A QUERY BY EXAMPLE APPROACH FOR XML QUERYING

Flávio Xavier Ferreira, Daniela da Cruz, Pedro Rangel Henriques
Department of Informatics, University of Minho
Campus de Gualtar, 4710-057 Braga, Portugal
flavioxavier@di.uminho.pt, prh@di.uminho.pt, danieladacruz@di.uminho.pt

Alda Lopes Gançarski, Bruno Defude
Institut Telecom, Telecom & Management SudParis, CNRS Samovar
9 rue Charles Fourier, 91011 Évry, France
Alda.Gancarski@it-sudparis.eu, Bruno.Defude@it-sudparis.eu

ABSTRACT: XML, as a general-purpose annotation system for creating custom markup languages, is becoming more and more important. XML annotations give structure to plain documents and help to interpret their content, making them human or machine readable. However, mechanisms to pretty-print those annotated documents or process them in order to extract information are crucial to make them useful. In a similar way, a collection of XML documents, without any tools capable of retrieving information from it, is useless. To search for specific elements in a marked up document we have, at least, two options: XPath and XQuery. However, the learning curve of these two dialects is high, requiring a considerable level of knowledge. In this context, the idea of Query-by-example can be an important contribution to make easier this learning process, freeing the user from knowing the specific query language details or even the document structure. In this paper, we describe our approach to QBE based on an sample document from the collection where the user specifies his needs. First, we focus on the choice of the adequate document to serve as a sample of the collection, based on different metrics computed over the document. Then, we discuss the query generation from the user needs' specifications, namely components (elements or attributes) selection and filtering.

Keywords: XML, XQuery, Query-by-example

INTRODUCTION

This paper presents an ongoing work addressing the problem of XML information access. The bigger the world wide collection of XML documents gets, the more relevant is the existence of an efficient search engine. These engines should be aware of the explicit structure of the documents. This has raised a research area called Structural Document Retrieval [6].

However, the creation of a query that yields valid results strongly depends on the user-friendliness of the search engine interface. As structured queries are powerful but complex to write (the user must have a deep knowledge of the query language as well as the document schema), some specialised editors have been developed to ease this task (XMLSpy[5], EditiX[1], oXygen[2]).

"Example is always more efficacious than precept". This statement, by Samuel Johnson, led Human Computer Interaction (HCI) researchers to suggest a new interaction paradigm called Query-by-example (QBE). Born in the context of database querying [7], typical QBE systems are based on the "fill in the blanks" approach. Zloof defined in [10] QBE as "a query language for use by non-programmers querying a relational database". QBE is based on the concept that the user formulates his query by filling in the appropriate skeleton tables the fields and/or restrictions on fields (the relational selection concept) he intends to search for.

We developed a QBE approach to XML using XQuery, which allows for the selection and restriction of entire paths (XML elements) directly on a sample document. We focus on two main aspects: the choice of the adequate document to serve as a sample of the collection; query generation from the user needs' specifications, namely component (elements or attributes) selection and filtering.

To present our approach, the remainder of this paper is organised as follows. We first present the languages usually used to query structured documents and we introduce the idea behind the

QBE approach for XML documents. We discuss then the choice of the sample document and how the query is generated by the user specifications. To conclude, we make some remarks and discuss the contribution of our approach, giving directions for future work.

QUERYING STRUCTURED DOCUMENTS

Queries for XML retrieval allow the access to certain parts of documents based on content and structural restrictions. Examples of such queries are those defined by XPath language [3] and XQuery [8], the standard proposed by the W3C. These languages are very expressive, allowing the specification of sophisticated structural and textual restrictions.

XQuery is formed by several kinds of expressions, including XPath location paths and "*for.. let.. where.. order by.. return*" (FLWOR) expressions based on typical database query languages, such as Structured Query Language (SQL). To pass information from one operator to another, variables are used. As an example, assume a document that stores information about articles, including title, author and publisher. The following query returns articles of author Kevin ordered by the respective title.

```
for $a in doc('articles.xml')/article
where $a/ author = `Kevin '
order by $a/title
return $a
```

XQuery operates in the abstract, logical structure of an XML document, rather than its surface syntax. The corresponding data model represents documents as trees where nodes may correspond to a document, an element, an attribute, a textual block, a namespace, a processing instruction or a comment. Each node has a unique identity.

However, structured queries construction is not always an easy process because, among other reasons, the user may not have a deep knowledge of the query language or of the documents collection structure. Moreover, after specifying a query, the user may get a final result that it is not what he expected. To solve these problems, many works are devoted to graphical user-friendly interfaces for query specification based on the Query-by-example paradigm, as explained in the next section.

QUERY-BY-EXAMPLE FOR XML

Through the years, the use of structured documents, like XML documents, in databases or as databases, led to an evolution of the QBE concept associated to XML retrieval. Often we are interested in searching for particular information in a document, but the learning of a new language (the query language) can be a challenge. So, the idea of generating queries through an example seems the perfect solution for this problem.

Most of the works [4, 9] adapt the relational QBE model by showing the XML Schema Definition (XSD) tree instead of the table skeleton. Our system also displays the XML Schema tree representation to the user. However, elements selection and restriction is done directly in an sample document, giving the user a complete indication of the information he is searching for. Moreover, differently from existing works, the user can, by using an sample document, query the entire collection.

Suppose the user is interested to search in a set of documents that represents a library. This library is composed by a set of books, where each book is defined by a title, an author, a publisher and a set of pages (illustrated in Figure 1).

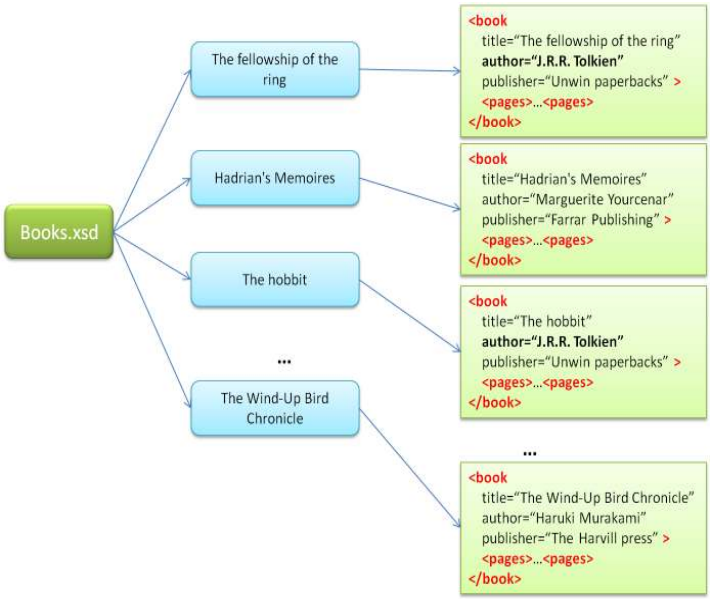

Fig. 1. A library collection

Now, suppose the user is interested to search, on this library, all books of the famous *J.R.R. Tolkien* author. If the user knows the XQuery language, he is able to write a query like this one:

```
for $x in doc(book_i)
where $x/book/@author=`J .R.R. Tolkie'
return $x/book
```

To get the desired results, the query is executed over each file corresponding to a book in the collection, replacing *book_i* by the corresponding file name (i=1 to D, being D the number of documents).

Using the QBE principle, instead of specifying the query in textual form, the user selects, in the interface showing an sample document, a *book* element. Then, he specifies the restriction by indicating an *author* element and associating to it the *J.R.R. Tolkien* value. The desirable resulting query should be exactly the same one returned by the previous query, i.e. books with titles *"The Fellowship of the Ring"* and *"The Hobbit"*.

CRITERIA FOR DOCUMENT SELECTION

The selection of the sample document is a focal point in our approach to QBE since the needs of the user are specified done over the sample document. This means that there must be a well founded logic behind the selection of the sample document from the documents conforming to the selected schema. We identified four metrics which should be combined to choose the sample document, as stated in what follows.

Document size: Big file sizes can slow down the system; also, smaller size files can contain too little information or elements to aid the user selection. This metric can be used as a delimiter to complement the others by not allowing a file bigger than a predefined size.

Number of elements/attributes: Taking into account the number of elements and attributes in the sample document is important. In one hand, if the file has too many components (elements or attributes), it can be too cluttered for the user to select his desired example. On the other hand, if the document has few components, it may not contain all those ones the user needs.

Number of different elements/attributes: To counteract some of the shortcoming of the previous metric, it may be interesting to look at the number of different elements and attributes in a file. This way, if a file contains almost all the elements and attributes present in the schema, the user gets a more complete variety of elements to specify his needs.

Diversity of Values: As stated before, the capacity of the user to see example data and not just the structure (schema) of the queried documents is the main innovation of our QBE approach. Therefore, a metric guaranteeing the diversity of data in the sample document is important. Having different values for the same element (or attribute) allows the user to better understand the fields in the document he is querying. However, similar to the other metrics, if there is too much diversity, the sample document may become too big.

As seen, each metric has its own merits and shortcomings, so they must be used together in a meaningfully way. The sample document should be diverse, which means that it must have a rich subset of the elements, attributes and possible values from the schema. However, it also must be a file contained in a predefined size. Therefore, we propose to use a combination of the 2nd, 3rd and 4th metrics restricted by a file size limitation (1st metric). We also intend to make a ranked list of possible sample documents, thus making easy for the user to retrieve the "second best choice" when the previous document suggested by the QBE system is not suitable.

COMPONENTS SELECTION AND FILTERING

Our approach to QBE has a selection part and a filtering part. The selection part corresponds to the *return* clause in XQuery, where the components to be retrieved are specified. The filtering part corresponds to the *where* clause and allows to filter out components from the result. In the user interface, the user may apply a filter to each component selection, thus generating a set of pairs <selection, filtering>. A full example is explained in the next section.

Each pair <selection, filtering> specified in the sample document yields a "*for.. where.. return*" query. For a pair, the query is inferred accordingly to the following pattern:

for $x in doc(doc_d)/nearest_path
where AND $_{i=0 \text{ to } N}$ *$x/xpath_filter*$_i$
return $x/xpath_selection

This pattern is applied to all D documents in the collection (d = 1 to D). Filters are specified as a sequence of logical AND operations connecting the N specified restrictions. The return expression takes the path which indicates the selected component. Selection and filter are unified by the nearest element common to both (*nearest_path*) in the *for* clause.

USER INTERFACE

Figure 2 shows the user interface of the QBE system we are developing. In this interface, the sample document is shown for user's selection and filtering specifications. Each pair <selection, filtering> is identified by the same colour with two different dark levels. The user starts by the selection phase, choosing the resulting component. He can, then, add restrictions to it by specifying components or values of components (to restrict their content to certain value).

Suppose the sample document of the collection in Figure 1 is *"The Hobbit"*. This sample document is shown in Figure 2. If the user specifies the

selection of the *title* element (shown in dark red colour), the corresponding query is, then:

for $x in doc(doc_i)/book
return $x/title

This query retrieves the authors from all the books in the collection (*i=1 to D*).

Now suppose the user restricts the value of the *publisher* element to "*Unwin paperbacks*" and the value of the *year* element to values greater or equal to 1937 ("≥*1937*"). These restrictions can be seen in Figure 2 in clear red colour. The generated query should be:

for $x in doc(doc_i)/book
where $x/publisher=` Unwin paperbacks'
and $x/year >= 1937
return $x/title

This query returns the titles from all the books published by *Unwin paperbacks* since *1937*.

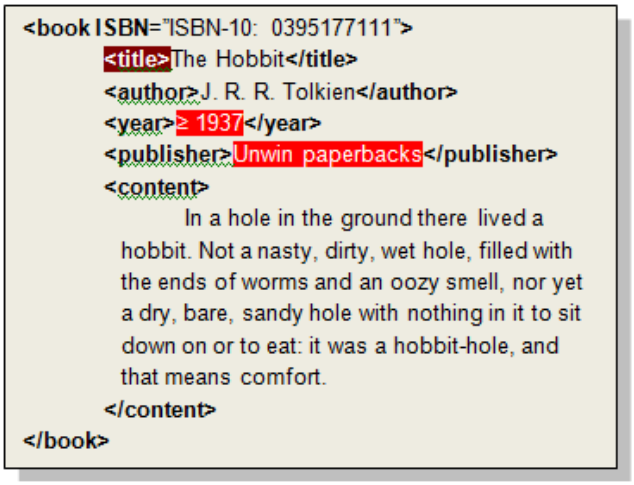

Fig. 2. Selecting and filtering in the QBE interface

CONCLUSION

The QBE approach we present helps the user in XQL query specification. By now, we concentrate our work in two main aspects: (1) how to choose the sample document; (2) what queries are generated depending on the user specifications. For the first aspect, we propose different metrics that express the adequacy of the document for the user to express his needs. As future work, we intend to formalize the combination of those metrics in a way to optimize the efficacy when the user expresses his queries.

Concerning the second aspect, we analyze the two main functionalities of XQuery: selection and filtering. This was done for one component selection. Next step is to extend the generated query pattern with the selection of several components. Moreover, the remaining XQuery functionalities will be studied, such as the creation of new elements in the result.

When completed, our query specification and processing system will be validated. We think about the Portuguese Emigration Museum information system [11] as a real application for testing with real users.

References

[1] EditiX XML Editor, last update 2008. http//www.editix.com.
[2] Oxygen XML Editor, last update 2008. http//www.oxygenxml.com.
[3] B. A., B. S., C. D., F. M., K. M., R. J., and S. J. Xml path language (xpath) 2.0 w3c working draft. http//www.w3c.org/xpath20/, 2005.
[4] D. Braga and A. Campi. Xqbe: A graphical environment to query xml data. World Wide Web, 8(3):287{316, 2005.
[5] L. Kim. The XMLSPY Handbook. John Wiley & Sons, Inc., New York, NY, USA, 2002.
[6] X. Lu. Document retrieval: A structural approach. Inf. Process. Manage., 26(2):209-218, 1990.
[7] R. Ramakrishnan and J. Gehrke. Database Management Systems, chapter 6. 2007.
[8] B. S., C. D., F. M., F. D., R. J., and S. J. XQuery 1.0: An xml query language. W3C working draft. http//www.w3c.org/TR/xquery/, 2005.
[9] J. H. G. Xiang Li1 and J. F. Brinkley1. XGI: A Graphical Interface for XQuery Creation. In American Medical Informatics Association Anual Symposium proceedings, volume 2007, pages 453{457. American Medical Informatics Association, November 2007.
[10] M. M. Zloof. Query-by-example: the invocation and definition of tables.
[11] Ferreira, F. X. and Henriques, P. R., Using OWL to specify and build different views over the Emigration Museum resources, National Conference XML Aplicações e Tecnologias Associadas 2008 (XATA08), Évora, Portugal.