HAL
open science

Towards extending and using SPARQL for modular document generation

Faisal Alkhateeb, Sébastien Laborie

To cite this version:

Faisal Alkhateeb, Sébastien Laborie. Towards extending and using SPARQL for modular document generation. Proc. 8th ACM symposium on document engineering (DocEng), Sep 2008, São Paolo, Brazil. pp.164-172, 10.1145/1410140.1410174 . hal-00793532

HAL Id: hal-00793532

https://inria.hal.science/hal-00793532v1

Submitted on 22 Feb 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Towards Extending and Using SPARQL for Modular Document Generation

Faisal Alkhateeb and Sébastien Laborie
INRIA Grenoble Rhône-Alpes and LIG
655 Avenue de l'Europe, 38334 St Ismier Cedex, France
{Faisal.Alkhateeb,Sebastien.Laborie}@inrialpes.fr

ABSTRACT

RDF is one of the most used languages for resource description and SPARQL has become its standard query language. Nonetheless, SPARQL remains limited to generate automatically documents from RDF repositories, as it can be used to construct only RDF documents. We propose in this paper an extension to SPARQL that allows to generate any kind of XML documents from multiple RDF data and a given XML template. Thanks to this extension, an XML template can itself contain SPARQL queries that can import template instances. Such an approach allows to reuse templates, divide related information into various templates and avoid templates containing mixed languages. Moreover, reasoning capabilities can be exploited using RDF Schema or simply RDFS.

Categories and Subject Descriptors

H.2.3 [**Languages**]: Query Languages; I.7.2 [**Document and Text Processing**]: Document Preparation.

General Terms

Languages.

Keywords

Semantic Web, SPARQL, RDF, Template, XML Document Generation.

1. INTRODUCTION

Nowadays, a fair amount of research has been conducted on documents generation. With the emergence of the Semantic Web [6], new applications based on expressive repositories should be developed. We propose to exploit current Semantic Web technologies in order to generate XML documents by using existing standard languages such as RDF and SPARQL.

RDF (Resource Description Framework) [19] is a knowledge representation language dedicated to the annotation of resources within the Semantic Web. Currently, many documents are annotated via RDF due to its simple data model and its formal semantics. For example, it is embedded in (X)HTML web pages using the RDFa language [2], in SMIL documents [12] using RDF/XML [5], etc.

SPARQL [21] is a W3C recommendation language developed in order to query RDF knowledge bases, e.g., retrieving nodes from RDF graphs. In addition, it can be used to construct RDF graphs from the instantiation of the query result to the graph pattern (i.e., an RDF graph with variables) specified in the CONSTRUCT clause of the query.

Hence, SPARQL could be used to generate documents. Nonetheless, it is limited to construct only RDF graphs, as it allows to specify exclusively RDF-based graph patterns in its CONSTRUCT clause.

To bridge the gap between RDF and XML technologies, transformation languages may be used. In this paper, to avoid the use of multiple languages, we extend SPARQL to allow constructing XML documents by specifying an XML template, i.e., an XML document with variables, in the query. Analogously to RDF graph patterns, XML templates are instantiated from the query result to construct the documents. Furthermore, our SPARQL extension applying on RDFS [9] expressions has the benefits to permit reasoning capabilities.

In order to apply several queries to a given template, we define an XML language which embed such queries inside a template. This language is useful for variable correspondences as queries are specified by authors while editing the template.

In this context, templates can be constructed with queries of the proposed SPARQL extension permitting modularity. That is, a query constructing one template can be used into another template and so on. Besides generating all templates instantiations, the modularity of our proposal has the following advantages:

- The template is more readable as we do not embed in one single template all information. Moreover, imported templates are constructed only if there are some answers to queries.
- Imported templates may be reused and are not duplicated into different templates.
- We do not need to merge the contents of all templates, which may be written in different languages, in a global one. Doing that necessitates developing a validator for the mixed template, if one wants to validate it.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DocEng'08, September 16–19, 2008, San Paulo, Brazil.
Copyright 2008 ACM 978-1-60558-081-4/08/09 ...$5.00.

The remainder of the paper is structured as follows. Section 2 introduces RDF and the SPARQL query language as will as our proposed extension. Section 3 describes how to embed and process queries into XML templates. Section 4 presents two direct applications of our work, namely document adaptation and metadata generation. Section 5 provides several enhancements to our proposal that can be used for controlling imported template structures. After a review of related work (Section 6), we conclude in Section 7.

2. A SPARQL EXTENSION FOR DOCUMENTS GENERATION

We first give in this section an introduction to the RDF and SPARQL languages (Section 2.1). Then, we propose a novel extension of SPARQL that allows generating XML documents, i.e., using XML templates in the CONSTRUCT clause of SPARQL queries (Section 2.2).

2.1 From RDF to SPARQL

RDF is a language for describing resources. In its abstract syntax, an RDF document is a set of triples of the form `<subject, predicate, object>`.

EXAMPLE 1. *The assertion of the following RDF triples:*

```
{ <ex:Iberia123, ex:from, ex:Paris>,
  <ex:Iberia123, ex:to, ex:Amsterdam>,
  <ex:Iberia123, ex:price, "325">,
  <ex:Iberia123, rdf:type, ex:Flight>
}
```

means that there exist a flight from Paris to Amsterdam with a price of 325 euros.

An RDF document can be represented by a directed labeled graph, as shown in Figure 1, where the set of nodes is the set of terms appearing as a subject or object in a triple and the set of arcs is the set of triples (i.e., if `<s, p, o>` is a triple, then $\texttt{s} \xrightarrow{\texttt{p}} \texttt{o}$).

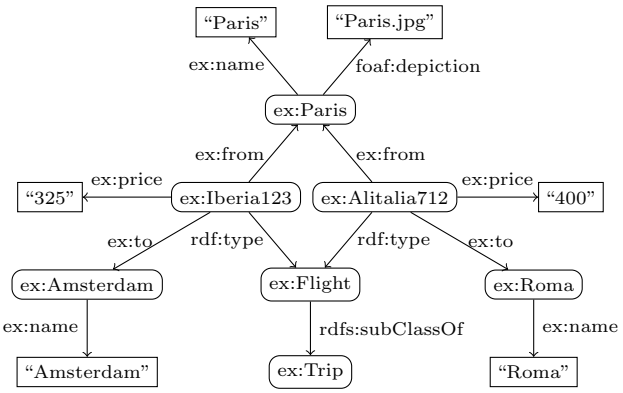


Figure 1: An RDF graph.

SPARQL is the query language developed by the W3C for querying RDF graphs. A simple SPARQL query is expressed using a form resembling the SQL SELECT query:

$$\texttt{SELECT}\ \vec{B}\ \texttt{FROM}\ u\ \texttt{WHERE}\ P$$

where u is the URL of an RDF graph G to be queried, P is a SPARQL graph pattern (i.e., a pattern constructed over RDF graphs with variables) and $\vec{B}$ is a tuple of variables appearing in P. Intuitively, an answer to a SPARQL query is an instantiation of the variables of $\vec{B}$ by the terms of the RDF graph G such that the substitution of the values to the variables of P yields to a subset of the graph G.[1]

EXAMPLE 2. *Consider the RDF graph of Figure 1 representing some possible flights between cities and some ticket prices. The existence of the following four triples* ⟨`?Flight, rdf:type, ex:Flight`⟩, ⟨`?Flight, ex:from, ex:Paris`⟩, ⟨`?Flight, ex:to, ?City`⟩ *and* ⟨`?Flight, ex:price, ?Price`⟩ *means that a* `?City` *is directly reachable with a* `?Flight` *from* `Paris` *and costs a certain* `?Price`.

The following SPARQL query modeling this information:

```
SELECT  *
FROM  <Figure1>
WHERE  {
         ?Flight rdf:type ex:Flight .
         ?Flight ex:from ex:Paris .
         ?Flight ex:to ?CityArrival .
         ?Flight ex:price ?Price .
       }
```

could be used, when evaluated against the RDF graph of Figure 1, to return the following answers:

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

In RDF there exists a set of reserved words (called RDF Schema or simply RDFS [9]), designed to describe the relationships between resources and properties, e.g., `classA subClassOf classB`. It adds additional constraints to the resources associated to the RDFS terms, and thus permitting more consequences (reasoning).

EXAMPLE 3. *Using the RDF graph presented in Figure 1, we can deduce the following triple* `<ex:Iberia123 rdf:type ex:Trip>` *from the triples* `<ex:Iberia123 rdf:type ex:Flight>` *and* `<ex:Flight rdfs:subClassOf ex:Trip>`. *Hence, the following SPARQL query :*

```
SELECT  *
FROM  <Figure1>
WHERE  {
         ?Trip rdf:type ex:Trip .
         ?Trip ex:from ex:Paris .
         ?Trip ex:to ?CityArrival .
         ?Trip ex:price ?Price .
       }
```

returns the same set of answers described in Example 1 because a flight is a subclass of trip.

SPARQL provides several result forms other than SELECT that can be used for formating the query results. For example, a CONSTRUCT query can be used for building an RDF graph from the set of answers to the query. More precisely, an RDF graph pattern (i.e., an RDF involving variables) is specified in the CONSTRUCT clause that will be constructed. For each answer to the query, the variable

[1] When using RDFS semantics [9], this intuitive definition is irrelevant and one could apply RDFS reasoning rules to calculate answers over RDFS documents.

values are substituted in the RDF graph pattern and the merge of the resulting RDF graphs is computed.[2] This feature can be viewed as rules over RDF permitting to build new relations from the linked data.

EXAMPLE 4. *The following CONSTRUCT query:*

```
CONSTRUCT {ex:Paris ex:reachable ?CityArrival .}
FROM   <Figure1>
WHERE  {
         ?Flight rdf:type ex:Flight .
         ?Flight ex:from ex:Paris .
         ?Flight ex:to ?CityArrival .
       }
```

constructs the RDF graph (containing the reachable relation) by substituting for each located answer the values of the variable `?CityArrival` *to have the following graph (as done for SPARQL, we encode the resulting graph in the Turtle language[3]):*

```
@prefix ex: <http://ex.org/> .

ex:Paris ex:reachable ex:Amsterdam .
ex:Paris ex:reachable ex:Roma .
```

Hence, SPARQL could be used to generate documents. However, the SPARQL CONSTRUCT clause is exclusively limited to use an RDF graph pattern, while one may want to generate XML documents. The next section provides an extension which overcome this limitation.

2.2 Using XML templates in SPARQL

This section proposes an extension to SPARQL that:

(1) can be used to generate XML documents;

(2) preserves the compatibility with SPARQL syntax.

In order to satisfy these requirements, we extend the syntax of SPARQL to allow using XML templates in the CONSTRUCT clause as described in the following way:

```
CONSTRUCT <cityTable.html>
FROM   <Figure1>
WHERE  {
         ...
       }
```

such that `cityTable.html` is an example of a URL referring to an XML template that we want to construct. An XML template is simply an XML document with some variables. These variables represent fields that will be instantiated from the query result. Figure 2 shows an example of such a template where variables are elements starting with ?. Note that template variables may also start with $ as in SPARQL.

In this template, variables are located either in attribute values (e.g., `src="?PhotoCityDeparture"`) or in specific tags (e.g., `<sparqlmm:var name="?CityNameDeparture">`).

After a query evaluation, we instantiate these variables according to the set of answers. More precisely, if a variable occurs inside the specific tag `sparqlmm:var`, the whole tag, including its content, is replaced by the value of that variable; the variable is replaced by its value in the attribute-value, otherwise. We repeat the process to obtain an XML document for each answer of the query.

[2]A definition of RDF merge operation can be found at `http://www.w3.org/TR/2001/WD-rdf-mt-20010925/#merging`.

[3]`http://www.dajobe.org/2004/01/turtle/`.

```
<table>
 <tr>
  <th><sparqlmm:var name="?CityNameDeparture">
      City Name Departure</sparqlmm:var></th>
  <th>
  </th>
  <th><sparqlmm:var name="?CityNameArrival">
      City Name Arrival</sparqlmm:var></th>
 </tr>
 <tr>
  <td class="centre"><img src="?PhotoCityDeparture"
      alt="Photo of City Departure" width="107" height =
      "167"/>
  </td>
  <td class="centre"><img src="arrow.jpg" alt="arrow"
      width="200" height="90" />
  </td>
  <td class="centre"><img src="?PhotoCityArrival"
      alt="Photo of City Arrival" width="107" height =
      "167"/>
  </td>
 </tr>
 <tr>
  <td><sparqlmm:var name="?CityDepartureDescription">
          little description of city departure
      </sparqlmm:var>
  </td>
  <td>
  </td>
  <td><sparqlmm:var name="?CityArrivalDescription">
         little description of city arrival
      </sparqlmm:var>
  </td>
 </tr>
</table>
```

Figure 2: An XML template (`cityTable.html`)

It should be noted that there are several output formats for the SPARQL query result such as N-Triples[4], Turtle, RDF/XML [5], etc. To transform from one result format to another, we must therefore use transformation tools, for example XSLT [13]. This is necessary, in particular, when there are different implementations of SPARQL supporting different output formats. Among the roles of our extension of SPARQL with XML templates is the natural ability of producing RDF/XML outputs by specifying an RDF/XML template.

Until now, the SPARQL query and the template are edited independently. Sometimes, it will be useful for an author to edit the query while editing the template. This is necessary because the query and its variables must fit the template. Moreover, several queries may be written such that each of them may be applied to a particular fragment of the template. In the next section, we propose to write and process queries inside templates.

3. COMPOSING MODULAR TEMPLATES

We call a document where some of the data is given explicitly while other parts are externally constructed by means of embedded calls to query evaluators, a modular template. In this section, we concentrate in modular XML templates: Section 3.1 introduces the syntax of a modular XML template containing SPARQL queries and Section 3.2 presents how to process such templates.

[4]`http://www.w3.org/2001/sw/RDFCore/ntriples/`

3.1 Syntax of modular templates

To allow SPARQL queries in templates we define an XML language composed of several XML tags[5]:

- `<sparqlmm:div>`: this tag encapsulates a query environment containing the specific tags `<sparqlmm:construct>` and `<sparqlmm:query>` that will be defined below. 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. One possible approach to do that is to assign the query to an attribute as done in SweetWiki [11]. However, the SPARQL syntax is not preserved (for instance, an HTML syntax is used for defining URI and URL references) which make their approach inapplicable to any kind of XML templates. Our approach considers the query as a value of the tag surrounded with a `<!-comment tag->` as done in Javascript. Doing so will ensure that browsers can at least display web-based templates correctly, e.g., (X)HTML templates. In contrast to the script tag of Javascript, the query tag may contain default values (see `<object data="cityTable.html"...>` of Figure 3).

Thanks to our proposed extension of SPARQL in Section 2.2, such templates may contain queries that can be used to construct external documents using external templates. This permits to reuse and compose existing templates in a global one (what is called modular document generation).

Example 5. *Figure 3 presents a modular XML template of a web page containing one query environment where:*

- *Part ⓐ is the query tag. This query finds information about cities and transportation means linking them from the graph of Figure 1. Moreover, it retrieves information about images and descriptions of cities from two web pages[6] (see Figure 4) encoded in XHTML+ RDFa [3]. Thereafter, it constructs, inter alia, the given template of Figure 2 (i.e., `cityTable.html`) from this information.*
- *Part ⓑ is the construct tag. This part contains some query variables, such as `?Price`, which are replaced if an answer to the query is found.*

As this template is based on XHTML, one may want to visualize it into a web browser before the instantiation of variables. Figure 5 presents the displayed page of the template of Figure 3.

Default values in the `sparqlmm:var` tags are used in order to display this page correctly, e.g., for the trip details schedule. These default values will be replaced when processing this templates.

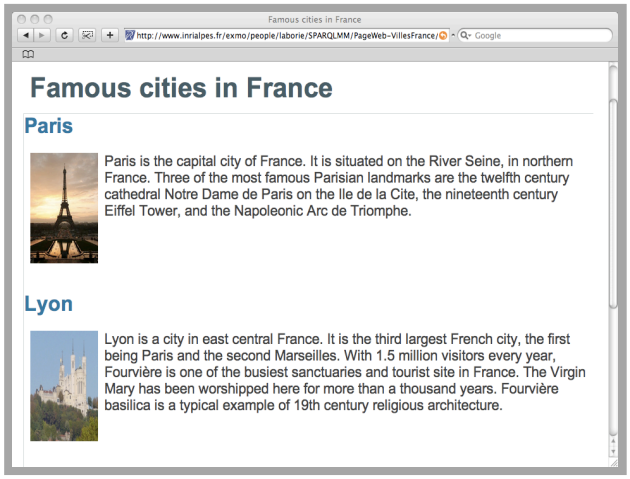
Next section describes how to generate final XML documents from the results of the queries and the involving modular XML templates.

[5]The XML Schema is accessible at `http://www.inrialpes.fr/exmo/people/laborie/SPARQLMM/sparqlmm.xsd`

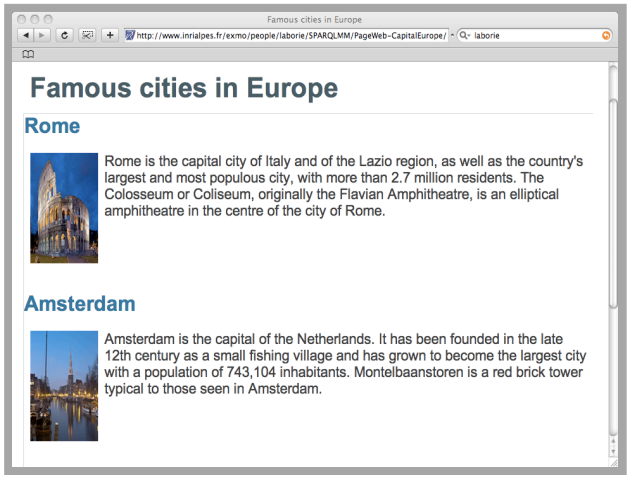
[6]Web pages are accessible at `http://www.inrialpes.fr/exmo/people/laborie/SPARQLMM/`

```
<body>
 <div>
  <h1>Trip Advertisement</h1>
  <sparqlmm:div>
   <div>
   <sparqlmm:query type="text/sparqlmm">
    <object data="cityTable.html" width="600" height=
     "255"/>
    <!--prefix ex: <http://ex.org/>
    prefix foaf: <http://xmlns.com/foaf/0.1/>
    prefix rdf: <http://.../22-rdf-syntax-ns#>
    construct <cityTable.html>
    from <Figure1>
    from <Figure4a>
    from <Figure4b>
    where {
     ?CityDeparture ex:name ?CityNameDeparture .
     ?CityDeparture ex:description
                 ?CityDepartureDescription .
ⓐ    ?CityDeparture foaf:depiction ?PhotoCityDeparture .
     ?CityArrival ex:name ?CityNameArrival .
     ?CityArrival ex:description
                 ?CityArrivalDescription .
     ?CityArrival foaf:depiction ?PhotoCityArrival .
     ?Flight rdf:type ex:Flight .
     ?Flight ex:from ?CityDeparture .
     ?Flight ex:to ?CityArrival .
     ?Flight ex:price ?Price .
     ?Flight ex:departureAirportName
                 ?DepartureAirportName .
     ?Flight ex:arrivalAirportName ?ArrivalAirportName .
     ?Flight ex:departureTime ?DepartureTime .
     ?Flight ex:arrivalTime ?ArrivalTime .}-->
   </sparqlmm:query>
   </div>
   <div>
   <sparqlmm:construct>
    <h2>Trip details</h2>
    <ul>
     <li>The price is <sparqlmm:var name="?Price">?
         </sparqlmm:var>in euros.
     </li>
     <li>Departure from <sparqlmm:var
       name="?DepartureAirportName">?</sparqlmm:var>
       airport at <sparqlmm:var name="?DepartureTime">
ⓑ      hh:mm</sparqlmm:var>.
     </li>
     <li>Arrival to <sparqlmm:var name=
       "?ArrivalAirportName">?</sparqlmm:var>
       airport at <sparqlmm:var name="?ArrivalTime">
       hh:mm</sparqlmm:var>.
     </li>
    </ul>
   </sparqlmm:construct>
   </div>
  </sparqlmm:div>
 </div>
</body>
```

Figure 3: A modular XML template.



(a)



(b)

Figure 4: XHTML+RDFa web pages describing famous cities in France and in Europe.

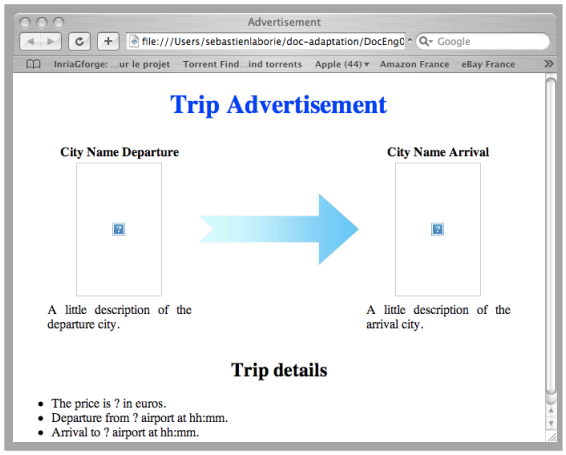


Figure 5: Displayed page of Figure 3.

3.2 Processing modular templates

Intuitively, processing a modular template means retrieving all query results and filling the missing values in the template, i.e., generate final documents. Algorithm 1 describes this process.

Algorithm 1: Process Modular Template

Input: A modular template.
Output: A set of documents.
if *the template contains a query environment* **then**
  we evaluate its query to find the results;
  for *each answer* **do**
    if *the query imports an external template* **then**
      we import the document constructed from the instantiation of result to its template;
    if *some construct tags exist* **then**
      we substitute the values to the variables;
we repeat the same steps for each query environment and return the Cartesian product of document instantiations;

First, this algorithm computes the answers for each query. Then, it replaces each query environment by the computed result. When there are several query environments, we compose all instantiations of the query environments by taking the Cartesian Product of all queries results. For example, if we have two query environments, q_1 and q_2, in a template such that q_1 has 2 answers and q_2 has 3 answers, the algorithm produces 6 final documents.

At the end of the algorithm, the `<sparqlmm:query>` tags are replaced by the instantiated imported templates, if they exists. The `<sparqlmm:construct>` tags are removed with keeping their contents, if at least one answer to their associated query exists. Finally, the `<sparqlmm:div>` tags are removed with keeping their contents.

Example 6. *Consider the modular template of Figure 3 which contains one query environment. According to Algorithm 1, we first compute the results of the query (Part ⓐ). The following table provides some of the results. For visibility reason, we select a subset of the query variables.*

#	?CityDeparture	?CityArrival	?Price	...
1	ex:Paris	ex:Amsterdam	"325"	...
2	ex:Paris	ex:Roma	"400"	...

For each answer of this table, we substitute the values to the variables of the cityTable template of Figure 2 (suppose we have a document `cityTable-sol1` *for the first answer), and we substitute the variable-values to the fragment in the construct tag of Part ⓑ (suppose the result of this instantiation is the fragment* `construct-sol1`*). The query environment is then replaced by the resulting documents,* `cityTable-sol1` *and* `construct-sol1`*. Figure 6 presents one document instantiation for the first answer.*

In the next section, we provide some direct applications of our proposal.

4. EXAMPLES OF APPLICATIONS

Section 4.1 is an application of the extension to SPARQL proposed in Section 2.2 and Section 4.2 shows the usefulness of composing modular templates.

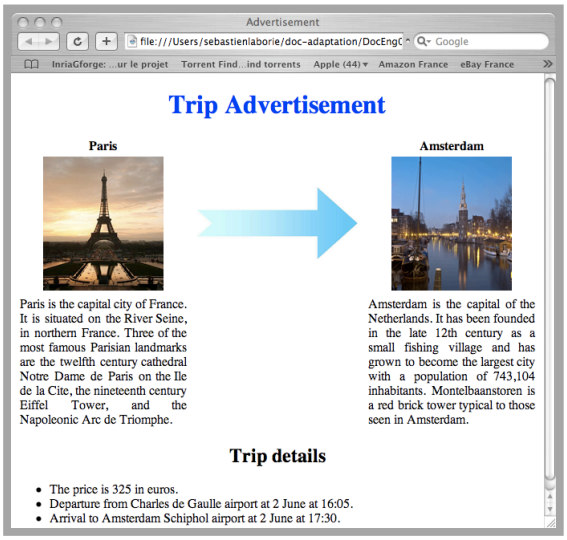


Figure 6: An executed document for the first answer of the template of Figure 3.

4.1 Document adaptation

One application of our proposed SPARQL extension, presented in Section 2.2, is to generate adapted documents according to target device constraints or user preferences.

Indeed, we can construct documents that may be constrained by a given profile (for example, one can use UAProf [23], based on CC/PP [18], to encode in RDF such a profile). In this context, adaptation could be achieved using the SPARQL FILTER clause to restrict the answers to the set that satisfies the given profile.

EXAMPLE 7. *Assume we want to construct several documents, according to a mobile phone template, which must satisfy a given profile. This profile contains information about display limitations of a particular mobile phone such as its maximum resolution.*

The following query:

```
PREFIX exif: <http://www.w3.org/2003/12/exif/ns>
PREFIX mms: <http://www.openmobilealliance.org/tech/
              profiles/MMS/ccppschema-20050301-MMS1.2#>
CONSTRUCT <MobileTemplate.html>
FROM <RDFGraph>
FROM <profile>
WHERE {
         ?Image exif:resolution ?Res .
         ?Image rdf:type exif:IFD .
         ?Image mms:MmsMaxImageResolution ?MaxRes .
         FILTER ( ?Res <= ?MaxRes ) .
}
```

which constructs the template `MobileTemplate.html`*, selects only images that have some resolutions less than a maximum profile resolution.*

4.2 Generating metadata

Most of the time, metadata are semantically linked to the content of the document but are syntactically described in separate parts of it. Moreover, their metadata schemas in most cases are recurrent.

For example, in the case of SMIL documents [12], while the content of the `head` and `body` elements are linked, they are described separately inside the document structure and the metadata schema of the `head` element is almost recurrent, e.g., author information, title, abstract, keywords, publisher, copyright.

With our approach, the metadata structure can be described once in a template. This template may be used multiple times in other templates in order to generate the whole documents.

EXAMPLE 8. *Given a part of an RDF metadata structure template (see Figure 7) that describes information about a media such as its author, its title and its source location. As this template is created separately, it can be validated by an RDF/XML validator*[7] *assuming that the namespaces are well defined.*

```
<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>
```

Figure 7: The RDF metadata template.

This template can be used to generate metadata for any kind of documents. For example, the SMIL document template of Figure 8 calls a query that uses this template. The related information of the query result is substituted in both the metadata template structure and the SMIL content. We recall that the substituted metadata template structure is imported in the place of the query.

5. CONTROLLING TEMPLATES GENERATION

This section includes possible structures extending our work that can be used for controlling the imported template instantiations. For that purpose, Section 5.1 describes how it is possible to import a fragment of a template, Section 5.2 considers its transformation in order to fit the language of the source template and Section 5.3 proposes to control its importation.

5.1 Fragment selection

Instead of importing a whole template, one may want to import a fragment of this template. In such a case, we can use XML queries, e.g., XPath [14], to select a desired fragment. The fragment selection can be applied before or after the evaluation of the template query.

EXAMPLE 9. *The XPath expression in the following query tag:*

```
<sparqlmm:query xpath="table/tr[2]">
```

could be used to select only the second row from the template of Figure 2 (`cityTable.html`*) to be constructed in the template query.*

[7] `http://www.w3.org/RDF/Validator/`

```
<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>
```

Figure 8: A SMIL template that uses a query which imports an RDF metadata template.

```
<Flight>
 <CityDeparture>
  <name>
   <sparqlmm:var name="?CityNameDeparture"/>
  </name>
  <photo>
   <sparqlmm:var name="?PhotoCityDeparture"/>
  </photo>
  <description>
   <sparqlmm:var name="?CityDepartureDescription"/>
  </description>
 </CityDeparture>
 <CityArrival>
  <name>
   <sparqlmm:var name="?CityNameArrival"/>
  </name>
  <photo>
   <sparqlmm:var name="?PhotoCityArrival"/>
  </photo>
  <description>
   <sparqlmm:var name="?CityArrivalDescription"/>
  </description>
 </CityArrival>
</Flight>
```

Figure 9: An XML template.

5.2 Transforming template importation

When importing a template, its language may correspond to the source template, otherwise the final constructed document may not be valid.

EXAMPLE 10. *Suppose we want to import the template of Figure 9 into the web page template of Figure 3, instead of* `cityTable.html`*. Without any transformation, the final document will not be HTML valid because the imported template of Figure 9 does not use HTML tags.*

In order to import a template in a specific format, we allow the use of XSLT stylesheets in query tags. This is done by using an `xslt` attribute containing the location of an XSLT stylesheet. For example, the template of Figure 9 can be transformed into the template of Figure 2 using an XSLT stylesheet, hence producing, in the context of the template of Figure 3, an HTML valid document.

5.3 Extending the language constructs

In a template, everything specified in the target language must occur as in all document instances. However, instead of having one template instantiation for each answer, one may want to have all instantiations in one document. This could be achieved by a form of repetition structure over the query answers. Moreover, conditions on solutions selection may be specified.

In this context, these kinds of structures could be applied to both the query tag, i.e., on the importation of a template, and the construct tag.

For the query tag, we can use the attributes `offset` and `limit` to import a set of constructed templates in the document. The former determines from which answer we start the repetition and the latter specifies the maximum number of answers. Although these features are supported by SPARQL, their use will restrict the answers to both the query and the construct tags. Indeed, the defined attributes controls only the answers that will be applied to the template of the query tag. The attribute `condition` could also be used to restrict the importation of a template, i.e., the template is imported only if the condition is satisfied.

For the construct tag, we can add the tag `<sparqlmm:repeat offset="..." limit="...">` to repeat a given structure for a set of answers and the tag `<sparqlmm:if condition="...">` to construct parts when the condition is satisfied.

6. RELATED WORK

Our goal, in this paper, is to bridge the gap between XML and Semantic Web technologies in the context of documents generation. In this section, we present some works which are related to this issue. In particular, Section 6.1 describes some systems using SPARQL queries in order to present information. Thereafter, Section 6.2 presents some pure XML approaches which generate documents.

6.1 Semantic Web technologies

With the emergence of the Semantic Web, new systems, based on SPARQL queries, are currently developed in order to display information:

The most related work is that of [11], which describes SweetWiki, a wiki engine that uses technologies from the Semantic Web. This system embeds knowledge, content, structures and links, in wiki pages. SweetWiki uses an XHTML WYSIWYG editor that allows embedding SPARQL queries. However, this work is restricted to XHTML pages and does not permit importing templates since they do not construct XML templates. Therefore, all desired information must be embedded while creating a wiki page.

XSPARQL [4] is a merge of SPARQL queries into XQuery [8]. This language has the potential to bring XML and RDF closer together. Indeed, XSPARQL provides concise and intuitive solutions for mapping between XML and RDF in either direction. Nonetheless, the SPARQL language is altered and in order to generate documents one is supposed to be familiar with the XQuery syntax.

SparqPlug [15] aims to ease the creation of RDF data from HTML data sources. One of its features is also the ability to insert variables you wish to use to construct your RDF within an HTML fragment. This is achieved by converting heuristically the given fragment into a SPARQL SELECT query and then getting the missing spots in the fragment. Nevertheless, this approach is restricted to HTML with no embedded queries. Moreover, it is unclear of how to construct the query from the template variables. That is, do they construct a query for each variable or a query for all. In both cases, do the semantic relations between variables are preserved or not.

Fresnel [20] provides an RDF vocabulary encoding information about how to present Semantic Web content to users, i.e., what content to show and how to show it. The former may be selected by specifying SPARQL queries and the latter uses existing styling languages, such as CSS, in order to determine how resources and properties are rendered. However, no XML templates are used.

Cuypers [17] is a system that generates multimedia documents from an RDF-based semantic graph. It uses queries supported by Sesame [10] to retrieve RDF data from the semantic graph and select the suitable discourse structure based on these data. The document structure is then defined from the discourse structure. This means that both of them must be recomputed each time a query is given. Moreover, the system does not support processing templates with embedded queries and/or construct template queries.

6.2 XML technologies

A fair amount of research has been conducted on documents generation using XML technologies. In the following, we present some examples:

[1] proposes a model based on Active XML documents (called AXML), which are XML documents that may contain embedded calls to Web services and AXML services (i.e., web services capable of exchanging AXML documents).

The STAMP model (Synchronized Templates for Adaptable Multimedia Presentations) [7] addresses the dynamic generation of multimedia presentations from XML data obtained by means of SQL queries.

XTiger (Extensible Templates for Interactive Guided Edition of Resources) [16] is a language designed to describe semantically rich XML languages in terms of other XML languages, such as XHTML. It is equipped with elements that can be used for controlling document structures, like repeat, option, etc.

However, all these approaches are limited to XML, hence no reasoning can be used, as our work can be applied on RDFS [9] (cf., Section 2.1).

7. CONCLUSION

The availability of RDF(S) data is recognized as a key asset that enables aggregation, provisioning, retrieval, reasoning, reusability, adaptation and personalization of multimedia content. However, the standard query language of RDF, SPARQL, is currently limited to achieve all these tasks.

We have proposed an extension to SPARQL which allows generating XML documents. One of the uttermost usefulness of this kind of queries is their embedding in XML templates which permits document generation modularity. An implementation in Java of our proposal is available at `http://sparqlmm.gforge.inria.fr`. The prototype generates all XML documents based on the number of query answers.

A future work consists in controlling the number of generated documents by permitting, for example, the user to interact with the system. This way she/he can select desired answers of a query to be composed with other query answers.

Another issue concerns semantic preservation of imported templates, e.g., preserving the URIs. We will investigate the approach presented in [22] which mainly contributes to document migration while maintaining "provable correctness".

Finally, we study to what extent we can mix other approaches, such as XTiger and Fresnel, with our proposal. In particular, using more specific tags that could control the importation of a template.

8. REFERENCES

[1] Abiteboul, S., Benjelloun, O., and Milo, T. The Active XML project: an overview. *The VLDB Journal* (2008).

[2] Adida, B., and Birbeck, M. RDFa primer - embedding structured data in web pages. Working draft, W3C, 2008. `http://www.w3.org/TR/xhtml-rdfa-primer/`.

[3] Adida, B., Birbeck, M., McCarron, S., and Pemberton, S. RDFa in XHTML: Syntax and processing. Working draft, W3C, 2008. `http://www.w3.org/TR/rdfa-syntax/`.

[4] Akhtar, W., Kopecký, J., Krennwallner, T., and Polleres, A. XSPARQL: Traveling between the XML and RDF worlds – and avoiding the XSLT pilgrimage. In *Lecture Notes in Computer Science (LNCS 5021), Proceedings of the 5th European Semantic Web Conference (ESWC)* (June 2008), Springer, pp. 432–447.

[5] Beckett, D., and McBride, B. RDF/XML syntax specification (revised). Recommendation, W3C, 2004. `http://www.w3.org/TR/rdf-syntax-grammar/`.

[6] Berners-Lee, T., Hendler, J., and Lassila, O. The semantic web. *Scientific American 284*, 5 (May 2001), 34–43.

[7] Bilasco, I. M., Gensel, J., and Villanova-Oliver, M. STAMP: a model for generating adaptable multimedia presentations. *Multimedia Tools and Applications, Special Issue on Metadata and Adaptibility in Web-based Information Systems 25*, 3 (March 2005), 361–375.

[8] Boag, S., Chamberlin, D., Fernández, M. F., Florescu, D., Robie, J., and Siméon, J. XQuery 1.0: An XML Query Language. Recommendation, W3C, 2007. `http://www.w3.org/TR/xquery/`.

[9] Brickley, D., and Guha, R. RDF vocabulary description language 1.0: RDF schema. Recommendation, W3C, 2004. `http://www.w3.org/TR/rdf-schema/`.

[10] Broekstra, J., Kampman, A., and van Harmelen, F. Sesame: An architecture for storing and querying RDF data and schema information. In *Semantics for the WWW* (2001), D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster, Eds., MIT Press.

[11] Buffa, M., Gandon, F., Ereteo, G., Sander, P., and Faron, C. SweetWiki: A semantic wiki. *Special*

Issue of the Journal of Web Semantics on Semantic Web and Web 2.0 6, 1 (2008), 84–97.

[12] Bulterman, D., Grassel, G., Jansen, J., Koivisto, A., Layaïda, N., Michel, T., Mullender, S., and Zucker, D. Synchronized Multimedia Integration Language (SMIL 2.1). Recommendation, W3C, 2005. `http://www.w3.org/TR/SMIL/`.

[13] Clark, J. XSL transformations (XSLT) Version 1.0. Recommendation, W3C, 1999. `http://www.w3.org/TR/xslt`.

[14] Clark, J., and DeRose, S. XML Path Language (XPath) version 1.0. Recommendation, W3C, 1999. `http://www.w3.org/TR/xpath`.

[15] Coetzee, P., Heath, T., and Motta, E. SparqPlug: Generating linked data from legacy HTML, SPARQL and the DOM. In *Proceedings of the Linked Data on the Web (LDOW2008), WWW Workshop* (2008).

[16] Flores, F. C., Quint, V., and Vatton, I. Templates, microformats and structured editing. In *Proceedings of the 2006 ACM symposium on Document engineering* (2006), ACM, pp. 188–197.

[17] Geurts, J., Bocconi, S., van Ossenbruggen, J., and Hardman, L. Towards ontology-driven discourse: From semantic graphs to multimedia presentations. In *Proceedings of the 2nd International Semantic Web Conference* (2003), pp. 597–612.

[18] Klyne, G., Reynolds, F., Woodrow, C., Ohto, H., Hjelm, J., Butler, M. H., and Tran, L. Composite Capability/Preferences Profiles (CC/PP): Structure and Vocabularies 1.0. Recommendation, W3C, 2001. `http://www.w3.org/TR/CCPP-struct-vocab/`.

[19] Manola, F., and Miller, E. RDF primer. Recommendation, W3C, 2004. `http://www.w3.org/TR/rdf-primer/`.

[20] Pietriga, E., Bizer, C., Karger, D., and Lee, R. Fresnel - A Browser-Independent Presentation Vocabulary for RDF. In *Lecture Notes in Computer Science (LNCS 4273), Proceedings of the 5th International Semantic Web Conference (ISWC 2006)* (November 2006), Springer, pp. 158–171.

[21] Prud'hommeaux, E., and Seaborne, A. SPARQL query language for RDF. Recommendation, W3C, January 2008. `http://www.w3.org/TR/rdf-sparql-query/`.

[22] Triebsees, T., and Borghoff, U. M. Towards automatic document migration: semantic preservation of embedded queries. In *Proceedings of the 2007 ACM symposium on Document engineering* (2007), ACM, pp. 209–218.

[23] Wireless Application Protocol Forum. User Agent Profile specification. Tech. rep., Open Mobile Alliance, 2001. `http://www.openmobilealliance.org/tech/affiliates/wap/wap-248-uaprof-20011020-a.pdf`.