A User Interface for XML Document Retrieval

Kai Großjohann Norbert Fuhr Daniel Effing Sascha Kriewel
University of Dortmund, Germany
{grossjohann,fuhr}@ls6.cs.uni-dortmund.de

Abstract: XML document retrieval requires new ideas for user interface design. The query language provides primitives for dealing with the tree structure, which needs to be reflected in an interface for query formulation. Further, the XML structure is also reflected in the retrieval results, where items may contain each other. In this paper, we present a user interface for formulating queries in an XPath-like language and an interface for presenting retrieval results.

1 Introduction

XML document retrieval poses new challenges for user interface design, arising from the complex structure of XML query languages and the documents themselves. The classical task in Information Retrieval is the selection of documents from a given collection. For XML, this should be extended such that parts of documents (XML elements) can be selected, as well. We chose XPath as the starting point for developing XIRQL [FG01], our own XML query language. While it is possible to select elements from documents with XPath, some things are missing from the Information Retrieval point of view:

Weighting and Ranking: The result of any query should be a ranked list. So, query results should always be weighted sets and all operators should be (re)defined to take these weights into account.

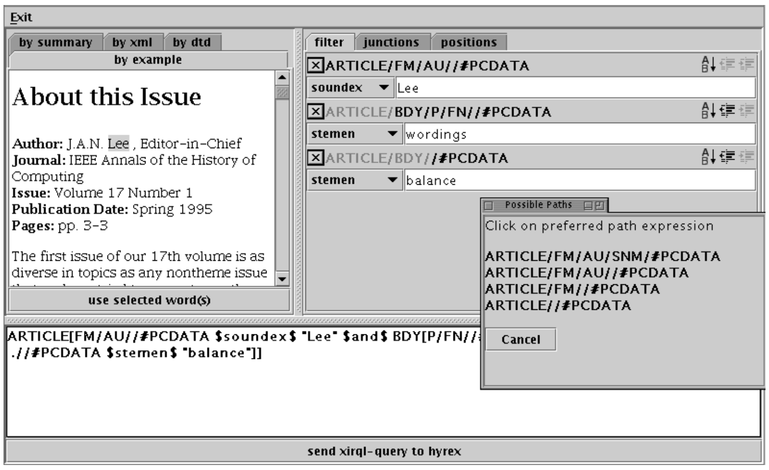
Data Types and Vague Predicates: XPath provides string and numeric predicates, but this does not reflect the semantic richness of the underlying data. For prose, users wish to search for ground forms, person names lead to search for phonetic similarity, and so on. The query language should provide appropriate search predicates for the different data types.

Compared with unstructured document retrieval, retrieval in XML documents results in increased complexity in two areas: first of all, XPath is a complex query language due to the conditions on the XML structure of the documents. Secondly, the query results exhibit internal structure, as one retrieval result might be the ancestor of another retrieval result. Section 2 presents a mechanism for constructing XIRQL queries without needing to know the syntax of the query language. In section 3, we deal with the presentation of retrieval results to the user.

2 Query Formulation

We aim at an interface for formulating XIRQL queries which does not require knowledge of the query syntax [Eff02]. The interface should be independent of the application, it should be applicable to any kind of documents.

Element names, the dot `.`, and the operators `/`, `//` and `[]` work in XIRQL as they do in XPath. Additionally, XIRQL supports various comparison operators for different data types. For example, where XPath allows constructions like `//element="string"` to search for a certain string, XIRQL offers `//author sounds_like "name"` for person names and `//para contains "word"` for English prose.



About this Issue

Author: J.A.N. Lee , Editor-in-Chief
Journal: IEEE Annals of the History of Computing
Issue: Volume 17 Number 1
Publication Date: Spring 1995
Pages: pp. 3-3

The first issue of our 17th volume is as diverse in topics as any nontheme issue

ARTICLE/FM/AU//#PCDATA
soundex: Lee
ARTICLE/BDY/P/FN//#PCDATA
stemen: wordings
ARTICLE/BDY//#PCDATA
stemen: balance

Click on preferred path expression

ARTICLE/FM/AU/SNM/#PCDATA
ARTICLE/FM/AU//#PCDATA
ARTICLE/FM//#PCDATA
ARTICLE//#PCDATA

```
ARTICLE[FM/AU//#PCDATA $soundex$ "Lee" $and$ BDY[P/FN//
.//#PCDATA $stemen$ "balance"]]
```

Figure 1: Interface for formulating queries.

A screenshot of our interface can be seen in figure 1. There are three areas: On the left, the *structure condition area* enables users to formulate single query conditions. On the right, the *condition list area* allows users to edit the query conditions and to specify how to combine them to form the whole query. At all times, a paraphrase of the current query in XIRQL syntax is kept up to date in the *paraphrase area* on the bottom.

For formulating a single query condition, the main mechanism is to use *Query by Example*. In the screenshot, the layout-oriented variant is shown. The user can click on a word in that document and the system derives from it a *structural condition* (candidate) and a *value condition* (candidate). The structural condition describes the list of element names on the path from the root node to the leaf node in the XML tree. From it, a number of generalizations (using the `//` operator and the `*` wild card) are produced and shown to the user (see the popup window in the middle of the screenshot). After selecting the structural condition, the query condition is added to the condition list area, where additional changes can be made: The comparison value (defaulting to the word the user selected) can be edited, and a search predicate can be chosen for this condition.

In addition to the layout-oriented variant of Query by Example, we offer a structure-oriented variant where people see an expandable tree of the XML document, as well as a structure-orient variant which shows a document surrogate only. Finally, as an alternative to Query by Example, we offer a *DTD oriented* method for specifying the structure condition which does not rely on an example document.

The next step is to specify how the query conditions thus collected should be combined to form the whole query. Here, we focus on the *structural dependence* between the conditions. This is achieved by specifying a common prefix for two query conditions. For example, in the third condition, `/ARTICLE/BDY` is grayed out. This means the match for the second and third conditions must be in the same `BDY` element (and hence within the same `ARTICLE` element). The graying-out connects two adjacent conditions, so it is possible to move conditions up and down in the list. In addition to the structural dependence, the *Boolean connectors* between the conditions also need to be specified. We do this in a simple manner, allowing the user to choose between `and` and `or` between any two conditions, but we plan more elaborate support, possibly based on Venn diagrams.

To test the usefulness of this approach, we performed a small preliminary user study, summarized in table 1. Three retrieval tasks (against the content of the IEEE Computer Society CD-ROMs from 1995 to 1997[1]) were given in natural language. Five users performed the tasks with the graphical interface described here, two also used a command-line tool to directly enter XIRQL queries. It seems that even people with no knowledge of XIRQL are enabled to pose queries using this interface. For more complex queries, the interface might speed up users who know XIRQL. The layout-oriented variant of Query by Example was popular with all users, the DTD-based method was rarely used.

Task	visual interface					cmd line	
	1	2	3	4	5	4	5
articles by Tomayko (family name)							
	3:10	7:00	3:54	0:55	0:30	0:56	0:33
by William (given name) mentioning the ENIAC computer							
	4:56	5:15	2:43	1:15	2:10	1:17	1:08
1995 articles by Helen M. Wood							
	4:12	3:50	2:15	0:40	1:15	1:45	1:59

Table 1: User study for formulating queries, times given in minutes and seconds. Users 1 to 3 had no knowledge of XIRQL, users 4 and 5 were XIRQL experts.

3 Result Presentation

The objective of traditional document retrieval systems is to select documents from a collection. For XML documents, the obvious extension is to select parts (elements) of documents, too. Items in such retrieval results may contain each other, so it is important to show the relationship in the tree between them. Since results are expected to come from several documents, the representation should be compact so that more than one document can be displayed at the same time. Treemaps [JS91] provide such a representation (see figure 2 for an example).

But for trees with many nodes, the representation is too cluttered. Therefore, we augment the concept and introduce *Partial Treemaps*, where we omit nodes if they are not a retrieved

[1] `http://www.computer.org/cspress/catalog/cs-96.htm`

item or an ancestor of a retrieved item [Kri01].

Tool-tips provide additional information about each retrieved item. In addition to a list of Partial Treemaps, the document itself is shown (processed by an XSL style-sheet) together with a 'table of contents' view. The table of contents is a tree view of the document, but certain 'unimportant' XML elements are left out to constrict the size of the tree. The elements to retain are those that contribute to the overall logical structure of the document. (Typically, the `section` element would be included, but the `bold` element would not be included.)

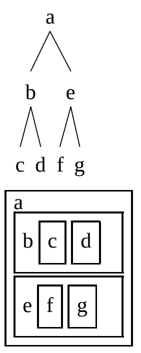


Figure 2: A simple tree and its treemap.

With XIRQL, a retrieval result is always a weighted set. So the visualization described in the previous paragraphs needs to be extended to deal with the weights, too. For the weights, a visual variable is needed that can be used together with Partial Treemaps. Since the weights impose a linear order on the results, the visual variable should be selective (allowing distinction between objects with and without a certain property) as well as ordered (allowing a less-than comparison between values). We choose brightness as the visual variable to use. This is implemented via shades of gray, where white means zero (the object is not relevant at all) and black means one (the object is highly relevant). The result is a generalization of the concept of *TileBars* [Hea95, Hea99] from linear text to tree structures. Our interface is shown in figure 3.

	text	Partial Treemaps
time	2:20	2:05
recall	0.51	0.75
precision	0.40	0.66

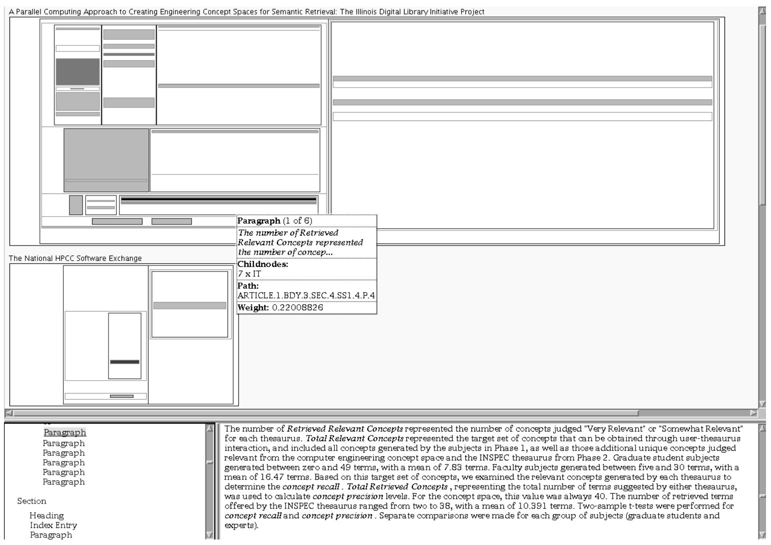
Table 2: User study on visualization of retrieval results. Times are given in minutes and seconds. All values are averages over the five users and six tasks. *Recall* means, how many of the relevant results were also judged as relevant; whereas *precision* means, how many of the results that were judged as relevant were actually relevant.

We performed a small user study to test the effect of the visualization on the time the users needed, and on the quality of the relevance judgments. Five users were given six queries each, together with a visualization of the query results. Each user chose three queries (query results) for judgment with a textual result representation, and three queries for judgment with a Partial Treemap visualization. Our findings on time and quality are shown in table 2.

The small difference in time between the the two methods is puzzling. Participants reported that they had a closer look at the retrieval results and their relationships when using the graphical method; this could be an explanation. The added information provided by the graphical method clearly improved the quality of the judgments.

4 Conclusion

Retrieval from XML documents brings up the issue of dealing with the XML tree structure both when formulating queries and when looking at the retrieval results. We have developed a visualization for both tasks. Our solution for the query formulation is application-independent and currently only suitable for experienced users. Support for naive users also


A Parallel Computing Approach to Creating Engineering Concept Spaces for Semantic Retrieval: The Illinois Digital Library Initiative Project

The National HPCC Software Exchange

Paragraph (1 of 6)
The number of Retrieved Relevant Concepts represented the number of concep...
Childnodes: 7 x IT
Path: ARTICLE.1.BDY.3.SEC.4.SS1.4.P.4
Weight: 0.22008826

Paragraph
Paragraph
Paragraph
Paragraph
Paragraph
Paragraph

Section
Heading
Index Entry
Paragraph

The number of *Retrieved Relevant Concepts* represented the number of concepts judged "Very Relevant" or "Somewhat Relevant" for each thesaurus. *Total Relevant Concepts* represented the target set of concepts that can be obtained through user-thesaurus interaction, and included all concepts generated by the subjects in Phase 1, as well as those additional unique concepts judged relevant from the computer engineering concept space and the INSPEC thesaurus from Phase 2. Graduate student subjects generated between zero and 49 terms, with a mean of 7.83 terms. Faculty subjects generated between five and 30 terms, with a mean of 16.47 terms. Based on this target set of concepts, we examined the relevant concepts generated by each thesaurus to determine the *concept recall* . *Total Retrieved Concepts* , representing the total number of terms suggested by either thesaurus, was used to calculate *concept precision* levels. For the concept space, this value was always 40. The number of retrieved terms offered by the INSPEC thesaurus ranged from two to 38, with a mean of 10.391 terms. Two-sample t-tests were performed for *concept recall* and *concept precision* . Separate comparisons were made for each group of subjects (graduate students and experts).

Figure 3: Result presentation with Partial Treemaps. Each element in the treemap has a tool-tip with a summary about that element. In the bottom left, we show a 'table of contents' (tree showing certain elements) and in the bottom right, we show the document itself, at the spot corresponding to the element in the treemap that the user has clicked on.

implies making the interface application-dependent; this is subject of further research. We are also working on supporting browsing and navigation in addition to formulation of queries.

Literature

[Eff02] Daniel Effing. Unterstützung von Nutzern bei der Erstellung von XIRQL-Anfragen. Diploma thesis, Universität Dortmund, FB Informatik, January 2002.

[FG01] N. Fuhr and K. Großjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In W.B. Croft, D. Harper, D.H. Kraft, and J. Zobel, editors, *Proceedings of the 24th Annual International Conference on Research and development in Information Retrieval*, pages 172–180, New York, 2001. ACM.

[Hea95] Marti A. Hearst. TileBars: Visualization of Term Distribution Information in Full Text Information Access. In *Proceedings of CHI*, May 1995.

[Hea99] Marti A. Hearst. User Interfaces and Visualization. *Modern Information Retrieval*, 1999.

[JS91] Brian Johnson and Ben Shneiderman. Tree-Maps: A Space Filling Approach to the Visualization of Hierarchical Information Structures. Technical Report CS-TR-2657, University of Maryland, Computer Science Department, April 1991.

[Kri01] Sascha Kriewel. Visualisierung für Retrieval von XML-Dokumenten. Diploma thesis, Universität Dortmund, Fachbereich Informatik, Dezember 2001.