟨Example.xml⟩
FROM RDF Graph
WHERE {
    ?CityArrival foaf:depiction ?PhotoCityArrival .
    ...
}
```

Our SPARQL Extension

We propose an extension to SPARQL that:

- can be used to generate XML documents;
- preserves the compatibility with SPARQL syntax.

Extension of the CONSTRUCT clause

```
CONSTRUCT ⟨Example.xml⟩
FROM RDF Graph
WHERE {
    ?CityArrival foaf:depiction ?PhotoCityArrival .
    ...
}
```

The XML document is a template (Example.xml)

```
<table>
<tr>
<td><img src="?PhotoCityArrival" .../></td>
...
</tr>
</table>
```

Our SPARQL Extension

We propose an extension to SPARQL that:

- can be used to generate XML documents;
- preserves the compatibility with SPARQL syntax.

Extension of the CONSTRUCT clause

```
CONSTRUCT ⟨Example.xml⟩
FROM RDF Graph
WHERE {
    ?CityArrival foaf:depiction ?PhotoCityArrival .
    ...
}
```

The XML document is a template (Example.xml)

```
<table>
<tr>
<td><img src="?PhotoCityArrival" .../></td>
...
</tr>
</table>
```

After the query evaluation, *n* documents are computed according to the *n* solutions

Our SPARQL Extension

We propose an extension to SPARQL that:

- can be used to generate XML documents;
- preserves the compatibility with SPARQL syntax.

Query Answers

?PhotoCityArrival

http://ex.fr/Eiffel.jpg

http://ex.fr/BigBen.jpg

The XML document is a template (Example.xml)

```
<table>
<tr>
<td><img src="?PhotoCityArrival" .../></td>
...
</tr>
</table>
```

After the query evaluation, *n* documents are computed according to the *n* solutions

Our SPARQL Extension

We propose an extension to SPARQL that:

- can be used to generate XML documents;
- preserves the compatibility with SPARQL syntax.

Query Answers

?PhotoCityArrival

http://ex.fr/Eiffel.jpg

http://ex.fr/BigBen.jpg

Document instantiations

Document 1

```
<table>
<tr>
<td><img src="http://ex.fr/Eiffel.jpg" .../>
</td>
...
</tr>
</table>
```

Document 2

```
<table>
<tr>
<td><img src="http://ex.fr/BigBen.jpg".../>
</td>
...
</tr>
</table>
```

Document adaptation

```
CONSTRUCT <MobileTemplate.html>
FROM <RDFGraph>
FROM <CC/PPprofile>
WHERE {
      ?Image exif:resolution ?Res .
      ?Image rdf:type exif:IFD .
      ?Image mms:MmsMaxImageResolution ?MaxRes .
     FILTER ( ?Res <= ?MaxRes ) .
}
```

Document adaptation

```
CONSTRUCT <MobileTemplate.html>
FROM <RDFGraph>
FROM <CC/PPprofile>
WHERE {
      ?Image exif:resolution ?Res .
      ?Image rdf:type exif:IFD .
      ?Image mms:MmsMaxImageResolution ?MaxRes .
     FILTER ( ?Res <= ?MaxRes ) .
}
```

Document adaptation

```
CONSTRUCT <MobileTemplate.html>
FROM <RDFGraph>
FROM <CC/PPprofile>
WHERE {
      ?Image exif:resolution ?Res .
      ?Image rdf:type exif:IFD .
      ?Image mms:MmsMaxImageResolution ?MaxRes .
    FILTER ( ?Res <= ?MaxRes ) .
}
```

Limitations

- The SPARQL query and the template are edited independently.
  ⇒ The query and its variables must fit the template.
- Several queries may be specified for a given template.
  ⇒ Each of them may be applied to particular fragments of the template.
- One may want to compose several templates (Modular templates).
  ⇒ Reuse existing templates.

Outline

Modular template

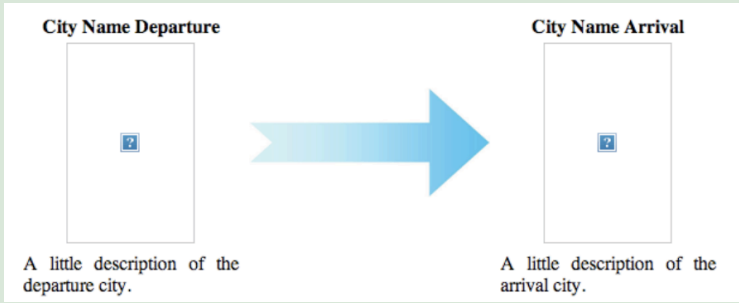
Definition (Modular template)

A modular template is a document where some of the data is given explicitly while other parts are described modularly or recursively, and externally constructed by means of embedded calls to query evaluators.

Example

imported

template



Syntax of a modular template

Syntax of a modular template

- `<sparqlmm:div>`:
  this tag encapsulates a query environment containing the specific tags such as `<sparqlmm:construct>` and `<sparqlmm:query>`. This tag may also embed other XML tags and/or even other query environments.
- `<sparqlmm:construct>`:
  this tag is composed of an XML fragment to be constructed from the answers of the query by substituting its variables.
- `<sparqlmm:query>`:
  we include in this tag a SPARQL query.

Metadata generation

```
<smil>
 <sparqlmm:div>
  <head>
   <metadata id="meta-rdf">
    <sparqlmm:query>
     <!-- construct <metadata.rdf>
          from <RDFGraph>
          where {
           ?Media rdf:type ex:Video .
           ?Media ex:author ?Author .
           ?Media ex:source ?Source .
           ?Media dc:title ?Title .
           ?Author foaf:name ?Name .}-->
    </sparqlmm:query>
   </metadata>
   ...
  </head>
  <body>
   <sparqlmm:construct>
    <video src="?Source" dur="30s"/>
   </sparqlmm:construct>
   ...
  </body>
</sparqlmm:div>
</smil>
```

Metadata generation

```
<smil>
 <sparqlmm:div>
  <head>
   <metadata id="meta-rdf">
    <sparqlmm:query>
     <!-- construct <metadata.rdf>
          from <RDFGraph>
          where {
           ?Media rdf:type ex:Video .
           ?Media ex:author ?Author .
           ?Media ex:source ?Source .
           ?Media dc:title ?Title .
           ?Author foaf:name ?Name .}-->
    </sparqlmm:query>
   </metadata>
   ...
  </head>
  <body>
   <sparqlmm:construct>
    <video src="?Source" dur="30s"/>
   </sparqlmm:construct>
   ...
  </body>
</sparqlmm:div>
</smil>
```

Metadata generation

```
<smil>
 <sparqlmm:div>
  <head>
   <metadata id="meta-rdf">
    <sparqlmm:query>
     <!-- construct <metadata.rdf>
          from <RDFGraph>
          where {
           ?Media rdf:type ex:Video .
           ?Media ex:author ?Author .
           ?Media ex:source ?Source .
           ?Media dc:title ?Title .
           ?Author foaf:name ?Name .}-->
    </sparqlmm:query>
   </metadata>
   ...
  </head>
  <body>
   <sparqlmm:construct>
    <video src="?Source" dur="30s"/>
   </sparqlmm:construct>
   ...
  </body>
</sparqlmm:div>
</smil>
```

Metadata generation

```
<smil>
 <sparqlmm:div>
  <head>
   <metadata id="meta-rdf">
    <sparqlmm:query>
     <!-- construct <metadata.rdf>
          from <RDFGraph>
          where {
           ?Media rdf:type ex:Video .
           ?Media ex:author ?Author .
           ?Media ex:source ?Source .
           ?Media dc:title ?Title .
           ?Author foaf:name ?Name .}-->
    </sparqlmm:query>
   </metadata>
   ...
  </head>
  <body>
   <sparqlmm:construct>
    <video src="?Source" dur="30s"/>
   </sparqlmm:construct>
   ...
  </body>
</sparqlmm:div>
</smil>
```

Metadata generation

```
<smil>
 <sparqlmm:div>
  <head>
   <metadata id="meta-rdf">
    <sparqlmm:query>
     <!-- construct <metadata.rdf>
          from <RDFGraph>
          where {
           ?Media rdf:type ex:Video .
           ?Media ex:author ?Author .
           ?Media ex:source ?Source .
           ?Media dc:title ?Title .
           ?Author foaf:name ?Name .}-->
    </sparqlmm:query>
   </metadata>
   ...
  </head>
  <body>
   <sparqlmm:construct>
    <video src="?Source" dur="30s"/>
   </sparqlmm:construct>
   ...
  </body>
</sparqlmm:div>
</smil>
```

metadata.rdf

```
<rdf:Description rdf:about="?Media">
 <dc:creator>
  <rdf:Description>
   <foaf:name>
    <sparqlmm:var name="?Name"/>
   </foaf:name>
  </rdf:Description>
 </dc:creator>
 <dc:title>
  <sparqlmm:var name="?Title"/>
 </dc:title>
 <ex:source>
  <sparqlmm:var name="?Source"/>
 </ex:source>
</rdf:Description>
```

Metadata generation

```
<smil>
 <sparqlmm:div>
  <head>
   <metadata id="meta-rdf">
    <sparqlmm:query>
     <!-- construct <metadata.rdf>
          from <RDFGraph>
          where {
           ?Media rdf:type ex:Video .
           ?Media ex:author ?Author .
           ?Media ex:source ?Source .
           ?Media dc:title ?Title .
           ?Author foaf:name ?Name .}-->
    </sparqlmm:query>
   </metadata>
   ...
  </head>
  <body>
   <sparqlmm:construct>
    <video src="?Source" dur="30s"/>
   </sparqlmm:construct>
   ...
  </body>
</sparqlmm:div>
</smil>
```

metadata.rdf

```
<rdf:Description rdf:about="?Media">
 <dc:creator>
  <rdf:Description>
   <foaf:name>
    <sparqlmm:var name="?Name"/>
   </foaf:name>
  </rdf:Description>
 </dc:creator>
 <dc:title>
  <sparqlmm:var name="?Title"/>
 </dc:title>
 <ex:source>
  <sparqlmm:var name="?Source"/>
 </ex:source>
</rdf:Description>
```

- **Template reusability**
- **Template readability**
- **Template modularity**

Metadata generation

```
<smil>
 <sparqlmm:div>
  <head>
   <metadata id="meta-rdf">
    <sparqlmm:query>
     <!-- construct <metadata.rdf>
          from <RDFGraph>
          where {
           ?Media rdf:type ex:Video .
           ?Media ex:author ?Author .
           ?Media ex:source ?Source .
           ?Media dc:title ?Title .
           ?Author foaf:name ?Name .}-->
    </sparqlmm:query>
   </metadata>
   ...
  </head>
  <body>
   <sparqlmm:construct>
    <video src="Seb.avi" dur="30s"/>
   </sparqlmm:construct>
   ...
  </body>
</sparqlmm:div>
</smil>
```

metadata.rdf

```
<rdf:Description rdf:about="vid1">
 <dc:creator>
  <rdf:Description>
   <foaf:name>
    Seb
   </foaf:name>
  </rdf:Description>
 </dc:creator>
 <dc:title>
  My video
 </dc:title>
 <ex:source>
  Seb.avi
 </ex:source>
</rdf:Description>
```

Control template importation

- Fragment selection.
  - `<sparqlmm:query xpath="..."> ...`
- Transforming template importation.
  - `<sparqlmm:query xslt="..."> ...`
- Condition on template importation.
  - `<sparqlmm:query condition="..."> ...`
- Template importation for some query answers.
  - `<sparqlmm:query offset="..." limit="..."> ...`

Outline

Conclusion

Bridge the gap between XML and RDF applications

- **A SPARQL extension** which supports the generation of XML documents based on templates
  - provides a formal and modular means to query RDF graphs
  - returns the results in XML
  - avoids developing ad hoc transformations from RDF to XML
- **A modular approach** that allows templates:
  - reusability
  - readability
- **Motivated examples** based on:
  - document adaptation
  - metadata generation
- **Implementation** on `http://sparqlmm.gforge.inria.fr` and **other concrete examples**

Future Work

- Controlling the number of generated documents by permitting the user to interact with the system.
- Mix our SPARQL templates with other template languages (e.g., XTiger).
- Implement a web service.

Questions ?

Thank you for your attention !

sebastien.laborie@inrialpes.fr
http://exmo.inrialpes.fr