ITERATIVE XML SEARCH BASED ON DATA AND ASSOCIATED SEMANTICS

Alda Lopes Gançarski
Institut TELECOM, TELECOM et Management SudParis, CNRS SAMOVAR, 9 rue Charles Fourier, 91011 Évry, France

Pedro Rangel Henriques, Flávio Xavier Ferreira
Department of Informatics, University of Minho, Campus de Gualtar, 4710-057 Braga, Portugal

Keywords: XQuery, iterative XML search, metadata, ontologies, RDF, SPARQL.

Abstract: In a previous work in the context of information retrieval, XQuery was extended with an iterative paradigm. This extension helps the user getting the desired results from queries. In a related work, XQuery was also extended to allow the inclusion of SPARQL queries; this is useful when XML documents are associated with semantic RDF descriptions. However integrating SPARQL in XQuery queries makes the construction of queries more complex (although more powerful). To leverage this integration, we propose to apply the iterative paradigm to the 'SPARQL extension to XQuery'. In the paper this proposal is introduced and justified and a case study is presented.

1 INTRODUCTION[1]

XML information access is done using structured query languages such as XPath (Berglund et al., 2007) and XQuery (Boag et al., 2006), the standard proposed by the W3C. To help the user get the desired information from his queries, (Gançarski et al 2006) proposed an iterative search over XML documents using an extension to XPath. The iterative paradigm was, then, included in XQuery (Gançarski and Henriques, 2006).

To improve data processing, the document collections and Web resources are associated with semantic descriptions, i.e. metadata. In order to be able to exchange the semantics of information, one first needs to agree on how to explicitly model it. This is usually done using a sophisticated description in the form of an ontology. An ontology is a formal explicit specification of a shared conceptualization. Using an ontology, any kind of description can be made about a resource. Ontologies can be used to annotate data with labels indicating their meaning, thereby making their semantics explicit and machine-accessible. W3C has created the Resource Description Framework (RDF) (Manola and Miller, 2004), a language for representing information about resources in the World Wide Web. RDF Schema (RDFS) is an RDF extension, which provides the basic elements for ontologies descriptions. To find information in RDF descriptions, the SPARQL query language was defined by the W3C (Prud'hommeaux and Seaborn, 2007).

In (Gançarski and Henriques 2007), they exploit the use of XML documents together with the respective semantics to access information, arguing that both may be interesting to the user and can help him to find the desired information. For that, they integrate SPARQL queries into XQuery ones.

In this paper, we propose to apply the iterative paradigm to the SPARQL extension made to XQuery. In fact, searching by data and metadata is more sophisticated than simple data search. Thus, the user may take advantage from the iterative paradigm when building his queries, not only in the XQuery component, but also in the SPARQL one.

Next section introduces the XQuery iterative model. Section 3 introduces the search based on data and associated semantics. Then, Section 4 integrates the iterative paradigm with the semantic search. The formal definition of the proposed XQuery extension is made in Section 5. A case study is described in

[1] This research is done in the context of the RESPIRE project financed by the French ANR-ARA program.

Lopes Gançarski A., Rangel Henriques P. and Xavier Ferreira F. (2008).
ITERATIVE XML SEARCH BASED ON DATA AND ASSOCIATED SEMANTICS.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - DISI*, pages 479-484
DOI: 10.5220/0001706504790484

Section 6. The article finishes with a brief conclusion, indicating some future work. For the sake of simplicity, we will ignore IRIref definitions in our examples.

2 ITERATIVE SEARCH

The iterative paradigm of query construction is based on selection operations that consist in restricting intermediate results to the subset of elements that satisfies the user. For that, the *mf:select* function may be used in location path expressions (*mf* from *my function*). The *mf:select* function selects the subset of interesting elements based on some criteria. While in a filter the set of elements is selected by intention, with *mf:select* it is by extension, i.e. explicitly referring to each element. This can be interesting when the specification of the criteria is too complicated (the user may even not know how to do it) or when it is more efficient/rapid to directly refer the desired elements.

Suppose each node is identified by a unique identifier and consider it as a string of characters. The input to *mf:select* is a node and a list of node identifiers (denoted by "(...)"). The output is the input node if it is selected (i.e., if its identifier belongs to the list of identifiers), or an empty sequence of nodes. For example, suppose the user wants references made inside interesting articles of author Kevin. The user can, then, make the following query:

```
for $a in /articles/article[author = ``Kevin"]
                    [mf:select(., ("a4", "a8"))]
return $a//references
```

In this query, function *mf:select* selects articles identified by "a4" and "a8". Symbol "." refers to each context node, i.e., each resulting node of the precedent operation. Thus, *mf:select* takes each article being a context node and returns it if it corresponds to some of the selected items.

3 DATA AND SEMANTICS SEARCH

When an XML document is associated to semantic descriptions expressed in RDF, SPARQL queries may be integrated in XQuery queries. This can be done by adding a new clause *metadata* to the *for* clause of XQuery. Let us call the extended XQuery with the *metadata* clause *XQuery+SPARQL*. As an example, consider the following ontology that includes as concepts, among other things, elements of an XML document.

```
Book.xml              about     "Beings".
Book.xml#Chapter1     about     "Fishes".
Book.xml#Section11    about     "Ocean Fishes".
Book.xml#Section12    about     "River Fishes".
Book.xml#Chapter2     about     "Birds".
Book.xml#Chapter3     about     "Vegetables".
Book.xml#Chapter4     about     "Fruits".
Men       eat   "Birds".
Men       eat   "Fishes".
Men             eat   "Vegetables".
Men             eat   "Fruits".
```

If the user wants chapters about what men eat, he can specify the following XQuery+SPARQL query:

```
1  for $c in
2  doc("http://.../Book.xml") /book//chapter
3  metadata  $c  in
4    SELECT   ?c
5    WHERE    {   Men  eat ?o.
6               ?c  about ?o. }
7  return $c
```

In this query, the *for* clause associates to variable *$c* the set of chapters of the document (lines 1 and 2). The *metadata* clause (line 3) includes a SPARQL query (line 4 to 6) which selects all the book parts (stored in variable *?c*) that are about beings eaten by men (stored in variable *?o*). This yields the set *{Book.xml#Chapter1, Book.xml#Chapter2, Book.xml#Chapter3, Book.xml#Chapter4}*, stored in variable *?c*. This result is intersected with the set of elements of the XQuery external query, stored in variable *$c*, to get the desired parts of the book.

4 ITERATIVE DATA AND SEMANTICS SEARCH

The formulation of XQuery+SPARQL queries is more complex than simple XQuery queries. To simplify this task to the user, we propose to extend the iterative paradigm to the SPARQL component of XQuery+SPARQL. For that, a selection function is included in SPARQL, associated to the FILTER clause. Let us see an example query using a filter and, then, incorporate the selection function on it.

4.1 FILTER Clause

SPARQL filters restrict solutions to those for which the filter expression evaluates to true. Filters use functions to define conditions. Those functions may come from XPath, XQuery or may be SPARQL specific. For example, consider the same ontology as before in Section 3. Suppose the user wants sections about fishes. He may, then, specify the following query:

```
1  for $s in
2  doc("http://.../Book.xml") /book//section
3  metadata  $s  in
4    SELECT   ?s
5    WHERE    {   ?s  about ?o.
6                FILTER regex(?o, "Fishes") }
7  return $c
```

The *regex* function (line 6) is SPARQL specific and allows matching a string with a pattern. In the example query, the string is stored in variable *?o* and the pattern is the simple string *"Fishes"*. The result of the *Where* clause is the set *{ Book.xml#Chapter1, Book.xml#Section11, Book.xml#Section12}*. When making the intersection with the result of the external XQuery query (which gives sections from *Book.xml*), the final result becomes *{Book.xml#Section11, Book.xml#Section12}*.

4.2 *Select* Function

With the proposed iterative paradigm, when a variable is computed, the user may see its content and select the subset of interesting values. For that, we define the *msf:select* function (*msf* from *my SPARQL function*). Suppose again the ontology of the previous example in Section 3. If now the user wants animals eaten by men, he may specify a query in the following steps:

Step 1.

```
for $s in
doc("http://.../Book.xml") /book//chapter
metadata  $s  in
  SELECT   ?s
  WHERE  { Men  eat ?o.
```

At this point, with the iterative paradigm, the user may access the intermediate result stored in variable *?o*. This result is the set *{"Birds", "Fishes", "Vegetables", "Fruits"}*. The user may, then, select the animals, as specified in the next step.

Step 2.

```
for $s in
 doc("http://.../Book.xml") /book//chapter
 metadata  $s  in
   SELECT   ?s
   WHERE
     { Men  eat ?o.
       FILTER msf:select(?o, ("Birds", "Fishes"))
       ?s  about ?o.
     }
 return $s
```

With the *msf:select* operation, the content of variable *?o* became *{"Birds", "Fishes"}*. Thus, the final result of the query is *{Book.xml#Chapter1, Book.xml#Chapter2}*.

5 FORMAL DEFINITION OF SELECT FUNCTION

The *FILTER* clause may be associated to user defined functions, as specified in the following productions:

```
[26]  Filter   ::=  'FILTER' Constraint
[27]  Constraint  ::= FunctionCall
```

The *msf:select* function is defined using the SPARQL grammar productions corresponding to user defined functions:

```
[28]  FunctionCall  ::=  IRIref ArgList
[29]  ArgList  ::=  ( NIL | '(' Expression ( ',' 
                    Expression )* ')' )
```

Here, the name of the function is derived by *IRIref*. This symbol allows for complete IRI references or prefixed names. In our case, we use the prefix *msf* and the name *select*.

The arguments of user defined functions are represented by the symbol *Expression*. This is a general symbol representing from simple literal strings to complex Boolean expressions. In the *msf:select* function, the first occurrence of *Expression* derives an RDF term, corresponding to the content of some variable, and the second one, a bracketed sequence of RDF terms. This sequence corresponds to the sequence of selected terms from an intermediate result. What follows defines the *msf:select* function:

```
xsd:boolean msf:select(RDF term t,
                       (RDF term)* tSeq)
```

```
    {
    for (i=0; i<length[tSeq]; i++) {
          if (sameTerm(t, tSeq[i]))
              { return TRUE; exit; }
    }
    Return FALSE;
}
```

The *sameTerm* SPARQL pre-defined function returns true if both arguments are equal.

For each term bound to a variable passed as the first argument of *msf:select*, this function returns true if the term exists in the sequence of terms passed as the second argument; otherwise, it returns false. Considering the example query of Section 4.2, variable *?o* is bound to the set of terms *{"Birds", "Fishes", "Vegetables", "Fruits"}*. This variable is the first argument of the *msf:select* function. The second argument is the sequence *("Fishes", "Birds")* choose by the user. So, verifying if each term stored in variable ?o occurs in the sequence, the content of *?o* becomes *{"Birds", "Fishes"}*.

6 CASE STUDY

We intend to experiment our approach with the resources of the Portuguese Emigration Museum (Museu da Emigração e das Comunidades - **MEC**). MEC is a web-museum that wants to make easily accessible to the general public the rich cultural heritage characterizing the Portuguese emigration phenomenon, and the impress left by the Portuguese people around the world.

MEC assets (resources) are vast and multifaceted because the emigration documents and objects come from the most diversified sources, ranging from official government records to old newspapers and photo albums, with the type of documents also heterogeneous (from official travel reports to local stories). So it becomes necessary to organize and categorize all this resources. As such, the information sources and the resources where classified using an ontology, presented partially in Figure 1 (at the end of the paper). Furthermore, using XML Schema (Fallside and Walmsley, 2004), we defined, an XML format for each type of document (the ellipses in Figure 1) - documents can be seen as the resources being described by the ontology.

The ontology gives a full categorization of the MEC resources universe; it shows the relations between the location of the resources (ex.: district archives), the information sources (ex.: passport's processes, almanacs) and the documents themselves (ex.: birth certificate, passport petition, event record).

Next subsections show different application scenarios using XQuery+SPARQL over the MEC resources.

6.1 Using XQuery+SPARQL

A visitor, wishing to explore MEC's resources, may be interested in searching for travelling details registered in *Fafe* related to the emigrant *Antonio Serra*; the details considered below are, *the date of departure,* and *the destination (target country)* .

Travelling information is obtained accessing passport records; an excerpt of a record of this kind is as shown in the following simplified:

```
<passportRecord  ...> ...
   <name>Antonio Serra</name> ...
   <place of="destination">Brasil</place>
   <place of="source">Portugal</place>...
   <date normDate="1866-11-13">...</date>
</passportRecord>
```

The XQuery+SPARQL query which yields to the desired information is:

```
for $d in doc("passRec.xml")/passportRecord
where $d/name = "Antonio Serra"
metadata $d in
   select ?d
   where {
     ?d  rdf:type  Passport_Record  .
     ?d  contained_in  ?p .
      ?p  rdf:type  Municipal_passport_record .
      ?p  acquired_in Fafe .
    }
 return
<pr> {
$d/passportRecord(place[@of="destination"]|date)
} </pr>
```

In the metadata clause of this query, the SPARQL query searches for documents contained in municipal passport records (stored in variable *?p*) which were acquired in *Fafe*.

6.2 Using Iterative XQuery+SPARQL

Suppose, now, the visitor is interested in searching for images and photos related to some families he knows. Let us assume a user from the city of *Braga*.

Family photos can be found in the respective family album. Instances of family albums are represented

by identifiers such as "*Freitas_Fam_Album*" for the "*Freitas*" family.

Image and photo documents have a structure/content similar to the following excerpt:

```
<image>
  <name>Family address of José Freitas</name>...
   <city>Braga</city>
   <date normDate="1885-01-01">...</date>
   <img>http://.../jose_freitas.jpg</img>   ...
</image>
```

Then, the query satisfying the current example is:

```
for $i in doc("image1.xml")/image
where $i/city ="Braga"
metadata $i in
     select ?i
     where {
          ?i rdf:type Images_and_photos  .
          ?i contained_in ?f .
          ?f rdf:type Family_album .
       FILTER msf:select(?f, (Freitas_Fam_Album,
                    Silva_Fam_Album)) .
      }
return $i/img
```

In the metada clause, the list of identifiers of the family albums found is displayed, as the answer computed in the third triple pattern of the SPARQL query (stored in *variable ?f*). The visitor may, then, immediately identify those belonging to families he knows. He can, then, select those albums using the *msf:select* function associated to the FILTER clause. In this query, the user selected albums from *"Freitas"* and "*Silva*" families.

7 CONCLUSIONS

In this paper, we propose to integrate the iterative paradigm for query construction into the XQuery+SPARQL semantic querying language. We believe this can help users to get the desired information.

We intend to create a prototype processing environment for the XQuery+SPARQL. We can use existing XQuery and SPARQL query processors integrating them with a special editor and result visualizer. We will, then, test this prototype and verify the usefulness of this approach using MEC assets, as described in Section 6.

REFERENCES

Berglund, A., Boag, S., Chamberlin, D., Fernandez, M., Kay, M., Robie, J., Siméon, J., 2007. XML Path Language (XPath) 2.0 W3C Recommendation 23 January 2007, URL: http://www.w3.org/TR/xpath20/.

Boag, S., Chamberlin, D., Fernandez, M., Florescu, D., Robie, J., Siméon, J., 2007. XQuery 1.0: An XML Query Language W3C Recommendation 23 January 2007, http://www.w3.org/TR/xquery/.

Fallside, D. and Walmsley, P., 2004. XML Schema Part 0: Primer Second Edition, W3C Recommendation 28 October 2004. URL: http://www.w3.org/TR/xmlschema-0/

Gançarski, A., Doucet, A., Henriques, P., 2006. AG-based interactive system to retrieve information from XML documents, *IEE Proceedings Software Journal*, Volume 153, Issue 2, p. 51-60, April 2006.

Gançarski, A., Henriques, P., 2006. A Formal Definition of Selection Operations that Extend XQuery with Interactive Query Construction. *International Conference in Web Information Systems and Technologies 2006* (Webist06), Setubal, Portugal, INSTICC Press.

Gançarski, A., Henriques, P., 2007. Using data together with metadata to improve XML information access. *International Conference in Web Information Systems and Technologies 2007* (Webist07), Barcelone, Spain, INSTICC Press.

Manola, F. and Miller, E., 2004. RDF Primer W3C Recommendation 10 February 2004. URL: http://www.w3.org/TR/rdf-primer/.

Prud'hommeaux, E. and Seaborn, A., 2007. SPARQL Query Language for RDF W3C Proposed Recommendation 12 November 2007. URL: http://www.w3.org/TR/rdf-sparql-query/.

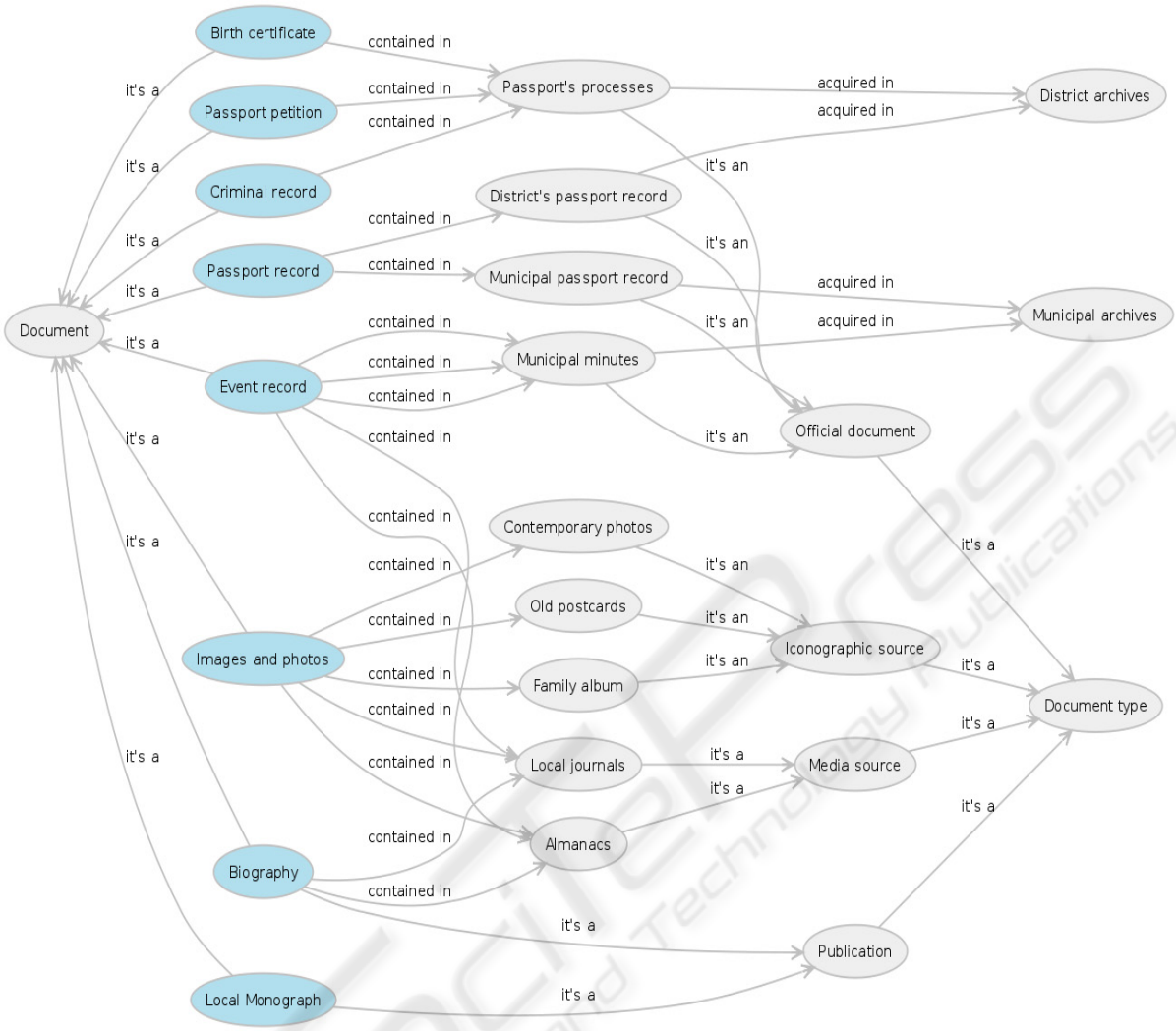

Figure 1: MEC Ontology.