CIRQuL - Complex Information Retrieval Query Language[*]

Vojkan Mihajlovic, Djoerd Hiemstra, Peter M.G. Apers

University of Twente, CTIT, Enschede, The Netherlands

Abstract

In this paper we will present a new framework for the retrieval of XML documents. We will describe the extension for existing query languages (XPath and XQuery) geared toward ranked information retrieval and full-text search in XML documents. Furthermore we will present language models for ranked information retrieval applied to the XML and describe the ultimate goal of our research.

1 Introduction

Information Retrieval (IR) theory is developed to overcome the task of searching for information in flat unstructured documents. The theory and the tools used in conventional IR systems usually completely disregard the structure of a document. However, with rapid proliferation of structured, and especially semi-structured documents (i.e. XML), a new research area for the IR community has been drawn. It can be defined as follows: formalizing a broad powerful query language that can be used for querying XML documents both, on structure and content, and building a powerful execution engine, that will be able to retrieve a (ranked) list of XML documents or fragments of XML documents, given the query.
The definition of XML as a structured (mark-up) language [5] implies the presence of structure information, besides content. Therefore, the data in XML can be displaced into two broad categories: (1) data that represents information about XML document structure, and (2) data that represents content information in XML documents. Furthermore, content of XML documents is much more complex than the content of a flat text documents. This is because each content word has its scope which is defined by XML structure, and bears a different kind of information depending on its position in XML document. Thus, XML brings more opportunities for modeling information in XML documents, as well as querying and searching tasks. It also enables more precise definition of search intentions of a user, in terms of defining the search space for computing relevance score and defining the retrieved portions of XML document. In other words, richer query languages have to be formalized, in respect to standard (flat-file) IR systems, to enable above mentioned functionalities.
The structure queries can be expressed using XQuery [8] and XPath [7] (which is an integral part of XQuery) capabilities. Queries in the traditional IR style or Full-Text Search (FTS) [11, 9] queries are currently gaining more popularity in the research community. However, most of the proposed XML query languages and retrieval engines, or XML database management systems, are oriented toward single aspect of XML documents (even XPath and XQuery are focused on structure part). Only few of them try to treat structure part in conjunction with ranked retrieval and FTS. For example in [12, 13] FTS in semi-structured documents is used, while in [2] authors try to support the retrieval of relevant parts of a document using containment queries. In [16, 14] another approach is presented which mostly supports IR-like query execution over XML databases, using XQL and predefined index objects for term weighting. This approach uses extra information to query XML documents (DTD or XML Schema) and does not support powerful FTS queries.
Recently, a proposal for XML full-text search was drafted, consisting of requirements [11] and use-cases [9]. Together with already developed structured query languages and current IR state of the art, the icon of the future query language has been established. Therefore, our aim will be to develop a powerful complex query language on top of a database that will enable us to define database operators. Moreover, these operators will be able to execute all three types of queries, and provide the user with the desired information. For ranking we will use statistical Language Models (LM) from the IR, extended with new capabilities to enable modeling of complex query expressions and the structure of XML documents.
The goal of the complex query syntax is to enable composition of proximity and Boolean queries performed on simple or distinct parts of XML documents. As an illustration we will give three examples, namely for: (1) proximity search, (2) Boolean search on different parts of XML documents, and (3) combination of the previous two cases. To form queries we will use XPath-like syntax that we will explain in details in next section:

```
//par[IR(boat near shark near teeth)];
/book[IR(title:(man and sea)
                 and .//par:shark)];
/book/title[IR((old adj man)
                 or (bell near tolls))].
```

[*]The work presented in this paper is funded by the Netherlands Organization for Scientific Research (NWO)

Here we will define some terms that we will use in this paper. Following the conventional IR theory ([10]) we can define XML *documents* as separate XML files, and XML *collection* as a set of XML documents (files), with additional meta-indexes stored in the database.[1] At the lowest level of granularity, we can define *content* of an XML document which represent all the words in the document that are not mark-up (i.e. `text()` nodes). On a higher level we can define XML *elements* that correspond to one of XML `tags` and all the encompassed information in it (including other descendant tags, their attributes and content information, as well as processing instructions and comments). Since there might be more than one sibling element with the same tag name in an XML tree model, we additionally introduce the concept of XML *fragments* which corresponds to the `node-set` construct in XPath. Using the notion of XML fragment we can define XML documents and even XML collection as "high level" XML fragments. For query formulation we will use XPath and extend it with complex IR query facilities, as we will see in the next section. In section three the language models used for XML fragments relevance score computation will be explained. Finally, the closing section will summarize the benefits of the proposed complex query language (CIRQuL) and give a notion of future work.

2 Structured Query Language Extensions for IR-like and FTS Queries

We will start from the XPath syntax, since we consider XPath as a good base for introducing the complex query extension. Furthermore, XPath is included in the XQuery definition, and therefore the complex query language we propose can be easily incorporated as an extension to XQuery. For the notation we will use Syntax Graphs (SG) as a graphical representation of Extended Backus-Naur Form (EBNF).

2.1 XPath Capabilities

The expressions defined in XPath [7] are evaluated against a tree model which represents the logical structure of an XML document. The basic types of expressions in XPath are location paths, while it's main goal is to enable traversing the tree model of an XML document to find a so called `node-set`. The notion of `node-set` represents nodes that are obtained by XML tree model traversal, and is one of the basic data types in XPath. Other data types that are supported in XPath are described in Table 1.

The basic definition of XPath is depicted in Figure 1. Part of the syntax that is marked by the dashed rectangle represents what can be done in each XPath step. Furthermore, there is a clear distinction between the structure part of the XPath expression (`axis`), tests performed on the fragment of XML document structure obtained by structure part of a query (`node_test`), and a content part for data manipulation and `node-set` filtering within the XML fragment (`predicate`).

Data type	Description
`node-set`	A collection of nodes (no duplicates)
`boolean`	`true` or `false`
`number`	A floating point number
`string`	A sequence of characters

Table 1: XPath data types

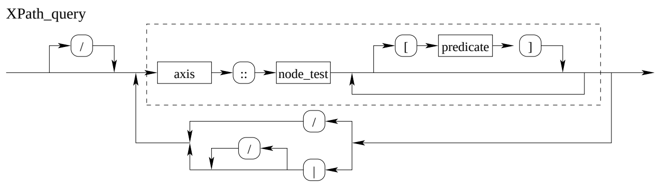

Figure 1: Syntax graph for XPath query definition

The result of each XPath step is an XML fragment which represents a *context node(s)* for the following XPath steps (if any). In explaining the new complex query language syntax we will start from `predicate`, since we consider it as a proper place for the complex IR query extension.

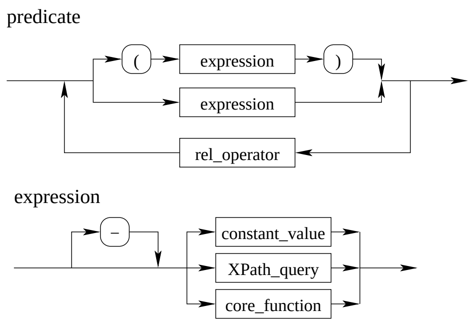

Figure 2: Syntax graph for `predicate`

The aim of the `predicate` (see Figure 2 for syntax specification) is to enable some basic `node-set` filtering using content of XML elements. Predicates can contain a number of expressions whose syntax is roughly described in the bottom part of Figure 2. The syntax of `core_function`, as a part of an `expression` syntax, is given in Figure 3.[2] Relational operators used for combining core functions and XPath expressions are given in EBNF below:

```
  rel_operator := or|and|=|!=|<=|>=|<|>
                                    |+|-|div|*.
```

The `constant_value` parameter, depicted in Figure 3, represents the constant value of a type defined by one of the basic XPath data types.

[1] In some cases distinction between XML documents stored into a database cannot be established due to the fact that XML documents are stored as a part of a large XML collection with one extra global root node, like in [3] for example.

[2] Here we generalized the complex syntax of `expression` to be able to represent its functionality. For full coverage of the `expression` symbols refer to [7].

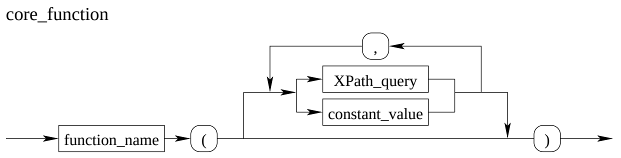

Figure 3: Syntax graph for `core_function`

2.2 Extending XPath Toward IR Capabilities

Although some query capabilities that are highly related to content retrieval exist in XPath (e.g. string functions like `contains`, `starts-with`, `substring`), they are hardly sufficient for powerful information retrieval. This especially stands for proximity queries (e.g. queries on near and adjacent terms) and the need for ranking of retrieved XML fragments.
Furthermore, XPath (XQuery) is impotent for expressing queries on word order (except `starting-with` clause), or queries that use thesaurus and stemming. Since these queries form the base for a ranked IR and FTS, in this paper we introduce an extension for XPath to enable the formulation of queries in a Complex Information Retrieval Query Language (CIRQuL).

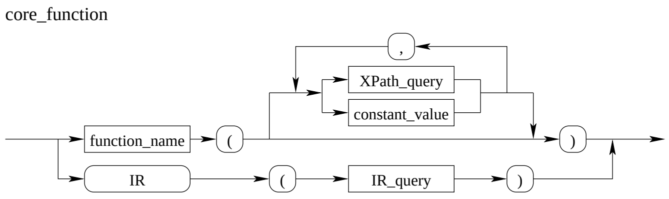

Figure 4: Syntax graph for complex `core_function`

We will start from the syntax graph of core functions depicted in Figure 4. Comparing it with the Figure 3 it can be noticed that the only difference is in yet another path with syntax nodes named `IR` and `IR_query`. We introduce an additional core function to XPath syntax, named `IR`, which returns a ranked fragments of XML documents (collection). The fragments are ranked according to the score functions that are defined in the next section.

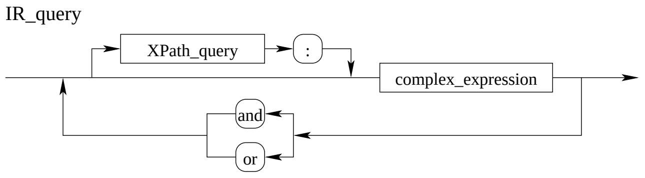

Figure 5: Syntax graph for complex `IR_query`

As depicted in Figure 5, to enable more expressive power for the task of ranked IR we introduced a recursive call of `XPath_query` in the complex query formulation. Thus, we enabled combination of more complex IR expressions on different XML fragments that are typically contained inside XML fragment defined by the XPath part of an `IR_query`. The combination of complex expressions can be expressed using `and` or `or` operators, and as we will see later, these operators have similar functionality as operators with the same name inside the `complex_expression`.

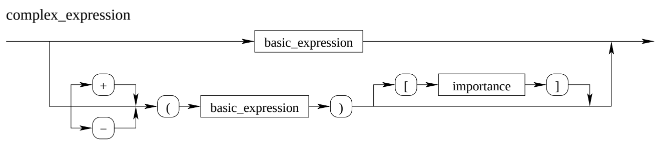

Figure 6: Syntax graph for `complex_expression`

The syntax of `complex_expression` is given in Figure 6. The complex expression consists of one or more basic expressions combined with brackets, inclusion (`+`) and exclusion (`-`) operators, and an `importance` attribute. Brackets are used to group terms in a simple expression. The inclusion and exclusion operators are used for specifying that the XML fragment must or must not contain `basic_expression`, respectively. The `importance` attribute is used to define the importance of an expression among all the other expressions. In cases where the expressions' importance is not specified it is equally distributed to every `basic_expression` (e.g. $1/\#(basic\_expression)$).

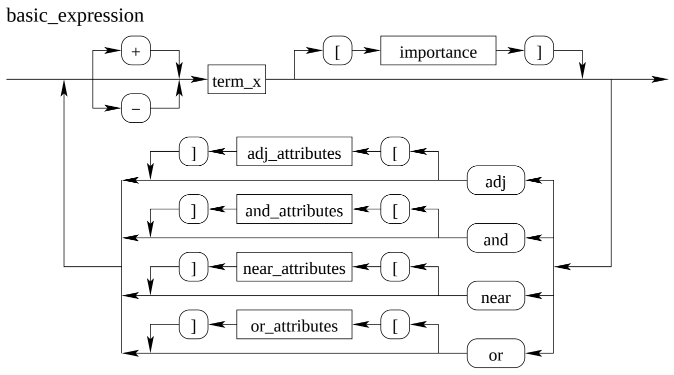

Figure 7: Syntax graph for `basic_expression`

A Basic expressions is formed using traditional boolean IR operators: `and` and `or`, and proximity operators: `adj` (adjacent) and `near`. Additionally, to enable full power of full-text search as defined in [9, 11] we introduce operators, operator attributes and term attributes. The syntax of a basic expression is depicted in Figure 7, where the `term_x` is defined as:

```
  term_x := term ['['term_attributes']'].
```

The function of operators `+` and `-` and attribute `importance` is the same as described for complex expressions, except its scope, which is now translated to query terms, instead of basic expressions. For example, the query:

```
  //par[IR(+(old adj man) and -shore)
```

should find all the paragraphs which contains the phrase 'old man' and do not contain term 'shore', while the query:
`//par[IR((old adj man)[0.2] and shark[0.8])]` stresses the importance of a term 'shark' against a phrase 'old man'.
For the operators we introduce next attributes:

```
adj_attributes := word_order[,skip_elem]
and_attributes :=
          distinct_element[,skip_elem]
near_attributes :=
          win[,word_order[,skip_elem]]
or_attributes :=
          distinct_element[,skip_elem]
```

Operator attribute `word_order` defines whether query word order should be used as a criteria for `adj` and `near` search, while attribute named `distinct_element` defines whether the `and` or `or` search should be performed in all the sibling elements (`distinct`) or each of them separately (`same`). For example we can compose next complex query:

```
/book/[IR(author:(Ernestadj[order]
Hemingway) and .//par:(big and[same] shark))]
```

This query will search for all the books whose author is Ernest Hemingway and which contain paragraphs such that inside some of the paragraphs terms big and shark can be found. If we use the `distinct` attribute instead of the `same` attribute for the `and` operator we will obtain all the books where 'big' and 'shark' can be found in distinct paragraphs of a book element.
We use the operator attribute `win` to define the window for near search (e.g. `boat near[10] shark`), and `skip_elem` attribute is introduced to cover FTS use-cases described in chapter 14 of [9].
Furthermore, to enable most of the FTS use-cases [9], we introduce next set of term attributes:

```
{case_sensitive, diacritics_support,
 stemming, word_division, position,
 term_expansion, term_prefix, term_infix,
 term_suffix, skip_tag}.
```

Each one of the term attributes will have values, defined to support FTS use-cases. For example:

```
case_sensitive :=
   exact_case|lower_case|upper_case|any_case;
term_expansion :=
   no_semblance|thesaurus_narrow
   |thesaurus_broad|pronunciation|spelling;
term_suffix := suffix '(' max [,min] ')'.
```

We can exert that almost every term attribute can be resolved using a complex lexicon (thesaurus). As an illustration for the usage of complex lexicon we can give next example query:

```
IR(boat near anchor[use_thesaurus]),
```

which can be expanded to:

```
IR(boat near (anchor or kedge or grapnel)).
```

Here terms `kedge` and `grapnel` represent terms with the same meaning as `anchor` term, captured by the complex lexicon (thesaurus).

Using the complex lexicon and database accessories, query expansion and rewriting can be performed in order to avoid adding unnecessary complexity in logical operators for explicitly expressing term attribute values. Due to the space limit we will not elaborate more on this.

3 LM for IR in XML documents

The basic idea behind the language modeling approach to information retrieval is to assign probabilities to relevance of each document (D) when the query Q is specified using query terms ($Q = q_1, q_2, ..., q_n$). Here, we will consider XML elements (E) instead of documents as a basic retrieval unit. Thus, using Bayes' rule we can express the relevance score like ([1]):

$$P(E|q_1, q_2, ..., q_n) = \frac{P(q_1, q_2, ..., q_n|E)P(E)}{P(q_1, q_2, ..., q_n)} \tag{1}$$

where the value of a denominator depends only on query formulation and thus might be ignored for a single query relevance score computation. Furthermore, if we assume uniform prior for all the elements in a collection, the prior $P(E)$ should be ignored. In other words we assumed that the elements are equally likely to be relevant in absence of a query. Since this is not usually the case, we must use some computations and estimate or learn the value of $P(E)$. For this purpose the relative size of an element in an XML collection might be used (similarly to relative document size in a collection in [10]). Furthermore, if we assume that the query terms are independent we can isolate the single terms of a complex expression in the numerator and express the probability that a query term q is drawn from a single element ($e \in E$): $P(q|E)$.
In defining LMs for complex CIRQuL expressions we will start from the simple query consisting from one single query term q. Following the traditional statistical LM formalism we can define the relevance assessments of an element e given the query term q as:

$$P(q|e) = \frac{tf(q, e)}{\sum_t tf(t, e)} \tag{2}$$

Here, $tf(t, e)$ denotes term frequency of a term t in an XML element e, and $\sum_t tf(t, e)$ represents the total number of terms in an XML element e, while the equation (2) define the probability that a term in element e is q. However, considering the hierarchical organization of XML documents we might alternatively define this expression as:

$$P(q|e) = \frac{1}{\sum_t tf(t, e)} \sum_{e_i \in dsc(e)} \frac{tf(q, content(e_i)) \sum_t tf(t, e_i)}{\sum_t tf(t, content(e_i))} \tag{3}$$

In this equation we compute the term frequency of a term inside the content of each descendant node of a current context node($tf(q, content(e_i))/\sum_t tf(t, content(e_i))$), multiply it by a bias factor $\sum_t tf(t, e_i)/\sum_t tf(t, e)$ (similar to augmentation factors in [2, 16] and mixture parameters in [10]), and sum the resulting values.
Using equations (2) and (3) as a starting point we can define more complex models for operators. Thus, for operators

`or`, `and`, `near`, and `adj`, that form the basic expressions we will use next equations:

$$P(q_1 \; or \; q_2 \; or \; ... \; or \; q_n|e) = \frac{1}{n}\sum_{i=1}^{n} P(q_i|e) \qquad (4)$$

$$P(q_1 \; and \; q_2 \; and \; ... \; and \; q_n|e) = \prod_{i=1}^{n} P(q_i|e) \qquad (5)$$

$$\begin{aligned} &P(q_1 \; near \; q_2 \; near \; ... \; near \; q_n|e) = \\ &\quad = P(q_1|e)P(q_2|q_1, e) \; ... \; P(q_n|q_{n-1}...q_1, e) \\ &= \frac{P(q_1|e)}{\sum_t tf(t,e)} P(q_1 \; near \; q_2) \; ... \; P((q_{n-1}, ..., q_1) \; near \; q_n)) \quad (6) \end{aligned}$$

$$\begin{aligned} &P(q_1 \; adj \; q_2 \; adj \; ... \; adj \; q_n|e) = \\ &\quad = P(q_1|e)P(q_2|q_1, e) \; ... \; P(q_n|q_{n-1}...q_1, e) \\ &= \frac{P(q_1|e)}{\sum_t tf(t,e)} P(q_1 \; adj \; q_2) \; ... \; P((q_{n-1}, ..., q_1) \; adj \; q_n)) \quad (7) \end{aligned}$$

Here $P((q_{n-1}, ..., q_1) \; near \; q_n)$ represents the number of occurrences of a term q_n near other terms $(q_{n-1}, ..., q_1)$ divided by the size of the window defined in `near` operator attributes and the relative position of previous terms $(q_{n-1}, ..., q_1)$ inside the element e. Similarly, $P((q_{n-1}, ..., q_1) \; adj \; q_n)$ denotes the number of occurrences of a term q_n adjacent to one of the previous terms $(\{q_{n-1}, ..., q_1\})$ divided by 1 or 2 in respect to the values of `adj` operator attributes (`word_order`).
Furthermore, to enable execution of operator `or` or `and` with `distinct` attribute value, we will use weighted (augmented) sum over all the sibling elements (e_i) that are in a fragment f $(\in F)$:

$$P(Q|f) = \sum_{e_i \in f} (P(Q|e_i) \sum_t tf(t, e_i)) \frac{1}{\sum_t tf(t,f)} \qquad (8)$$

Since in our complex query syntax we allowed the usage of `or` and `and` terms for forming complex expressions, we will support their representation in a logical algebra in the same fashion as for their counterparts defined in equations (4) and (5). However, instead of single query terms q_i depicted in these equations we will use term Q_i which stands for basic query expression. Using LMs defined in equations (2) - (8) we will try to develop all the operators in a logical algebra. Together with the rewriting rules defined for the query language we will be able to perform score computation for all the XML fragments in a database collection.[3]

4 Conclusions and future work

In this paper we presented a Complex Information Retrieval Query Language (CIRQuL) whose goal is to enable ranked retrieval and full-text search in XML documents. The language is proposed as an extension to XPath, with the introduction of IR operators, and operator and term attributes. Furthermore, we explained how statistical language models can be used to support the relevance score computation for the simple one-term queries, as well as for the complex queries composed with `adj`, `and`, `near`, and `or` operators.
The ultimate goal of our future research will be to build a stable database management system with a powerful logical algebra based on language models that will support the execution of complex queries defined in CIRQuL. Furthermore, we will use scalable storage schema (similar to [3]) on physical level to support fast execution of logical algebra operators.

[3]At current stage of the development we can not give the estimation for the complexity of CIRQuL query evaluation when using equation (2) or equation (3).

References

[1] D. Hiemstra “A Database Approach to Content-based XML retrieval” In Proc. of the First Annual Workshop of INEX, pp. 53-58, Schloss Dagstuhl, Germany, 2002.

[2] T. Grabs, H-J. Schek, “ ETH Zrich at INEX: Flexible Information Retrieval from XML with PowerDB-XML” In Proc. of the First Annual Workshop of INEX, pp. 35-40, Schloss Dagstuhl, Germany, 2002.

[3] T. Grust, M. van Keulen, J. Teubner “Staircase Join: Teach A Relational DBMS To Watch Its (Axis) Steps” Proc. of the 29th Intl. Conference on Very Large Databases (VLDB), Berlin, 2003

[4] T. Grabs, H-J. Schek, “Generating Vector Spaces On-the-fly for Flexible XML Retrieval” In Proc. of the 25th ACM SIGIR Conference, Tampere, 2002.

[5] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, “Extensible Markup Language (XML) 1.0. (Second Edition)” Technical Report (TR), W3C, 2000. http://www.w3.org/XML/TR/REC-xml

[6] A. Skonnard, M. Gudgin, “Essential XML Quick Reference: A Programmer’s Reference to XML, XPath, XSLT, XML Schema, SOAP, and More” Addison-Wesley, 2001.

[7] J. Clark, S. DeRose, “XML Path Language (XPath)” Version 1.0, TR, W3C, 1999. http://www.w3.org/TR/xpath

[8] S. Boag, D. Chamberlin, M.F. Fernandez, D. Florescu, J. Robie, J. Simeon, “XQuery 1.0: An XML Query Language” TR, W3C, 2002. http://www.w3.org/TR/xquery/

[9] S. Amer-Yahia, P. Case, “XQuery and XPath Full-Text Use Cases” TR, W3C, 2003. http://www.w3.org/TR/xmlquery-full-text-use-cases/

[10] D. Hiemstra “Using Language Models for Information Retrieval” PhD thesis, University of Twente, 2001.

[11] S. Buxton, M. Rys, “XQuery and XPath Full-Text Requirements” TR, W3C, 2003. http://www.w3.org/TR/xmlquery-full-text-requirements/

[12] H. Ahonen, B. Heikkinen, O. Heinonen, J. Jaakkola, P. Kilpelainen, G. Linden, H. Mannila, “Constructing Tailored SGML documents” In Proc. of SGML Finland, pp. 106-116, Espoo, Finland, 1996.

[13] H. Hosoya, B. Pierce, “Regular Expression Pattern Matching for XML” In Proc. of ACM Symposium on Principles of Programming Languages, pp. 67-80, London, 2001.

[14] J.E. Wolf, H. Florke, A.B. Cremers, Searching and Browsing Collections of Structural Information” In Proc. of IEEE Advances in Digital Library, pp. 141-150, USA, 2000.

[15] T. Schlieder, F. Naumann, “Approximate Tree Embedding for Querying XML Data” In ACM SIGIR 2000 workshop on XML and IR, pp. 53-67, Athens, 2000.

[16] N. Fuhr, K. Grossjohann, “XIRQL: A Query Language for Information Retrieval in XML Documents” In Proc. of the 24th Annual ACM SIGIR Conference on Research and Development in IR, pp. 172-180, USA, 2001.

[17] R.W.P. Luk, H.V. Leong, T.S. Dillon, A.T.S. Chan, W.B. Croft, J. Allan, “A Survey in Indexing and Searching XML Documents” Journal of the American Society for Information Science and Technology, vol. 53, pp. 415-437, 2002.