Extending XQuery With Selection Operations to Allow for Interactive Construction of Queries

Alda Lopes Gançarski, Pedro Rangel Henriques

Universidade do Minho
DI, Campus de Gualtar, 4710-057 Braga, Portugal
{arl | prh}@di.uminho.pt

Abstract

XQuery is the standard language for querying XML documents using structural and content restrictions. In this paper, we propose to extend XQuery with selection operations that allow for the selection of the interesting subset of elements from each intermediate result of a query. To make this possible, intermediate results must be available during the construction of the query. This helps the user in building a query to retrieve the desired result. XQuery is being extended with a *Full-Text* language that allows to perform operations on text treating it as a sequence of words, units of punctuation, and spaces. This language includes a *score* clause that associates relevance measures to the results. The computation of these measures can be done using the method we also describe in this paper.

1 Introduction

Thanks to the benefits of electronic publishing, today an incredible amount of information is available to all. However, when a collection of documents is available, it is hard for the user to read all of them in order to get the needed information. Thus, available document collections are usually accompanied by an Information Retrieval (IR) system, which allows to automatically access documents with the desired information. It is the case, for example, of the documents published in the Web with respective search engines, like Google. Traditional IR consists of retrieving from a collection the relevant documents to a query, while returning as few as possible of non-relevant documents. Moreover, the resulting documents should be ranked by their relevance to the query (i.e., by decreasing probability of being relevant). A query is a natural language expression describing the desired subject. In traditional IR, documents and queries are represented by the set of terms, which are considered to describe the meaning of the corresponding text. The selection of those terms is called automatic indexing and consists of some operations on the text, like stemming (reduction of a word to its root) and elimination of non-informative words (stopwords). One possible textual representation is a vector in which the components store a measure of how representative is each term to the meaning of the text. This measure is based both on the term frequency in a document and the number of documents that contain the term. The relevance of a document with respect to a query is measured by the similarity between the document and the query. A common way to compute this similarity is the cosine between the vector that represents the document and the one that represents the query.

To take advantage from the structural information of XML documents, query formats for structured documents retrieval were enriched to access certain parts of documents. So, the user can access those parts based on content restrictions (comparisons with a value) and structural restrictions (referring to elements and their structural relations). The W3C Consortium is currently developing XQuery (Boag et al., 2005) to become the standard XML query language. XQuery is based on different query languages, such as XPath (Berglund et al., 2005) and XML-QL (Deutsh et al., 1998). These query languages can be referred to as data retrieval (instead of information retrieval) languages because the retrieval system searches for all documents that satisfy the query. Thus, there is no notion of relevance attached to the retrieval task. Some extensions to these languages appeared to allow for textual similarity restrictions returning a ranked list of elements by their relevance, as do works presented in the INEX workshop (Fuhr et al., 2004). XQuery is also being extended with the possibility of associating a *score* to an expression that verifies if some phrase exists in the content of some element or attribute. This function is included in the *Full-Text* language proposed by the W3C (Amer-Yahia et al., 2005).

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 may not know a priori exactly what to search.

Moreover, after specifying a query, the user may get a final result that it is not what was expected. To solve this problem, IXDIRQL (Gançarski & Henriques 2005) was defined as an extension to XPath, not only with textual similarity operations, but also with an interactive/iterative paradigm for building queries. With this paradigm, each operation specified by the user leads to an intermediate result which the user can access. This helps the user choosing the next operation, or changing an operation already introduced in the query, or selecting interesting subsets in the intermediate results, until reaching the adequate query and thus the desired result.
In this paper we extend XQuery with selection operations, as we did from XPath to IXDIRQL. Selection operations consist of restricting intermediate results, accessible by the user, to the subset of elements that satisfy the user. Before presenting the selection operations, sections 2 and 3 introduce XQuery and Full-Text search languages, respectively. Then, sections 4 and 5 explain how selection operations are introduced in XQuery, namely *select* and *judgeRel* operations, respectively. Despites the *score* functionality in the Full-Text language is independent of how relevancies are estimated, we propose a method to make those computations, which we introduce in section 6. The article finishes with a conclusion giving some directives for future work.

2 XQuery

XQuery is an XML query language in which queries are concise and easily understood. It is also flexible enough to query a broad spectrum of XML information sources, including both databases and documents. XQuery is a functional language, which means that expressions can be nested with full generality. 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 can correspond to a document, an element, an attribute, a textual block, a namespace, a processing instruction or a comment.

XQuery is formed by several kinds of expressions, like the following ones:

1. XPath location path expressions to extract nodes from document trees. They include filters, set and logical operators.
2. "*for..let..where..order by..return*" (FLWOR) expressions, based on typical database query languages, like the "*select..from..where*" expression of SQL. To pass information from one operator to another, variables are used.
3. Element or attribute constructors to build/produce new elements or attributes.
4. Conditional "*if..then..else..*" expressions.

To show an example, assume a document that stores information about articles, including title, author and publisher. The following query returns information about articles with an author and published by *Addison-Wesley*. Each article is returned in an *article_info* element that includes its title and its year if it was published before *1990*. In the result, *article_info* elements are ordered by the article's title.

```
for $a in doc("http://...") /articles/article[author]
where $a/publisher="Addison-Wesley"
order by $a/title
return <article_info> {$a/title} { if ($a/year<1990) then $a/year else ( )}</article_info>
```

Concerning the *for* clause, in a XQuery query, the document to query is given by the *doc( )* function. In the location path, the filter *[author]* restricts the articles to those that have an author. Variable *$a* refers to each article found. In the *where* clause, a restriction is made to the articles stored in variable *$a*. This restriction imposes the publisher of the article to be *Addison-Wesley*. The *order by* clause orders by title the articles found by the *where* clause. The result is constructed in the *return* clause. There is an element constructor that creates an *article_info* element for each article found and ordered so far. The *if..then..else* expressions makes that the element *title* of the corresponding article is included inside each *article_info* element, together with its *year* element if it has a value less than *1990*.

3 *Full-Text* Language

XML documents may contain highly-structured data (numbers, dates, etc.), unstructured data (untagged free-flowing text), and semi-structured data (text with embedded tags). Where a document contains unstructured or semi-structured data, it is important to be able to search that data using techniques such as the ones proposed in the *Full-text* language. Full-text queries are performed on text which has been tokenized, i.e., broken into a sequence of words, units of punctuation, and spaces. Tokenization enables functions and operators which work

with the relative positions of words (e.g., proximity operators). Tokenization also enables functions and operators, which operate on a part or the root of the word (e.g., wildcards, stemming). Full-text extends XQuery with *ftcontains* expressions and the inclusion of a clause *score* into the FLWOR expressions yielding to a *for..let..where..score..order by..return* expression.

3.1 ftcontains operator

The *ftcontains* function can be used anywhere a comparison can occur, like the *equal* operator. An *ftcontains* expression includes a location path to specify the nodes where the function is applied; the expression of the search strings to be found as matches; match options, such as the case sensitivity or indication to use stop words; and Boolean operators, like '||' (or) and '&&' (and). *ftcontains* returns a Boolean value true if there is some node in the path expression that matches the expression of the search strings. For example, the following query returns the author(s) of each article whose title contains a word with the same root as "*information*" and the word "*XML*".

for$a in doc("http://...") /articles/article
where $a/title ftcontains ("information" with stemming) && "XML"
return $a/author

The match option *with stemming* allows to search for occurrences of the root of some word.

3.2 *score* clause

The *score* clause generates a variable bound to a number representing a relevance score of full-text expression matches. The following example returns, for each article, the combined relevance of its title with respect to *"information"* and its introduction with respect to *"XML"*.

for $a in doc("http://...") /articles/article
score $s as $a/title ftcontains "information" and $a/introduction ftcontains "XML"
return $s

Syntactically, a *score* clause associates a variable to an expression. The expression is restricted to a Boolean combination of *ftcontains* expressions involving only *or* and *and* Boolean operators. The variable gets bound to a value in the range [0, 1], a higher value implying a higher degree of relevance. The value reflects the relevance of the match criteria and its calculation is left implementation-dependent. It is important to note that the *ftcontains* expression inside a *score* clause do not evaluate their arguments as regular XQuery expressions but use them interpreted.

4 Extending XQuery with Selection Operations

4.1 *select* function

The interactive paradigm of query construction is based on selection operations which consist of restricting intermediate results to the subset of elements that satisfy the user. There are three ways of making selection in location path expressions: using *select()* function, using XQuery pre-defined function *position()* and using *judgeRel* operator. We describe *judgeRel* later in section 5. The *select()* function selects the subset of interesting elements based on some criteria that the user does not know or can not specify by a filter. *select()* operates over the list of nodes obtained in the previous operation. The input to *select()* is a set of node identifiers (natural numbers) and the output is a list of nodes. For example, in *article/title[select({tit4, tit8})]*, *select* selects titles *{tit4, tit8}* from the ones given by the path *article/title*.

There is still the possibility of changing an intermediate result using the pre-defined function *position()*. Suppose an intermediate result has a large size and the user is interested in reducing it without a specific criteria. This can be the case of having a final result with the appropriate size for further processing. Using *position()*, the user can select the subset of the first *n* elements. For example, */article[author="Kevin"][position()<10]/title* yields to the title of the first 10 articles of author *"Kevin"* found in the collection.

To respect the interactive paradigm, variable values used in FLWOR expressions should be accessible any time the user wants to see them, as well as results inside *if..then..else* expressions and constructors results.

4.2 *judgeRel* operator

The ***judgeRel*** operator selects the subset of elements judged relevant by the user after using a *ftcontains* expression inside a *score* clause. For example, query

for $a in doc("http://...")/articles/article
score $s as $a/title ftcontains "XML"
order by $s
return $a/references

gives a list of references ranked by the relevance corresponding to the title of the article where they are cited. Using the *judgeRel* operator, the user can judge and select relevant elements during query construction. Consequently, the final result is relevant and can be directly used for further processing. The previous query would then be

for $a in doc("http://...")/articles/article
score $s as $a/title ftcontains "XML" judgeRel {tit4, tit8}
return $a/references

if *tit4* and *tit8* are judged relevant. When an *ftcontains* expression is followed by *judgeRel*, the relevance associated to the operator *ftcontains* is first calculated. Then, the relevance of each element selected by *judgeRel* becomes 1 and the relevance of the remaining elements becomes 0. As *judgeRel* was used, only elements associated with a relevance of value 1 (stored in the respective variable *$s*) will be part of the result.
Next section presents a method to compute the *score*.

5 *Full-Text score* Calculation Method

To compute the result of a *score* clause, first relevance measures associated to *ftcontains* expressions are calculated and then they are combined depending on the involved Boolean operators. The calculation method for *ftcontains* expressions is based in the traditional IR vector model. This choice is due to the fact that the vector model is very popular since it is simple and fast, being either superior or almost as good as a large variety of alternative ranking methods. Next section presents this model. Then, sections 6.2 and 6.3 present the calculation of relevancies associated to *ftcontains* and associated to Boolean operands, respectively.

5.1 Traditional Information Retrieval Vector Model

A term with a high frequency in a document is considered adequate to represent that document. Let $\{d_1, ..., d_N\}$ be a collection of documents; $\{t_1, ..., t_T\}$ a set of terms; n_i the number of documents where term t_i appears ($i=1,...,T$); $freq_{ij}$ the frequency of the term t_i in document d_j ($j=1,...,N$). The normalized frequency tf_{ij} of term t_i in document d_i is calculated by:

$$tf_{ij} = \frac{freq_{ij}}{\max_{i=1,...,T} freq_{ij}}$$

Terms which appear in a small fraction of the collection are good discriminates for that fraction. The discriminate power of a term t_i is calculated by:

$$idf_i = \log N/n_i$$

The product of these two measures, referred to as *tf.idf*, is frequently used to represent the terms of the collection for relevance estimations. Documents and natural language expressions of queries can be represented by a vector having as components the correspondent tf.idf measures of terms. In this vector model, the relevance of a document with respect to a query is the result of a correlation function between the corresponding vectors. A simple, efficient and, thus, frequently used correlation function is the cosine, which gives the relevance of document d_j ($j=1, ..., N$) with respect to query q by:

$$sim(d_j, q) = \frac{d_j \times q}{|d_j| \times |q|}$$

Let w_{ij} be the measure *tf.idf* of term t_i ($i=1..T$) in vector d_j ($j=1..N$). The relevance is, then:

$$sim(d_j, q) = \frac{\sum_{i=1}^{T} w_{ij} \times w_{iq}}{\sqrt[2]{\sum_{i=1}^{T} w_{ij}^2} \times \sqrt[2]{\sum_{i=1}^{T} w_{iq}^2}}$$

5.2 Relevance Associated to *ftcontains*

Suppose the *score* clause of the following query that associates a relevance to titles with respect to the subject *"information"*:

for $a in doc("http://...")/articles/article
score $s as $a/title ftcontains "information"

For this kind of search, a method was proposed in (Gançarski & Henriques, 2005), inspired by different works including some participating in INEX, to calculate the relevancies based on the vector model. In this model, to adapt to the structured format of documents, tf.idf measures are taken from the set of elements, instead of documents, of the collection. Let *Ne* be the number of elements in the collection; n_i the number of elements where term t_i appears ($i=1..T$); $freq_{ij}$ the frequency of term t_i in element e_j ($j=1..Ne$). The tf.idf representation is now called *tf.ief* where:

$$tf_{ij} = \frac{freq_{ij}}{\max_{l=1..T} freq_{lj}} \quad \text{and} \quad ief_i = \log Ne/n_i.$$

According to the use of the match option *with stemming* or *without stemming*, the same method can be used with or without reducing words to their root, respectively. For simplicity, we do not cover here the remaining match options.

When '||' and '&&' operators are used, the partial relevancies associated to each match expression can be given a probabilistic interpretation. An event is the fact that an element is relevant to a match expression. The relevance is the probability associated to the event, i.e. the probability that an element is relevant. Thus, to calculate the final relevance associated to *ftcontains*, partial relevancies can be combined using the following probabilistic formulas for disjunction and conjunction of events, respectively:

$$P(p_1 \vee \ldots \vee p_n) = \sum_{i=1}^{n} (-1)^{i-1} \left(\sum_{1 \le j_1 < \ldots < j_i \le n} P(p_{j1} \wedge \ldots \wedge p_{ji})\right) \qquad (F.1)$$

$$P(p_1 \wedge \ldots \wedge p_n) = P(p_1) \times \ldots \times P(p_n), \text{ assuming } p_1, \ldots, p_n \text{ are independant.} \qquad (F.2)$$

Formula *(F.1)* is used for the '||' operator and formula *(F.2)* is used for the '&&' operator. For example, the *ftcontains* expression

/articles/article ftcontains "XML" && "Applications"

has two match expressions: *"XML"*, which we call *m1*, and *"Applications"*, which we call *m2*. This *ftcontains* expression gives, for an article *a1*, a final relevance based on the partial relevancies *P(a1,m1)* and *P(a1,m2)*. These relevancies correspond to the fact that article *a1* is relevant with respect to the match expression *m1* and the fact that the same article is relevant with respect to the match expression *m2*, respectively. The final relevance associated to article *a1* is calculated, using formula *(F.2)*, by:

$$P((a_1, m_1) \wedge (a_1, m_2)) = P(a_1, m_1) \times P(a_1, m_2)$$

5.3 Relevancies Associated to *and* and *or* Boolean Operators

To combine relevancies from different *ftcontains* expressions, a probabilistic interpretation of such relevancies is made, as for *ftcontains* expressions (Section 6.2). For example, in the *score* clause of the query

for $a in doc("...")/articles/article
score $s as $a ftcontains "information" and $a//reference ftcontains "XML"

there are two *ftcontains* expressions, one for articles and other one for references. Suppose that article a_1 has relevance $P(a_1, m_1)$ interpreted as the probability that a_1 is relevant with respect to match expression m_1 "*information*". Suppose also that references r_{11} and r_{12} are cited in article a_1 and have relevancies $P(r_{11}, m_2)$ and $P(r_{12}, m_2)$, respectively, with respect to match expression m_2 "*XML*". The set of m_{11} and m_{12} references may be called r_1. $P(r_1)$ is the probability that the set r_1 is relevant with respect to m_1, i.e., that the set of references of article a_1 is relevant. $P(r_1, m_2)$ is calculated, using formula *(F.1)*, by:

$$P((r_{11}, m_2) \vee (r_{12}, m_2)) = P(r_{11}, m_2) + P(r_{12}, m_2) - P(r_{11}, m_2) \times P(r_{12}, m_2)$$

In the result, both relevancies of article a_1 and set of references r_1 must be combined to get the final relevance associated to each article. This is done, using formula *(F.2)*, by:

$$P((a_1, m_1) \wedge (r_1, m_2)) = P(a_1) \times P(r_1)$$

which is the probability that article a_1 and its set of references r_1 are both relevant.

5.4 Other Relevance Estimations

We suggest that the score measures associated to the elements of a query result should be taken into account if new score calculations are made over this result if it used inside another query. In this case, the combination of the two scores for each element is made using formula *(F.2)*.
The *distinct_values()* pre-defined function gives as result the distinct nodes inside the set of nodes it receives as argument. If the nodes of the argument are associated to a score variable, the resulting nodes are associated to the same score. However, if some value appears several times, in the result it will be associated to a relevance measure that is the combination of the score values of its different occurrences estimated using formula *(F.1)*.

6 Conclusion and Future Work

XQuery is becoming the standard query language for XML. This paper suggests the use of an interactive paradigm for iterative and incremental query construction and presents an extension to XQuery to allow the selection operations over intermediate results. This helps the user, not only in choosing the operations that yield the desired answer, but also in restricting each intermediate result to the subset of nodes that pleases the user. As future work, we intend to build a prototype for the interactive edition and processing of XQuery, in a similar way we did for IXDIRQL (Gançarski & Henriques, 2005). The evaluation of query operations is incremental such that, after some change in the query, only calculations depending on that change are made. For that, a formal definition of the extended XQuery language will be given to LRC, an attribute grammar-based syntax-directed editor and language processor generator (Kuiper & Saraiva, 1998) to build an incremental environment for query edition. In the prototype, *score* calculations of Full-Text language will be based on the method proposed in this paper.

Acknowledgement
The authors are grateful to the Portuguese *Fundação para a Ciência e a Tecnologia* for the financial support.

References

Amer-Yahia S., Botev C., Buxton S., Case P., Doerre J., McBeath D., Rys M. and Shanmugasundaram J. (2005). XQuery 1.0 and XPath 2.0 Full-Text Working Draft 04 April 2005, from http://www.w3.org/TR/2004/WD-xquery-full-text-20040709/.

Berglund A., Boag S., Chamberlin D., Fernandez M., Kay M., Robie J. and Siméon J. (2005). XML Path Language (XPath) 2.0 W3C Working Draft 04 April 2005, from http://www.w3c.org/xpath20/.

Boag, S, Chamberlin D., Dernadez M., Florescu D., Robie J. and Siméon J. (2005), XQuery 1.0: An XML Query Language W3C Working Draft 04 April 2005, from http://www.w3c.org/TR/xquery/.

Deutsh, A., Fernandez M., Florescu D., Levy A., and Suciu D. (1998), XML-QL: A query language for XML. W3C Note. http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/.

Fuhr N., Lalmas M., Malik S. and Szlávik Z. Editors (2004), INEX: Initiative for the Evaluation of XML Retrieval, DELOS Network of Excellence in Digital Libraries, Schloss Dagsthul, Germany.

Gançarski A. and Henriques P. (2005). A processing environment for the IXDIRQL XML query language, *Proceedings of the IADIS Virtual Multi Conference on Computer Science and Information Systems (MCCSIS 2005)*.

Gançarski A. and Henriques P. (2003). IXDIRQL: an Interactive XML Data and Information Retrieval Query Language. *Proceedings of the 7th ICCC/IFIP International Conference on Electronic Publishing*. Guimarães, Portugal, pp. 316-323.

Kuiper M. and Saraiva J. (1998). LRC: A generator for incremental language-oriented tools. *Proceedings of the 7th International Conference on Compiler Construction, Lisbon, Portugal,* LNCS 1383, 298-301. Springer.