Query Formulation and Result Visualization for XML Retrieval

Kai Großjohann  Norbert Fuhr  Daniel Effing  Sascha Kriewel
University of Dortmund, Germany
{grossjohann,fuhr}@1s6.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.

Figure 1: Interface for formulating queries.

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 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.

A screenshot of our interface can be seen in figure 4. 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-oriented 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. This method allows for formulating all structure conditions which are meaningful according to the DTD.

The next step is to specify how the query conditions thus collected should be combined to form the whole query. We provide a simple way for users to specify the Boolean connectors between query conditions, but we focus on the structural dependence between the

<table>
<thead>
<tr>
<th>Task</th>
<th>visual interface</th>
<th>cmd line</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>articles by Tomayko (family name)</td>
<td>3:10</td>
<td>7:00</td>
</tr>
<tr>
<td>by William (given name) mentioning the ENIAC computer</td>
<td>4:56</td>
<td>5:15</td>
</tr>
</tbody>
</table>

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.
conditions. This is achieved by first reordering the query conditions and then specifying a common prefix for two adjacent query conditions. For example, in the third condition, 
\texttt{/ARTICLE/BDY} is grayed out. This means the match for the second and third conditions must be in the same \texttt{BDY} element (and hence within the same \texttt{ARTICLE} element).

To test the usefulness of this approach, we performed a small preliminary user study, summarized in table

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

Tooltips provide additional information about each retrieved item. In addition to a list of Partial Treemaps, the document itself is shown (processed by an XSL stylesheet) together with its logical structure. The latter is a tree view of the document, where elements have been left out so that only those are left that contribute to the overall logical structure.

We use shades of grey for the rectangles to show the retrieval weight because brightness is a selective as well as ordered visual variable. Also, it can be used in conjunction with Partial Treemaps. Our interface is shown in figure

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. In addition to a Google-style textual representation of the query results, the two graphical visualizations TextBars and Partial Treemaps were provided. TextBars are based on TileBars as described in [Hea95, Hea99]. We segment the bars according to the XML structure. The start of an XML element is indicated with a red bar, and the shade of gray indicates the score.

As for Partial Treemaps, only those elements present in the result set are shown. We use this representation to investigate the effect of also showing the tree structure instead of only a linear one.

<table>
<thead>
<tr>
<th></th>
<th>text</th>
<th>TextBars</th>
<th>Partial Treemaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>2:20</td>
<td>2:04</td>
<td>2:05</td>
</tr>
<tr>
<td>recall</td>
<td>0.51</td>
<td>0.76</td>
<td>0.75</td>
</tr>
<tr>
<td>precision</td>
<td>0.40</td>
<td>0.52</td>
<td>0.66</td>
</tr>
</tbody>
</table>

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 nine 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.
Five users were given nine 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 each for judgment with a TextBars and a Partial Treemap visualization. Our findings on time and quality are shown in table Figure 3.

Figure 3: Result presentation with Partial Treemaps. Each element in the treemap has a tooltip with a summary about that element. In the bottom left, we present the logical structure of the document and in the bottom right, the document itself is shown, at the spot corresponding to the element in the treemap that the user has clicked on.

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 methods clearly improved the quality of the judgments, and the structural information from the Partial Treemaps helped to weed out false positives.

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 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.

Literaturverzeichnis


