Generating Vector Spaces On-the-fly for Flexible XML Retrieval

Torsten Grabs Hans-Jörg Schek
Database Research Group, Institute of Information Systems
ETH Zentrum, 8092 Zurich, Switzerland
{grabs,schek}@inf.ethz.ch

1 Introduction

While documents are flat with conventional information retrieval, i.e., they are unstructured information, this is no longer adequate with semistructured data such as XML for two reasons.

First, XML allows to hierarchically structure information within a document such that each document has tree structure. Users in turn want to refer to this structure when searching for relevant information. To do so, users pose so-called *content-and-structure queries*. Such queries refer to the document structure, e.g., by restricting the context of interest to some XML elements. Relevance ranking consequently has to properly reflect both document structure and the constraints that the query poses on the structure. Namely, the contents at the different levels of the tree are considered of different importance for a query. The intuition behind this is that content that is more distant in the document tree is less important than the one that is close to the context node. We subsequently denote this concept as *nested retrieval*, and it is a crucial requirement for meaningful retrieval from XML documents. Fuhr et al. tackle this issue by a technique denoted as *augmentation* [2, 3]. The idea is to introduce so-called *augmentation weights* that downweigh statistics such as inverted document frequencies of terms when the terms are propagated upwards in the document tree. To do so, Fuhr et al. [3] group XML element types to so-called *indexing nodes* that implement the inverted lists for efficient retrieval. They constitute the granularity of retrieval with their approach, i.e., indexes and statistics such as document frequencies are derived separately per indexing node. Users can search at the granularity of the indexing nodes and hierarchical combinations of them if indexing nodes are along the same path in the document. Term weights are properly augmented in this case. The drawback of the approach is that the assignment of XML element types to indexing nodes is static. Hence, users cannot retrieve dynamically, i.e., at query time, from arbitrary combinations of element types.

The second reason why conventional retrieval techniques do not suffice for XML retrieval is that even a single XML document may have very heterogeneous content. Take for instance

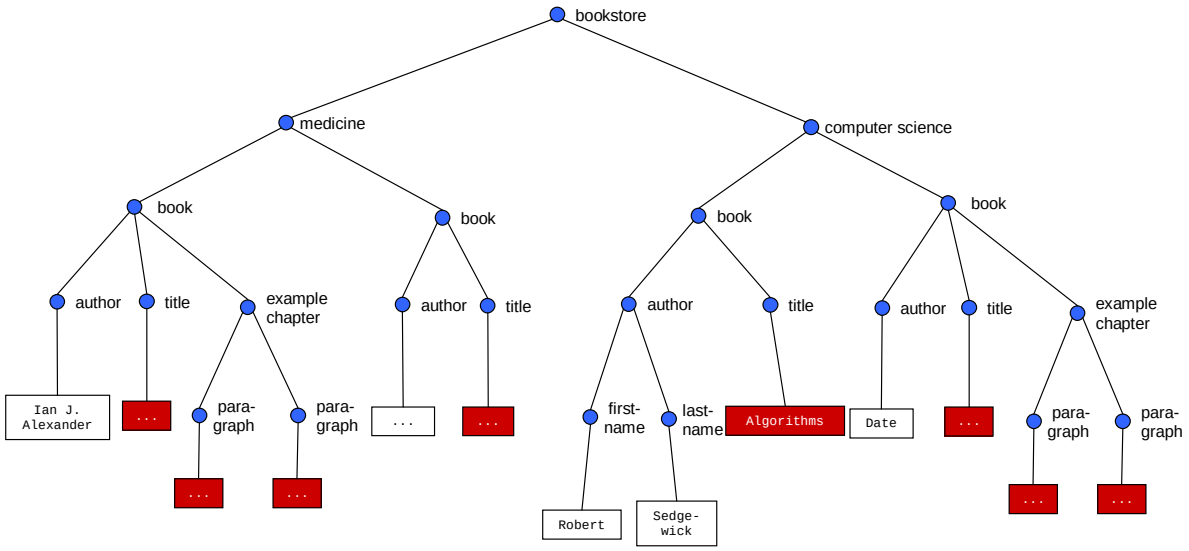

Figure 1: Example XML document with textual content represented as shaded boxes

an XML document that contains information for tourists. One part of the document may be devoted to restaurants while another part describes the ultimate places for sightseeing. When posing their queries, users may want to refer to such a part, or *category*, in isolation, i.e., only content from one category is subject to the query. We then speak of so-called *single-category retrieval*. With other queries in turn, users may want to process the query on several such categories, or they do not care at all to which category the content belongs. We call these queries *multi-category retrieval*. The difficulty with this type of queries is that common statistics for information retrieval such as document frequencies must properly reflect the scope, i.e., the granularity, of the query. Otherwise, query processing may lead to inconsistent rankings. Hence, a further important requirement for XML retrieval is that users can dynamically, i.e., at query time, define the granularity of retrieval. This is in contrast to conventional retrieval where the retrieval granularity always is the complete document or a single field such as 'abstract' or 'title'. The following example further illustrates the shortcomings of conventional retrieval techniques in the context of XML.

> **Example 1:** An online store for scientific books keeps its information about books as an XML document as the one shown in Figure 1. The document reflects the different categories of books such as 'medicine' or 'computer science' with separate element types for the respective categories. Further, the document stores the usual information about books such as 'author', 'title', 'summary', and 'price' (not all of them are shown in the figure). Note that it also comprises example chapters for some selected books.[1] The document also comprises content that the users want to access using information retrieval techniques: namely, the example chapters and the title fields shown in the figure are of interest.
>
> Consider now a user searching for relevant books in the 'computer science' category. Obviously, he restricts his queries to books from this particular category.

[1]This is for instance the case with the Web site of Microsoft Press (http://mspress.microsoft.com).

Thus, it is not appropriate to process this query with term weights derived from both the categories 'medicine' and 'computer science' in combination. This is because the document frequencies in 'medicine' may skew the term weights such that the ranking for a query on 'computer science' in isolation is inconsistent. Consequently, term weights for this query must only be based on the content of the 'computer science' category. Hence, we speak of *single-category retrieval*.

Now, think of another user who does not care to which category a book belongs, as long as it covers the information need expressed in his query. The granularity of his query are all categories. The query is an example of *multi-category retrieval* which requires that the statistics properly reflect the scope of the query.

Recall that both these queries search for relevant 'book' elements. Hence, they have to process content that is hierarchically structured: both the 'title' elements and the 'paragraph' elements of the example chapters describe a particular 'book' element. Intuitively, content that occurs in the title element is deemed more important than that in the paragraphs of the example chapter. Consequently, both query types also require the functionality for *nested retrieval* and proper *augmentation* as discussed by Fuhr et al. [3]. ◇

Our aim is to provide efficient and consistent *retrieval over arbitrary combinations and nestings of element types* – or briefly *flexible retrieval* from XML documents. This is challenging since common IR statistics such as element frequencies (as opposed to document frequencies with conventional IR) are different depending on the scope of the query. A naive implementation would be to keep indexes and statistics for each combination of element types and element nestings that could possibly occur in a query. Using the terminology from Fuhr et al. [3], each such combination would lead to a separate indexing node. However, the amount of storage that this approach requires for indexes and statistics is prohibitively large. Hence, it is not a viable solution. This paper in turn proposes to keep indexes and statistics only for basic element types. When it comes to multi-category retrieval or nested retrieval, it derives the required indexes and statistics from the underlying basic ones on-the-fly, i.e., at query runtime. This has the advantage that the amount of storage that is needed to efficiently process IR queries on XML content is small as compared to the naive approach. Moreover, results from our previous work on retrieval from different categories of flat documents indicate that the overhead on query response times with on-the-fly integration is small: it is less than 30% with up to 16 categories as compared to a setting with pre-computed statistics [5].

In this paper, we focus on vector space retrieval and the usual TFIDF ranking. Our contribution is to dynamically derive the vector space that is appropriate for the scope of the query from underlying basic vector spaces. In order to do so, we generalize previous work on augmentation [3] and multi-category retrieval from flat documents [5] to flexible information retrieval on XML. We discuss the semantics of the different query types under the vector space retrieval model and explain the algorithms required to implement efficient query processing.

The remainder of the paper is as follows: Section 2 explains the semantics of generating vector spaces on-the-fly depending on the scope of the query. In Section 3, we propose the algorithms that integrate indexes and statistics on-the-fly to the vector space needed to process a given query. Section 4 discusses related work, and Section 5 concludes.

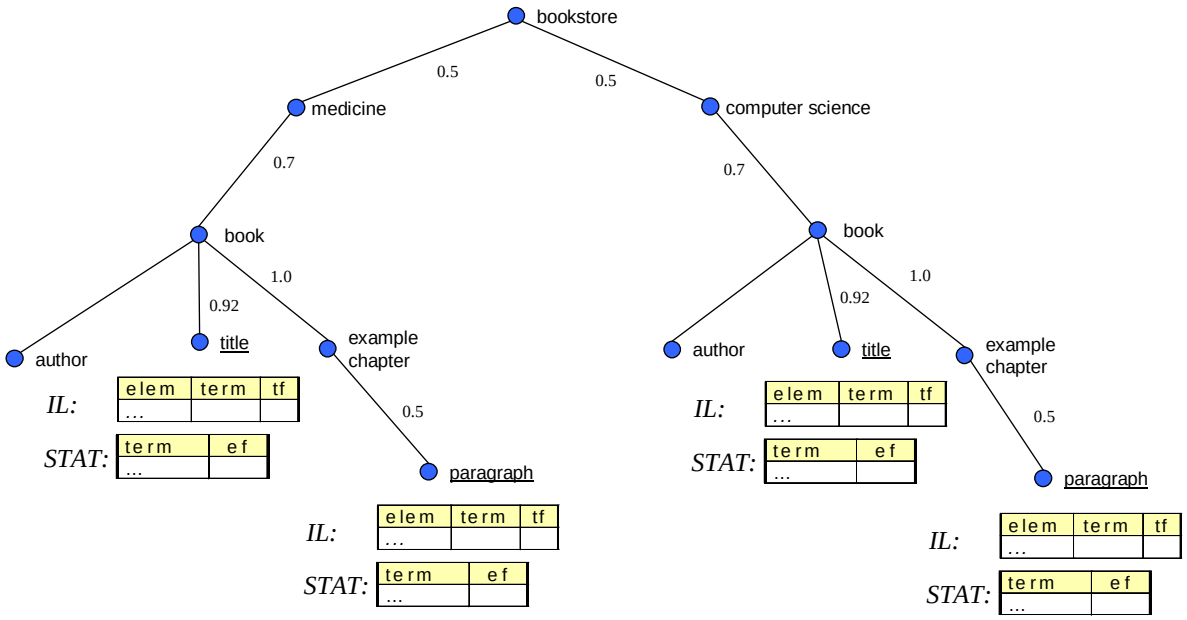

Figure 2: Basic indexing nodes for the example document from Fig. 1

2 Flexible Information Retrieval from XML Documents

As motivated in the previous section, different retrieval granularities are crucial for effective retrieval from an XML collection. To address this requirement, our approach first requires to identify the basic element types of an XML collection that contain textual content. We denote them as *basic indexing nodes*. There are several alternatives how to derive the basic indexing nodes from an XML collection: (1) The decision can be taken completely automatically such that each distinct element type with textual content is treated as a separate indexing node. (2) An alternative is that the user or an administrator decides how to assign element types to basic indexing nodes. Both these approaches can further rely on an ontology that, for instance, suggests to group element types 'title' and 'abstract' into the same basic indexing node. For the remainder of this paper, we assume that the basic indexing nodes have already been determined. This includes that indexes and statistics for vector space retrieval on the basic indexing nodes have already been generated using standard IR techniques such as term extraction, stemming, and stopword elimination.

> **Example 2:** Consider again the online bookstore from Example 1. Figure 2 illustrates the basic indexing nodes for the bookstore example. For the sake of illustration, the figure depicts the element types of the original document in a structure similar to a DataGuide [4]. Element types with textual content are underlined. Each such element type constitutes a basic indexing node and has been annotated by an inverted list (*IL*) and the vector space statistics (*STAT*). In addition, edges are labeled with the augmentation weights required for nested retrieval. We postpone their explanation to our discussion of nested retrieval. ◇

Taking the basic indexing nodes as a starting point, the following paragraphs explain how to dynamically derive the appropriate indexes and statistics for queries with arbitrary scope.

We start with a brief review of *single-category retrieval*. Then, we explain how to dynamically derive the indexing node data for *multi-category retrieval* and *nested retrieval* from the underlying basic indexing nodes.

Single-Category Retrieval. Single-category retrieval with XML works on the element type of a basic indexing node. For example, the path expression '/bookstore/medicine/book/title' defines a single category with Figure 2. The granularity of retrieval are all elements of that element type. For the moment, let us assume for ease of presentation that categories represent those leaf nodes of the XML documents that are basic indexing nodes. We will shortly give up this restriction.

In order to allow for relevance ranking, we take over the usual definition of retrieval status value with the vector space retrieval model: As usual, t denotes a term, and $tf(t,e)$ is its term frequency with an element e. Let N_{cat} and $ef_{cat}(t)$ denote the number of elements at the single category *cat* and the element frequency of term t with the elements of *cat*, respectively. In analogy to the inverted document frequency for conventional vector space retrieval, we define *inverted element frequency (ief)* as $ief_{cat}(t) = \log \frac{N_{cat}}{ef_{cat}(t)}$. The retrieval status value of an element e for a single-category query q is then

$$RSV(e,q) = \sum_{t \in q} tf(t,e)\ ief_{cat}(t)^2\ tf(t,q) \tag{1}$$

Consider now the case of single-category retrieval on an inner node of the XML structure that is not a basic indexing node. In this case, our approach derives the indexing node data of the inner node from the underlying basic indexing nodes by applying augmentation to the term weights, as defined by Fuhr et al. [2, 3].

Multi-Category Retrieval. In contrast to single-category retrieval, *multi-category retrieval* with XML works with *multi-categories*. For example, the path expression '/bookstore/medicine/book/title|/bookstore/computerscience/book/title' defines a multi-category query (cf. Figure 2). The granularity of retrieval with a multi-category are all elements that match this path expression. Formally, a multi-category is given by a path expression that may contain choices. When it comes to retrieval from a multi-category, statistics such as element frequencies for vector space retrieval and especially the *rsv* must reflect this. Otherwise, inconsistent rankings are possible. Our approach to guarantee consistent retrieval results is similar to integrating statistics for queries over different document categories with conventional retrieval [5]. We extend this notion now such that statistics for multi-category retrieval depend on the statistics of each single-category that occurs in the query. As the subsequent definitions show, our approach first computes the statistics for each single-category as defined in Definition 1 and then integrates them to the multi-category ones as follows. $\mathcal{M}$ denotes the set of single categories of the multi-category. $N_{mcat} = \sum_{cat \in \mathcal{M}} N_{cat}$ stands for the number of elements of the multi-category. Thus, $ief_{mcat}(t) = \log \frac{N_{mcat}}{\sum_{cat \in \mathcal{M}} ef_{cat}(t)}$, where $ef_{cat}(t)$ denotes the single-category element frequency of term t with category *cat*. The retrieval status value of an element e for a multi-category query q is then using again TFIDF ranking:

$$RSV(e,q) = \sum_{t \in q} tf(t,e)\ ief_{mcat}(t)^2\ tf(t,q) \tag{2}$$

This definition integrates the frequencies of several single categories to a consistent global one. It equals Definition 1 in the trivial case with only one category in the multi-category.

Nested Retrieval. Another type of requests are those that operate on complete subtress of the XML documents. For instance, the path expression '/bookstore/medicine/book/example-chapter/' defines such a subtree for the XML document in Figure 1. However, there are the three following difficulties with this retrieval type:

- A path expression such as the one given above comprises different categories in its XML subtree. With the element types from Figure 2 for instance, these are 'title', and 'paragraph'. Hence, retrieval over the complete subtree must consider these element types in combination to provide a consistent ranking.
- Terms that occur close to the root of the subtree typically are considered more significant for the root element than ones on deeper levels of the subtree. Intuitively: the larger the distance of a node from its ancestor is, the less it contributes to the relevance of its ancestor. Fuhr et al. [2, 3] tackle this issue by so-called *augmentation weights* which down-weigh term weights when they are pushed upward in hierarchically structured documents such as XML. Thus, relevance ranking for nested retrieval must include augmentation.
- Element containment is at the instance level, and not at the type level. Consequently, element containment relations cannot be derived completely from the element type nesting.

To define relevance ranking for nested retrieval, let e denote an element that qualifies for the path expression of the nested-retrieval query. Let $SE(e)$ denote the set of sub-elements of e including e, i.e., all elements contained by the sub-tree rooted by e. For each $se \in SE(e)$, $l \in path(e,se)$ stands for a label along the path from e to se, and $aw_l \in [0.0;1.0]$ is its augmentation weight as defined by the annotations of the edges in the element type structure (cf. Figure 2). $cat(se)$ denotes the category to which se belongs. The category of an inner node is dynamically generated by applying augmentation to the underlying basic indexing nodes. $ief_{cat(se)}(t)$ stands for the inverted element frequency of term t with the category $cat(se)$. The retrieval status value rsv of an element e under a nested-retrieval query q using the vector space retrieval model is then:

$$RSV(e,q) = \sum_{se \in SE(e)} \sum_{t \in q} tf(t,se) \Big(\prod_{l \in path(e,se)} aw_l \Big)^2 ief_{cat(se)}(t)^2\ tf(t,q) \tag{3}$$

$$= \sum_{se \in SE(e)} \Big(\Big(\prod_{l \in path(e,se)} aw_l \Big)^2 \sum_{t \in q} tf(t,se)\ ief_{cat(se)}(t)^2\ tf(t,q) \Big) \tag{4}$$

As Definition 4 shows, nested retrieval is a weighted sum of constrained single category retrieval results. The constraint is such that an element se and its textual content only contribute to the retrieval status value of e if se is in the sub-tree rooted by e. Moreover, Definition 3 and 4 revert to the common TFIDF ranking for conventional retrieval on flat documents when all augmentation weights are equal to 1.0. In the trivial case where a nested query only comprises one single-category, Definition 3 and 4 equal Definition 1.

```
Algorithm MULTICATEGORY
Parameters: Query q, path expression p
var hits := ∅; M := ∅;
begin
    // Step 1: Determine the single-categories and their basic indexing nodes for p
    M = LookUp(p)

    // Step 2: Collect and integrate statistics
    for each single-category cat ∈ M do in parallel
        Get per-category statistics (ef_cat(t), N_cat); end;
    Compute multi-category statistics stat_mcat (ief_mcat and N_mcat for Def. 2);

    // Step 3: Execute query for each category
    for each category cat ∈ M do in parallel
        // process the query in combination with the integrated statistics
        hits := hits ∪ Query_mcat(cat, q, stat_cat); end;

    // Step 4: Post-processing and output of results
    Sort hits by RSV; Return the ranking (element id and RSV);
end;
```

Figure 3: Algorithm **MULTICATEGORY** for multi-category query processing

3 Implementing Flexible Retrieval from XML Documents

In the following paragraphs, we explain how to implement multi-category retrieval and nested retrieval using the data of the basic indexing nodes.

Multi-Category Retrieval. Using the statistics of the basic indexing nodes directly for multi-category retrieval is not feasible since statistics are per element type. Hence, query processing must dynamically integrate the statistics if the query encompasses several categories. Using single-category statistics directly may lead to wrong rankings with multi-category queries. Multi-category queries compute the correct multi-category statistics during query processing. Algorithm **MULTICATEGORY** shown in Figure 3 reflects this. First, it determines the basic indexing nodes contained in the path expression of the multi-category query. Its second step is to retrieve the statistics for each such basic indexing node and to use them to compute the integrated ones. The third step executes the lookup in parallel at the inverted lists. The inverted list lookup takes the integrated multi-category statistics as input parameter and computes the partial ranking. The fourth step of **MULTICATEGORY** integrates the partial results from the third step and returns the overall ranking.

Nested Retrieval. As with the previous retrieval type, nested retrieval requires integrating statistics and processing queries over different indexes. In addition, it must also reflect element

Algorithm NESTEDRETRIEVAL
Parameters: Query q, path expression p
var hits := $\emptyset$; $\mathcal{N}$:= $\emptyset$;
begin
// Step 1: Determine the single-categories and their basic indexing nodes from p
$\mathcal{N} = LookUp(p)$

// Step 2: Compute the integrated statistics with augmented weights
// $\mathcal{W}(STAT_{cat}, \prod_{l \in path(base(p),cat)} aw_l)$ *denotes the weighted projection of the per-category statistics*
// $base(p)$ *denotes the element type of the query root*
for each category $cat \in \mathcal{N}$ **do in parallel**
$STAT_{temp} := STAT_{temp} \cup \mathcal{W}(STAT_{cat}, \prod_{l \in path(base(p),cat)} aw_l)$ **end**;

// Step 3: Process the query on each category with the augmented statistics
for each category $cat \in \mathcal{N}$ **do in parallel**
hits := hits $\cup$ Query$_{ncat}$(q, $STAT_{temp}$); **end**;

// Step 4: Post-processing and output of results
Sort hits by RSV; Return the ranking (element id and RSV);
end;

end;

Figure 4: Algorithm **NESTEDRETRIEVAL** for nested-retrieval processing

containment and augmentation weights properly. This makes processing of this query type more complex than with the other types.

Our algorithm to process nested queries is called **NESTEDRETRIEVAL**, and it comprises four steps, as shown in Figure 4. The first step computes the categories that qualify for the path expression defining the scope of the nested query. The second step then iterates over the categories, their underlying basic indexing nodes, and dynamically generates the statistics for the appropriate vector space of the scope of the query.

Note that the dynamically generated statistics $STAT_{temp}$ comprise different inverted element frequencies (*ief*) for the same term depending on the category where the term occurs and the weight of the category. The weighting function $\mathcal{W}$ augments each term $t \in q$ from the statistics $STAT_{cat}$ with its proper augmentation weights regarding the context node of the query. This ensures that the properly augmented *ief*s are used to compute the *rsv*. The last step of the algorithm then computes the overall ranking.

4 Related Work

As a first measure to enhance functionality for document centric processing of XML, Florescu et al. realize searching for keywords in textual content of XML elements [1]. However, the mere capability to search for keywords does not suffice to address the requirements for doc-

ument centric processing: support for state-of-the-art retrieval models with relevance ranking is needed. To tackle this issue, Theobald et al. propose the query language XXL and its implementation with the XXL Search Engine [6]. Regarding statistics such as inverted document frequencies (*idf*) their approach treats XML documents as flat structures. Fuhr et al. have already argued in [2, 3] that this comes too short for semi-structured data such as XML. They propose to downweigh terms by so-called augmentation weights when the terms are propagated upwards in the document hierarchy. However, their approach groups element types to so-called indexing nodes which constitute the basis for statistics such as *idf*. Hence, consistent retrieval is only feasible at the granularity of indexing nodes and hierarchical combinations of them.

Our approach takes over these ideas and generalizes them such that consistent retrieval with arbitrary query granularities, i.e., arbitrary combinations of element types, is feasible. This makes the restriction of retrieval granularity to indexing nodes obsolete and allows for flexible retrieval from XML collections.

5 Conclusions

Flexible retrieval is crucial for document centric processing of XML. Flexible retrieval means that users may dynamically, i.e., at query time, define the scope of their queries. So far, consistent retrieval on XML collections has only been feasible at fixed granularities [3, 6]. The difficulty is to treat statistics such as document frequencies properly in the context of hierarchically structured data with possibly heterogeneous contents. Our approach in turn allows for flexible retrieval over arbitrary combinations of element types. In this paper, we propose *single-category retrieval*, *multi-category retrieval*, and *nested retrieval* for flexible retrieval from XML documents. To tackle the aforementioned difficulty, we rely on basic index and statistics data and integrate them on-the-fly, i.e., during query processing, into a consistent view that properly reflects the scope of the query. Taking the vector space retrieval model for instance, our approach dynamically generates the appropriate vector space for each query from the underlying basic ones. A nice characteristic of our approach is that it covers conventional retrieval on flat documents as a special case.

An extensive experimental evaluation has already investigated dynamically integrating statistics for different categories of flat documents [5]. Our findings from this previous work show that the overhead of dynamic statistics integration is less than 30% with up to 16 different categories, compared to a setting with pre-computed statistics. We are currently extending the implementation of our XML repository with the functionality for flexible retrieval on XML documents and the algorithms for multi-category and nested retrieval as outlined in this paper. Our expectation is that the overhead for generating appropriate vector spaces on-the-fly is also reasonable for retrieval on XML documents such that we can guarantee interactive retrieval response times.

References

[1] D. Florescu, D. Kossmann, and I. Manolescu. Integrating Keyword Search into XML Query Processing. In *Proc. of the Intern. WWW Conference, Amsterdam, May 2000*. Elsevier, 2000.

[2] N. Fuhr, N. Gövert, and T. Rölleke. Dolores: A system for logic-based retrieval of multimedia objects. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1998, Melbourne, Australia*, pages 257–265. ACM Press, 1998.

[3] N. Fuhr and K. Großjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *Proceedings of the 24th Annual ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 172–180. ACM Press, 2001.

[4] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proceedings of 23rd International Conference on Very Large Data Bases, 1997, Athens, Greece*, pages 436–445. Morgan Kaufmann, 1997.

[5] T. Grabs, K. Böhm, and H.-J. Schek. PowerDB-IR – Information Retrieval on Top of a Database Cluster. In *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM2001), November 5-10, 2001 Atlanta, GA, USA*. ACM Press, 2001.

[6] A. Theobald and G. Weikum. The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking. In *Advances in Database Technology - EDBT 2002, 8th International Conference on Extending Database Technology, Prague, Czech Republic*, volume 2287 of *Lecture Notes in Computer Science*, pages 477–495. Springer, 2002.