International Journal on Digital Libraries manuscript No.
(will be inserted by the editor)

Visual Exploration and Retrieval of XML Document Collections with the Generic System X^2

Holger Meuss[2], **Klaus U. Schulz**[1], **Felix Weigel**[1], **Simone Leonardi**[1], **François Bry**[3]

[1] Centre for Information and Language Processing, University of Munich (LMU), Germany
[2] European Southern Observatory (ESO), Garching, Germany
[3] Institute for Computer Science, University of Munich (LMU), Germany

Received: date / Revised version: date

Abstract. This article reports on the XML retrieval system X^2 which has been developed at the University of Munich over the last five years. In a typical session with X^2, the user first browses a structural summary of the XML database in order to select interesting elements and keywords occurring in documents. Using this intermediate result, queries combining structure and textual references are composed semi-automatically. After query evaluation, the full set of answers is presented in a visual and structured way. X^2 largely exploits the structure found in documents, queries and answers to enable new interactive visualization and exploration techniques that support mixed IR and database-oriented querying, thus bridging the gap between these three views on the data to be retrieved. Another salient characteristic of X^2 which distinguishes it from other visual query systems for XML is that it supports various degrees of detailedness in the presentation of answers, as well as techniques for dynamically reordering and grouping retrieved elements once the complete answer set has been computed.

Key words: XML, graphical user interfaces, interactive information retrieval, schema browsing, answer exploration, result visualization, semi-automatic query creation

1 Introduction

Two tendencies for the current and future development of digital libraries are crucial for the purpose of this paper. First, the amount of books, journals, articles that are made available in a typical digital library grows continously. For example, more than 15 million citations are now available in the PubMed (MEDLINE) database [14], and $460,000$ references have been added in 2002. The PubMed Central digital archive [20], launched in 2000, currently contains issues of some 160 journals as searchable documents. Second, an increasing number of documents in digital libraries come with their logical structure and appropriate meta-information made explicit and accessible. The eXtensible Markup Language XML [2] has become a generally accepted standard for representing such aspects of structure and contents. Often all documents in a digital library – or all entries indexed by a scientific search engine – are annotated with XML, following a common document grammar. In some of our own experiments, e.g., we used a large collection of textual XML documents from the Bertelsmann company, as well as the complete INEX 2004 collection containing more than 12,000 scientific articles totalling to 500 MB [13].

As far as adequate IR technologies are concerned, the continous growth of information represents a serious problem, whereas the explicit encoding of structural aspects and the annotation with meta-information represents a chance. Improved retrieval techniques that take advantage of the structural information to be found in document repositories might represent one key for successfully dealing with large data sets. Two specific challenges are relevant for the system presented in this paper.

First, special querying technologies for XML documents exploiting both their structure (or meta-information) and contents may help to improve the behaviour of traditional flat-text IR systems in terms of the common performance parameters *precision* and *recall*. In fact, structured document retrieval occupies a position in between classic (vague) IR and database technology. On the one hand, suitable references to structure can be used to formulate strict and non-vague conditions on answer documents and thus increase retrieval precision. In a "documents as databases" paradigm, this allows direct access to the more database-like parts of a document, such as its title, author, year of publication, or keywords. On the other hand, queries that are more precise in one part can be made more liberal in other parts, which explains that also a higher recall can be achieved. The improvement of precision

Send offprint requests to: Holger Meuss
Correspondence to: hmeuss@eso.org

and recall seems to be the guiding idea in the heart of most approaches to structured document retrieval.

The second challenge, ignored by many approaches to structured document retrieval, is of equal importance from our point of view. The structural conditions of queries used in XML retrieval induce a specific structure on each answer. It is natural to ask which new forms of user interaction are within reach if this additional information is used in an appropriate way. New interaction paradigms should allow for visual exploration of XML documents, their schema and query results, involving both structure and content in an integrated way. We should depart from the naive idea of classical IR systems which assume that the task is solved merely by passing an ordered list of documents to the user. Some obvious questions are: How can we use the inherent structure of answer documents in order to support new forms of visual and interactive navigation in the answer space, to simplify the selection of relevant documents and (perhaps more importantly, e.g. in the "documents as databases" paradigm) of relevant document parts? How can we use the inherent structure in order to group and order answer documents and their parts, enabling a "bird's eye" perspective, and to adjust the retrieval granularity dynamically?

In this paper we describe the XML retrieval system *X*^{2} (for *querying and eXploring XML*), concentrating on the visual and interactive exploration of (1) the database schema and (2) answer sets to queries. X^2 has been conceived, implemented and tested at the University of Munich over the last five years. While in the beginning we concentrated on algorithmic problems related to query evaluation [15], the focus of our recent work is the second of the aforementioned challenges. After a series of practical tests, a third general objective has been added: In order to formulate a query with interesting structural conditions, the user must have an idea of the structure of documents in the repository. Hence the system should offer some kind of active support for exploring the structure of entries of the library, and for formulating suitable structural conditions in queries. This advocated the integration of a schema browsing facility into X^2 which tightly interacts with the module for visual query creation. As a common frontend to the different interactive modules (schema exploration, query formulation, answer exploration), X^2 provides a graphical user interface (GUI) that helps users to identify, formulate and satisfy their information needs.

Before explaining, at the end of the following section, the structure of the paper which parallels the system architecture to a certain extent, we start with a brief overview of X^2.

2 System overview

The current retrieval model used for X^2, which takes all the abovementioned aspects into account, is shown in Figure 1.

Structure and contents of the database (DB) are summarized in the *DB schema*, which visualizes existing label paths and occurrences of keywords under particular label paths. Using a well-defined set of operations, the user may generate

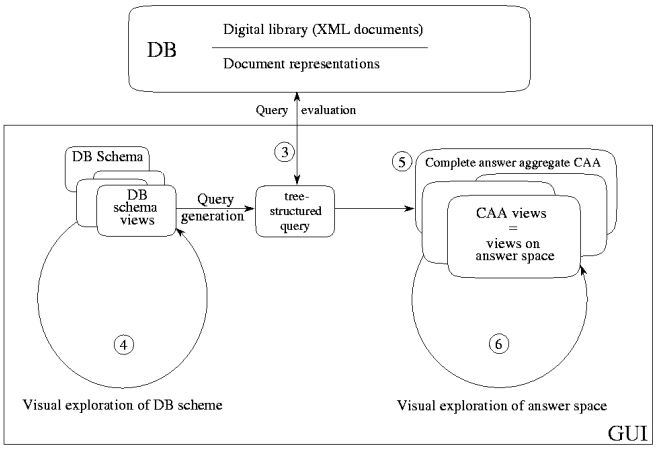

Fig. 1. The retrieval model of X^2. The numbers ③ to ⑥ refer to related sections in the remainder of this article.

distinct simplified *views* on the DB schema, hiding parts that are not of interest. The resulting restricted view, which only contains label paths and keywords that seem relevant to the user, defines a "sphere of interest". This *visual exploration of the DB schema* ends with the *automated generation of a tree-structured query* based on the final view. Of course, the generated query may be post-edited. Alternatively, an experienced user may directly compose a (textual or graphical) query. These additional possibilities are not indicated in the figure.

The query is *evaluated* in the database. The full answer space for the query is computed and represented in the form of a so-called *Complete Answer Aggregate (CAA)* [15]. The CAA collects all nodes (elements) of the database that contribute to some answer, avoiding redundancies. It also encodes structural relationships between relevant elements and organizes document nodes in a way that reflects the structure of the query. Consequently users can easily relate parts of the answer space with nodes and conditions in their query, and therefore fully understand the topology of the answer space. Owing to these properties, CAAs offer a good basis for the last retrieval step to be described next.

The final *visual exploration of the answer space* is again supported by a rich set of manipulation techniques for the visual representation of CAAs. Using these operations, the user may display, group, re-order, or hide elements in the answer set. In this way, the interactive separation of relevant and irrelevant parts of the original answer space is supported. Eventually the exploration leads to a reduced answer space that is nearer to the intended retrieval result and presents elements that are judged relevant, organizing them in a structured way. This reduced answer space represents the final retrieval result, which may be visualized, exported and used in different ways. If the user is not satisfied with the result, the exploration of the answer space might have given valuable hints as to how to create a modified query which better covers the user's real information need. In this sense, our approach is open to query refinement and iteration.

It is important to note that X^2 is a *generic* retrieval system for all sorts of XML documents in digital libraries, independent of the actual domain being described in the documents. It has been used and tested with document collections from a broad range of domains, such as scientific literature databases, linguistic corpora, judicial text documents, lexicographic data and documents containing travel information.

Organization of the paper. In the remaining sections, the central components and ideas behind X^2 are explained in greater detail, following the organisation indicated in Figure 1: Section 3 explains the formal document model underlying X^2 and describes the syntax and semantics of the query language. The database schema, which builds on the notion of a *Content-Aware DataGuide* [25], is introduced in Section 4 where we also describe operations and strategies for the visual exploration of the database schema and for the automated generation of queries. Section 5 introduces Complete Answer Aggregates and explains how this data structure is used to represent the set of all answers to a given tree query in a structured, condensed and non-redundant way. Section 6 describes the visual exploration of the answer space, presenting in more detail a typical retrieval session with X^2. It also covers the integration of result ranking with the system. In Section 7 we sketch some ideas for future work and describe related work.

3 Preliminaries

We briefly comment on possible problems related to distinct information needs that motivate the techniques for user interaction supported by X^2. Afterwards we sketch the formal query model employed in X^2. A more detailed discussion of the mathematical foundations of X^2 can be found e.g. in [15].

3.1 Information needs and complex information spaces

From an abstract point of view, the purpose of any information system is to map the current *information need* of a user into the *information space* spanned by the underlying data resource. To satisfy the information need, appropriate parts of the data must be selected and possibly reorganized, at least in case of structured retrieval, to produce a meaningful and pertinent answer to the user's inquiry. In his influential work on the notion of relevance in Information Retrieval (IR), Mizzaro [16] distinguishes various manifestations of the information need of a user. In a typical retrieval situation, the objective *real information need* underlying an inquiry, which the user is often not fully aware of, is perceived by the user in a particular way. This *perceived information need* is transformed into an (informally) *expressed information need* and finally translated into a *formalized information need*, i.e. a query in a formal query language of an information system, be it a database, a digital library, a web search engine, etc.

Mizzaro's work explains the problems and errors that may occur in this complex transformation process, emphasizing that retrieval systems should offer strong support to the user in order to avoid failure and useless work. Interestingly, less attention has been paid to the reverse translation process, in which users have to relate the query result to their formulated query and perceived information need. Few retrieval systems try to support the user in this "backward" translation process, to the best of the authors' knowledge none of them being an XML system. XML retrieval systems consider their task finished as soon as the formal result has been delivered in the form of XML documents, as they regard only a formalized query as input, ignoring the long way a perceived information need has to go before becoming a formalized information need.

This paper deals with retrieval of XML documents, a situation where both the forward and backward translation process is complex since queries and documents are structured in a non-trivial way. For a typical user who is not quite familiar with the (details of the) document structure, it is difficult to bridge the gap between the perceived and the formalized information need. He may not even be familiar with the information need formalized by a given query he is running, which might have been created earlier by himself or someone else. Therefore the system should offer special assistance in expressing and formalizing the queries. Since answers come with their own structural relations, additional support is also required for relating the items in answer sets to the formalized and to the perceived information need. The graphical user interface of X^2 addresses all these issues by providing iterative, interactive and visual access to queries, documents and answers.

3.2 XML documents as trees

XML documents are formalized as finite ordered labeled *trees*. The set of *nodes* is given by the elements of the document; *edges* describe the direct inclusion of document parts. We distinguish between *text nodes* and *structure nodes*. *Labels* of structure nodes are members of a given set of node labels L (corresponding to XML element types). Text nodes are always leaves and labeled with text. The alphabet of text tokens is Σ. The symbol $<$ denotes the *left-to-right ordering of siblings*, which relates two distinct nodes iff they have a common parent node, whereas $<_{lr}$ stands for the *document order* (capturing XPath's `preceding` and `following` axes).

In the remainder of the paper, standard notions from tree (graph) theory such as *child, ancestor, descendant* etc. are used without further explanations. A textual node v is said to *contain* a keyword k iff the label (textual contents) of v contains k as a token. By contrast, v *governs* k iff k is contained by v or any of its descendants.

For simplicity, attributes, namespaces, etc. are ignored in this informal presentation. Nonetheless, these features can be used in our framework; the current implementation of X^2 distinguishes between attributes, structure nodes (elements) and textual nodes.

3.3 Simple tree queries and full query language

We present the syntax and semantics of the kernel query language of X^2. This language subsumes the core of the W3C's XML Path Language *XPath* [6] and adds extra features like optional nodes and wildcards for search terms. Ignoring additional constraints, our queries have the form of trees that are to be matched against document trees. Using the standard X^2 mode for query formulation, queries are represented in a graphical way in the form of trees (see Figure 4 on page 7 for an example). This (1) makes it easy to understand the meaning of a query; (2) leads to a more concise representation than the term-based linear syntax used in XPath (with its inherent syntactical redundancy [18]); (3) supports various forms of visual user interaction like query formulation, manipulation and presentation; and (4) simplifies the correct association between image nodes in documents and the corresponding query nodes in the visual presentation of the answer space.

From a purely formal point of view, tree queries are described as conjunctive queries over trees [12], which facilitates the use of additional constraints (like horizontal order conditions) as well as the analysis of expressivity and tractability properties [12].

Definition 1 (Atomic path constraint). Let X be a finite set of variables, $x, y \in X, A \in L, w \in \Sigma$. An *atomic path constraint* is an expression of the form (and meaning)

- $x \rightarrow_1 y$ (y is a child of x)
- $x \rightarrow_+ y$ (y is a descendant of x)
- $A(x)$ (structure node x has label A)
- $w\#x$ (token w occurs in the label of text node x).

Definition 2 (Simple tree query). A *simple tree query* is a conjunction of atomic path constraints such that child and descendant constraints impose a tree structure on the set of query variables.

Since child and descendant constraints impose a tree structure on a simple tree query, we may use terms like *root, child, sibling* etc. in the context of such queries. When referring to the tree structure of a query, the aforementioned child constraints and descendant constraints are treated as interchangeable, i.e. both are considered as simple edges of the query. Descendant constraints are called *soft edges*, whereas child constraints are referred to as *rigid edges*.

Definition 3 (Answer to a simple tree query in a document). An *answer* to a simple tree query Q in a document is a variable assignment μ mapping each query variable x in Q to a document node (i.e. XML element) $\mu(x)$, such that all atomic path constraints of Q are satisfied. If we consider a collection $\mathcal{C}$ of documents, then any answer to Q in a document T of $\mathcal{C}$ is called an *answer to Q in $\mathcal{C}$*.

The full query language of X^2 is given by the class of *tree queries*. These queries are more expressive than simple tree queries, extending them with additional constraints:

Order: Siblings in a tree query can be specified to stand in one of two possible order relations: *document order* $<_{lr}$ or *direct order* $<$.

Optional subtrees: A full subtree of a query can be marked as *optional*. An answer μ may be partial in the sense that variables of the optional subtree do not receive images, and constraints involving these variables are ignored.

Root preservation: A full tree query can be marked to be *root-preserving*. An answer μ of a root-preserving tree query has to map the root of the query to the root of a document.

Leftmost/rightmost children: A query node can be marked to be a *leftmost* or *rightmost* child. An answer μ must map leftmost and rightmost children to nodes having no left or right siblings, respectively.

Text wildcards: The alphabet Σ is enriched such that the reserved symbols `*` (representing an arbitrary sequence of letters) and `%` (representing one arbitrary letter) can be used in labels of textual query nodes.

Keyword conjunctions and disjunctions: Keywords in labels of text nodes can be either disjoined or conjoined.

XML attributes: Structural nodes can be equipped with constraints specifying attribute names and values.

This selection of additional constraints has proved to cover a range of useful and expressive constructs while guaranteeing efficient evaluation.

4 Visual schema exploration and query generation

In Section 3.1 we explained Mizzaro's [16] observation that the successful translation of the real information need of a user into a formalized information need is a difficult process. From this picture it is clear that the usability and effectiveness of a retrieval system crucially depends on the kind of support that is offered to formulate queries which really cover the actual information need.

We now decribe the *SchemaBrowser* used in X^2. The motivation for such a browser has been discussed in the introduction. Before explaining the functionality of this tool, a few words on the kind of schema we use are in order.

4.1 DTDs versus DataGuide and CADG

Consider a DTD as the one shown in Figure 2, which describes the documents used in the following examples. We call DTDs *normative schemata*, since they impose a structure on a document collection that must be obeyed by XML documents. Typically a DTD permits a variety of content models that might not occur in a given conforming document collection. This is contrasted by structural summaries like the *DataGuide* [11] or *Content-Aware DataGuide (CADG)* [25], which are *descriptive schemata* of a document collection: a DataGuide summarizes the currently expressed document structure in tree structure such that all document nodes with the same label path to the root are represented by a single node in the index tree, called an *index node*.

```
<!-- ........... library ........... -->
<!ELEMENT library (publication*)>

<!-- ......... publication ......... -->
<!ELEMENT publication (meta, content)>
<!ATTLIST publication type (article|book|proceedings|
                    technicalReport|thesis) #REQUIRED>

<!-- .......... metadata ........... -->
<!ELEMENT meta (authors, title, subtitle?,
                inPublication?, volume?,
                number?, pages?, editor?,
                publisher?, date?, uri?)>

<!ELEMENT authors (author+)>
<!ELEMENT author (#PCDATA)>

<!ELEMENT title (#PCDATA)>
<!ELEMENT subtitle (#PCDATA)>
<!ELEMENT inPublication (#PCDATA)>

<!ELEMENT volume (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT pages (#PCDATA)>

<!ELEMENT editor (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>

<!ELEMENT date (month?, year)>
<!ELEMENT month (#PCDATA)>
<!ELEMENT year (#PCDATA)>

<!ELEMENT uri (#PCDATA)>

<!-- ........... content ........... -->
<!ELEMENT content (abstract?, section+, bibliography?)>

<!ELEMENT abstract (paragraph+)>
<!ELEMENT paragraph (#PCDATA)>

<!ELEMENT section (title, subsection+)>
<!ELEMENT subsection (title?, paragraph+)>

<!ELEMENT bibliography (reference+)>
<!ELEMENT reference (meta)>
```

Fig. 2. Sample DTD

The DataGuide has originally been designed as a main memory index structure for semistructured documents. Evaluating a query against a DataGuide involves (1) finding document nodes with the right label path by matching query paths to index paths, and then (2) filtering out those document nodes whose textual content (held in secondary storage) does not contain the query keywords.

The CADG enhances this concept by storing additional keyword information in the index nodes (not necessarily physically, as shown in [25]). Thus the relevance of an index node (and all document nodes it represents) for a given query keyword may be decided during the path matching process, which drastically reduces access to secondary storage. For a detailed description and evaluation of the CADG, see [25].

4.2 A SchemaBrowser based on the CADG

In our work, the CADG not only serves as an index structure for the document collection being queried. It is also the basis for the SchemaBrowser, which visualizes the CADG in the form of a GUI tree, thus providing a compact and simplified view on the database schema.

From the internal representation of the CADG, other useful information about the database is obtained and displayed on demand to give a more precise picture of its contents. This extra-schema information includes e.g. the list (or else the number) of keywords contained in or governed by an index node[1], as well as the number of label paths in the database with a given prefix or where a given keyword occurs.

Query generation. As described in the sequel, the user interactively adapts the view on the database schema provided by the SchemaBrowser until he has obtained enough information to formulate a query. Simultaneously, the user decides which part of the schema should be mentioned in the query. To reflect the user's "sphere of interest", the resulting query typically not only contains nodes for selecting parts of documents with a specific content, but also "output" nodes representing parts of the documents the user wishes to see regardless of the text they contain (see Query 1 for an example). The SchemaBrowser then feeds this subschema as input into a *QueryGenerator*, which produces an editable graphical query representation according to the user's selection. In terms of Mizzaro, this translation is meant to support users in the error-prone transformation process from perceived to formalized information need.

Adjusting focus and selectivity. During the exploration of the schema, the aforementioned statistical information provided by the SchemaBrowser helps the user estimate the expected size of the answer space. He may choose to include or avoid particular labels and keywords in his query in order to render it more selective and keep the query result reasonably small. For instance, realizing that the keyword "markup" occurs too frequently, the user may decide to search for "XML" instead. Returning to Mizzaro's picture, in this phase the user (1) compares his perceived information need with the content and structure of the document collection, thus getting closer to his real information need, and (2) expresses this information need – in terms of a subschema and the resulting query – such that it can be successfully matched to the documents in the collection.

Browser details. Figure 3 shows the SchemaBrowser, which visualizes the schema of an XML literature database conforming to the DTD from Figure 2. In this representation, index nodes can be *activated* (shown in red/bold) with the mouse or by using the widgets on the right: the user can activate all nodes with a certain label, of a certain type (element oder attribute), or nodes that contain or govern a given keyword. In Figure 3 all *title* nodes were selected by using the operation *activate all nodes with label 'title'*. One can also inspect contained or governed keywords of a marked node (highlighted in blue) in a separate window, as shown for the *author* node in Figure 3. Various inheritance and set operations are supported for propagating node activations, such

[1] An index node contains (governs) a keyword if any of the document nodes it represents contains (governs) that keyword, as defined in Section 3.2.

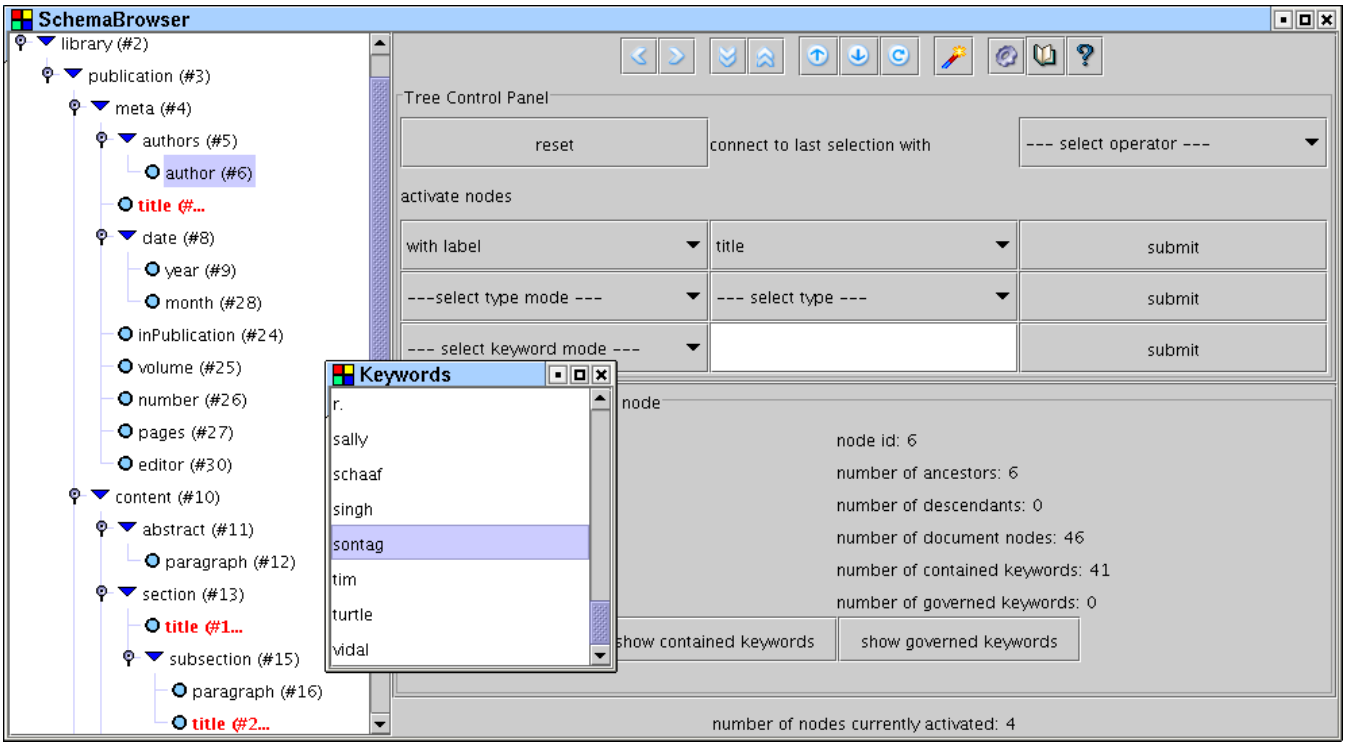

Fig. 3. View of the SchemaBrowser

as intersection, set complement, propagation to ancestors etc. Activated nodes can be hidden and, most importantly, they can serve as a basis for automatically generating a query tree.

The panel on the lower right in Figure 3 depicts some of the extra-schema information obtained from the underlying CADG. We plan to extend this rudimentary statistic summary with some data mining functionality.

Schema exploration and query generation: an example. The following sample information need illustrates the capabilities of the SchemaBrowser and QueryGenerator:

Query 1. *"I would like to see interesting recent papers on XML written by Sally Sonntag."*

This (expressed) information need is transformed into a formalized information need (query) step by step: First, we look at the list of all available authors by marking the upper `authors` node and clicking the button *show governed keywords*. In this way we see that there is no author with surname "Sonntag". We realize that there has been a slight mismatch between our expressed information need and the real information need, which might refer to the surname "Sontag", a name that in fact appears in the list of authors.

As we are interested in papers on XML by Sally Sontag, we activate all nodes containing the keyword "Sontag" or governing the keyword "XML" and hide all other nodes in the index tree. The resulting keyword-specific tree in Figure 4, containing only nodes which are potentially relevant to the above information need, is much smaller and hence easier to browse. After manually deactivating some nodes not to be reflected in the answer (like `library`, e.g.), we let the SchemaBrowser generate a query which we will further edit to reflect our above mentioned information need: A textual query node with the keywords "Sally Sontag" is added to the `author` node, and a textual query node with the keyword "XML" to the `section` node. Eventually we add an optional `year` node to the `publication` node.[2] The resulting query is shown in Figure 4. This query contains nodes that seem unnecessary at first sight, since they do not restrict the query (like the optional `year` node or the `section` node). They are included nonetheless as "output nodes", reflecting part of the "sphere of interest" during exploration of the answer set, as we will see later. Every query node is represented by a rectangle, structural nodes are annotated with the node's label, textual query nodes are labelled with the word `<text>` and show the keyword being queried. A rigid edge is expressed by a solid line, while for soft edges a dashed line is used.

Generating query trees automatically from nodes selected in the index tree has several advantages. While inspecting the index tree the user learns about the structure of the underlying document. In our example he will become aware of the fact that there are `author` nodes on two different levels, thus being able to specify in his query whether he is interested in authors of publications in the document collection or authors of cited papers. This underlines the fact that the free use of the full XML structure (as opposed to the restricted field search in database-oriented query systems, see Section 7.1) allows for the formulation of new and more precise queries.

[2] In the visual query representation optional query nodes are represented by a dashed rectangle.

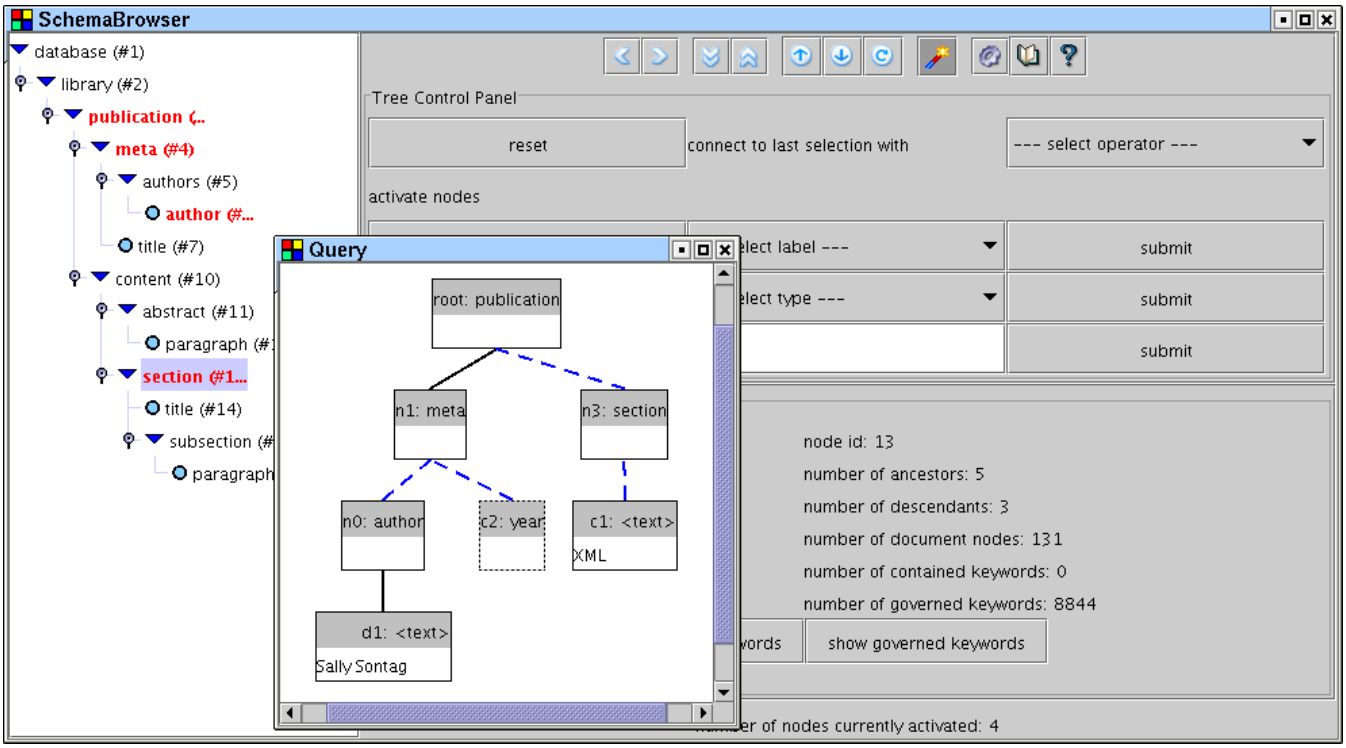

Fig. 4. The generated and post-edited Query 1

5 Complete Answer Aggregates

For the sake of usability, XML retrieval systems should not present a query result as a simple enumeration of individual answer mappings (see Defintion 3 on page 4), as suggested by the following observations:

- There might be exponentially many answer mappings, which aggravates the user's burden of conquering the result space.
- Distinct answer mappings may share common parts (submappings). This again makes it difficult for a user to understand the topology of the result space completely, compare and analyze distinct answers and relate them to each other.
- An answer mapping, in its basic form, is a flat concept (list of query node/document node pairs) that ignores the geometric form of tree queries and document trees. This makes it less amenable to visual presentation and exploration.

In particular, these issues discourage the presentation of search results as a list of flat documents, which is common in traditional IR systems. In X^2, by contrast, the full set of answers is computed and visually represented in the form of a *Complete Answer Aggregate (CAA)*. This data structure is introduced informally here, a formal definition can be found in [15]. CAAs collect answers in a structured, condensed and non-redundant way, using mechanisms for structure sharing. From a complexity point of view, this leads to query evaluation in polynomial time (see below), even though the total number of single answer mappings may be exponential. The visual presentation of CAAs (see Figure 5) encourages and supports various exploration and browsing techniques to be presented in the next section.

A CAA inherits its macro structure from the tree structure of the query Q: For each query node (variable) v, a *slot* s_v is computed that collects all possible images of v in answers to Q. In the GUI, distinct slots are represented as boxes and graphically arranged in a way that directly reflects the structure of the query, as shown in Figure 5. Each box/slot s_v for a given variable v is graphically partitioned into subrectangles that represent the possible image nodes of v. In the following, these subrectangles are called *fields*. For both slots and fields, the GUI provides a variety of display modes (*views*) differing for example in their degree of detailedness. These views are chosen individually for each slot and field in a context menu.

In the internal representation of a CAA, pointers are used to encode the following relationship: assume that in Q there exists a (soft or rigid) edge from variable u to v (i.e., v is a child of u in Q). Let d and e respectively denote possible images of u and v that are listed in s_u and s_v and represented as fields. Then d is linked to e iff there exists an answer to Q that maps u to d and v to e. Guaranteeing that all single answer mappings can be reconstructed from the CAA, these links are used in the GUI to display structural dependencies between image nodes (fields) of distinct variables. In this way, e.g., the matching part of each retrieved document can be visualized.

For the following examples, the reader should keep in mind that only those database nodes are added to the CAA and represented as fields that participate in at least one *full*

answer mapping. Hence the set of fields assigned to a variable depends on the conditions imposed on other query variables. In this sense, the CAA *exactly* represents the document nodes of the answer space.

The algorithm for computing the CAA (described in [15]) for a query Q runs in time $O(|Q| \cdot |D| \cdot h_D \cdot log(|D|))$ where $|Q|$ and $|D|$ respectively denote the size of the query and the database, and h_D denotes the maximal depth of a path in a document. If no ordering constraints are used, then the complexity is $O(|Q| \cdot |D| \cdot h_D)$, which is optimal [15]. We developed a special index structure for textual XML documents, the CADG [25]. Using CADGs, typically only a small portion of the database has to be visited during query processing, which contributes to the high overall efficiency of X^2.

6 Interactive and Visual Result Exploration

This section deals with the graphic representation and handling of Complete Answer Aggregates, illustrating the main exploration capabilities of X^2 in a real-world retrieval scenario. The exploration techniques, while presented in a small-scale example for didactic reasons, also apply to larger answer sets. A systematic overview of all exploration features is given in Section 6.2. The last part addresses scalability issues which arose during our experiments with very large test collections. We show how integrated result ranking and thresholding cope with unselective queries and large answer sets.

6.1 A Sample Exploration Session with X^2

Suppose the user runs two queries against a digital library of scientific papers (contents and metadata) conforming to the DTD shown in Figure 2. The first query is driven by the following information need:

Query 0. *"I would like to know in which papers Sally Sontag cited her colleague Jim Jones."*

which is expressed by the query shown in Figure 5. The user would like to end up with a list of cross-references as a basis for further literature research. The example is meant to show how the user implicitly takes advantage of the higher precision and result specificity[3] provided by structured document retrieval (as opposed to flat-text search), e.g. for inspecting an interesting document excerpt.

The CAA obtained for the query, also given in Figure 5, has the same tree-shaped macro structure as the query. Initially, all slots in the aggregate tree simply display the respective number of their fields (*initial view*). Curious about which papers were selected, the user switches to the *summary view* on the `publication` slot (see Figure 6). In this view, the slot expands in size to show the individual fields it contains, each accompanied by a short descriptive label extracted from the

[3] Retrieval precision is defined as the percentage of relevant documents among those retrieved. The fewer irrelevant parts of documents a structured query result contains, the more specific it is.

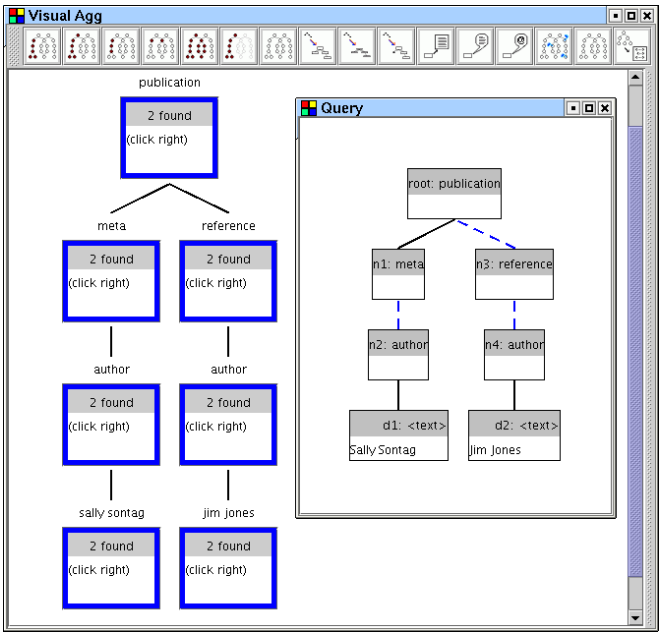

Fig. 5. Query 0 and initial view on the corresponding CAA

corresponding document. As shown in the figure, the description here consists of the paper's title. By contrast, the user activates the *exhaustive view* on the `reference` slot, which provides more detailed information on its fields. As depicted in Figure 6, the authors and title of each cited paper are displayed, along with an optional URI and publication date. The exhaustive view thus provides the user with a linked list of all relevant references. The two `author` slots, collecting the document nodes where the authors Sontag and Jones appear in the metadata, were only needed for selecting the right documents during query processing and can be hidden during browsing, as indicated in Figure 6.

In the above example, a flat retrieval result would consist of a set of entire documents, each with much irrelevant content besides the actual references. Moreover, since there would be no means to specify that "Sally Sontag" must occur as author of a paper and "Jim Jones" as author in a reference, the user would need to filter out manually those documents where Jones cited Sontag, or Jones and Sontag published together, or someone else cited both Sontag and Jones, etc.

A second example is meant to illustrate the various exploration modes supported by CAAs, including clustered and re-ordered views of slots, as well as activation, inheritance, and inspection of fields and slots at different granularity levels. It shows how answer sets can be scaled down during answer exploration by (1) aggregation and (2) hiding answer elements. These techniques prove especially valuable when dealing with larger result sets (but see Section 6.3 for a discussion of scalability issues). The information need in this example is expressed by the query from Section 4.2 (see Figure 4 on page 7):

Query 1. *"I would like to see interesting recent papers on XML written by Sally Sontag."*

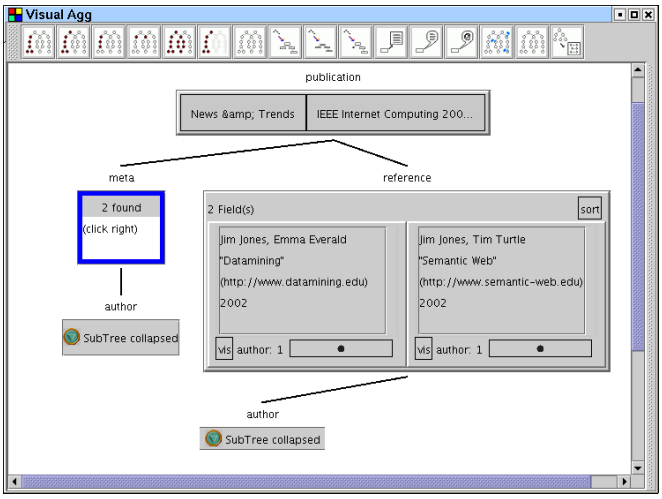

Fig. 6. List of references obtained from the CAA in Figure 5

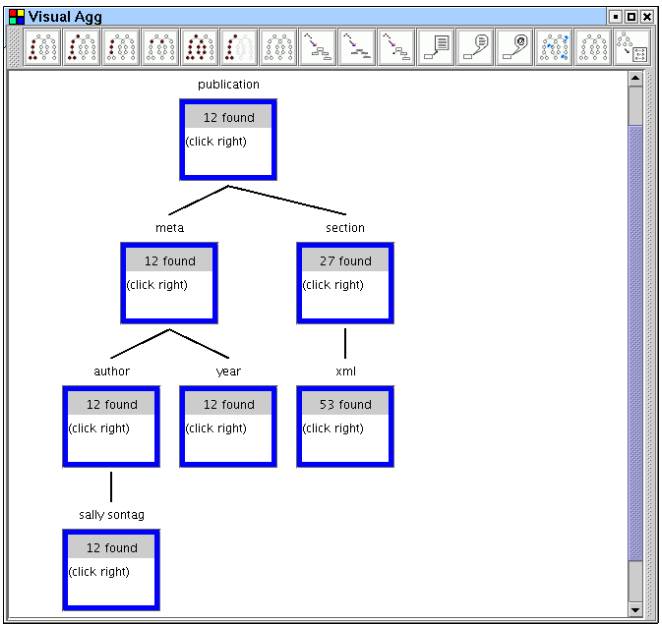

Fig. 7. Initial view on the CAA for Query 1 (see Figure 4)

As can be seen from the field counts in Figure 7, the matches for this query do not fit the rendering space available. One way to structure this query result is to cluster and order salient parts of the documents according to their textual content. For instance, enabling the *cluster view* on the `year` slot, the user can browse through the XML-related publications by Sally Sontag published in 2003, 2002 and so on (see Figure 8). In a situation where the document collection in question provides appropriate mark-up, categories like “database systems”, “information retrieval” etc., assigned to each publication, could be used to cluster matching documents according to the research field they belong to.

In a next step, the user aims to reduce the number of items being displayed in the CAA. To this end, he *activates* fields by a mouse click (activating all fields of a cluster can be done

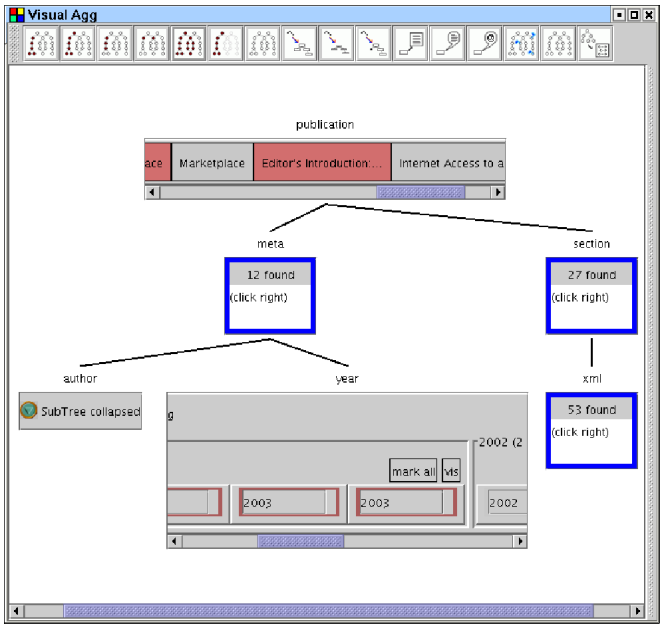

Fig. 8. Clustering and activating fields in the CAA from Figure 7. Fields belonging to papers published in 2003 are highlighted red.

in one step by using the respective *mark all* button). In our example, the user activates all fields in the 2003 cluster of the `year` slot. Activated fields are highlighted red (e.g. in Figure 8). Activations can be passed through to other fields by choosing one of several *inheritance* mechanisms for the full CAA. In our example the inheritance mechanism is simple co-occurrence in the same document, i.e. all fields belonging to the same document as one of the activated fields are now activated, too. By pressing a dedicated *show activated-only* button in the menu bar, only activated nodes are displayed in the CAA, as can be seen in Figure 9. Note that as a result, all slots contain only the highlighted fields belonging to papers published in 2003.

Faced with this reduced number of documents, the user starts exploring the remaining fields in the `publication` slot. A small floating frame, similar to the quick info or tool tip in common GUIs and triggered by a button in the menu bar, displays a *field info* for each slot entry being hovered, as shown in Figure 9. The field info data is automatically extracted from the documents. In this scenario, some metadata of the publication in question is listed.

The article on XML indexing techniques, whose field info is displayed in Figure 9, arouses the user’s interest. He now wishes to see the sections on XML in a restricted table of contents, which is not part of the document schema (see the DTD in Figure 2 or alternatively the CADG in Figure 3 on page 6). To this end, he abandons the previous activations and activates only the respective field in the `publication` slot, along with all fields belonging to the desired document. Again this is achieved by inheritance. Since we are still in the *show activated-only* mode, only parts of that particular publication are now displayed in the aggregate (see Figure 10). The user may now simply inspect the list of highlighted `section`

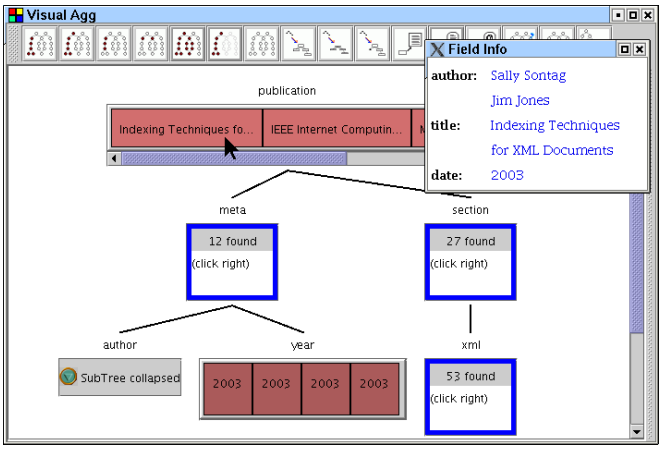

Fig. 9. Field info for a particular paper in the CAA from Figure 8. The aggregate now only contains fields belonging to documents published in 2003.

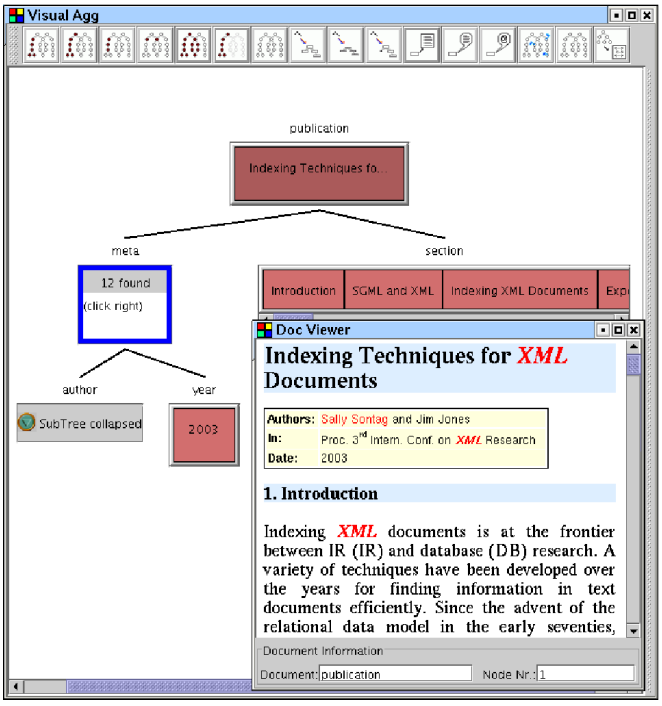

Fig. 10. Table of contents obtained from the CAA in Figure 9. The CAA has been further restricted to display only one particular publication from 2003. A *document viewer* shows that paper with the search keywords highlighted.

fields in the summary view, which assembles all headings of top-level sections in the order of their appearance in the paper (see Figure 10).

In the last step of this example, the user chooses to see the paper about indexing XML in a *full-text view*. Triggered by the field context menu, the entire document content is displayed in a separate *document viewer* window, as illustrated in Figure 10. The formatted document content is automatically generated from the XML source using a stylesheet for transformation. In order to establish the relation to the user's information need, all occurrences of the query keywords are highlighted in the text. After inspecting the full document, the user may either terminate the retrieval session or continue browsing the CAA, or else enter a new retrieval step by reformulating the query.

6.2 Exploration Techniques in X^2

The sample exploration session above has demonstrated many features of X^2. We will now briefly review these features in a more systematic way and explain others that have not been mentioned so far. Note that while not all of the following functionality makes sense for the sample document collection used in this paper, it proved to be useful for most of the collections X^2 has been tested with.

Views. In general, a view defines how a field corresponding to an XML node is displayed in the CAA. Views can be selected individually for slots and fields. Apart from the *inital view*, the *summary view*, and the *exhaustive view* already described, the current implementation of X^2 provides a *technical view* displaying the internal CAA structure in more detail, a *minimal view* for hiding a slot and its descendants, and a *cluster view* (see below). Novel views can easily be plugged into the system in order to adapt X^2 to special requirements of new document collections. While this is currently done programmatically (without modifying the document collection), we plan to define views using XSLT stylesheets in the future.

Node activation and inheritance. The context menu associated with fields in a slot allows to activate arbitrary many documents nodes individually, which are then highlighted red. In order to display hierarchic relations and order relations between nodes, there are a number of *inheritance* mechanisms that propagate activations through the CAA. An inheritance mechanism is always chosen for the entire CAA. Nodes having inherited an activation which are not individually activated by the user do not propagate activation themselves. They are displayed in a lighter red than individually activated nodes. The button *show activated-only* in the menu bar of the CAA window serves to hide all document nodes not being activated (whether inherited or individually). Apart from inheritance based on document co-occurrence, which has already been used in the example, there is a variety of inheritance mechanism based on hierarchic node relation like ancestor/descendant, and on order relations.

Node ordering. Each slot's context menu provides a variety of ordering mechanisms for fields in that slot, based on their textual content, attribute values or the number of children they have in the CAA. Alternatively, fields may be sorted according to their relevance score (see Section 6.3). They are displayed in ascending or descending order according to either numerical or lexicographical comparison.

Clusters. Fields can be clustered according to their textual content, attribute values, or the number of their children. In the current implementation, all fields contained in a cluster are still visible, whereas we plan in the future to treat field

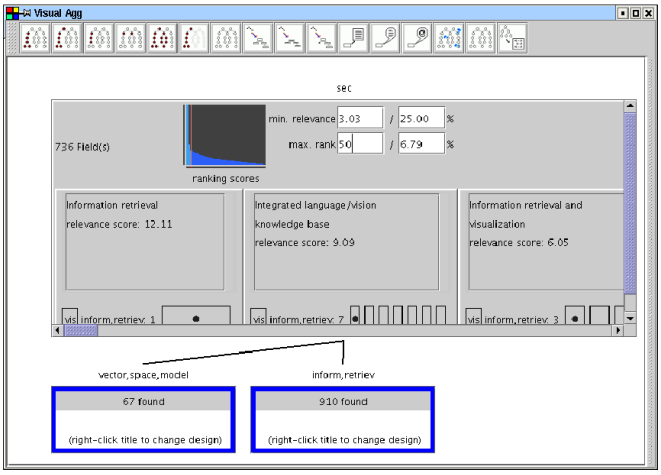

Fig. 11. Fields sorted by decreasing relevance w.r.t. a query against the INEX 2004 collection [13]. Only the 50 most relevant fields in this slot are displayed, corresponding to nearly 7% of the answer set (*top-k thresholding*).

clusters like individual fields with respect to display and most interaction functionality.

Windows: favourites, field info, document viewer. There are three extra windows supporting the user in browsing and navigation. The *favourites window* allows to store selected nodes from a CAA temporarily. This is helpful when the user needs to keep track of visited nodes in a large answer space. The *field info* and the *document viewer* windows have already been introduced in the examples above.

6.3 Result ranking and scalability issues

While the aforementioned visualization techniques help to organize the query result and reduce the number of fields to be displayed simultaneously, they are not entirely satisfactory when applied to large answer sets. As a matter of fact, visualizing a slot containing some hundred or even more fields in a compact and useful manner requires refined techniques which allow the user to (1) reduce the number of fields displayed in the slot, and (2) sort the remaining fields by decreasing relevance w.r.t. the query. Figure 11 shows a CAA computed for a rather unselective query against the 500-MB INEX 2004 collection [13] which gives rise to the scalability issues just mentioned. The user searches sections talking about the vector space model as well as Information Retrieval in general. As indicated in the upper left corner of the sec slot in Figure 11, the system retrieves 736 sections, each with a relevance score computed according to one of the implemented ranking models, currently either *XPRES* [27] or *s-term* [22]. (For a comparison of these and other models, see [26].)

The exhaustive view on the sec slot visualizes the relevance distribution over all fields in a small histogram. Each vertical bar in the histogram represents one or more fields (depending on the total number of fields in the slot) whose greatest relevance score determines the height of the bar. The red vertical line in Figure 11 symbolizes the number of fields currently displayed in this slot, which we refer to as the *rank threshold*. Those bars which are left of the threshold are highlighted, such that the user can tell from a quick glance at the histogram which portion of the answer set is currently visible. The four numerical displays next to the histogram serve both to visualize the exact threshold value and to accept a new threshold, which immediately affects the histogram and the displayed fields. The lower two displays, labelled “max. rank”, hold the rank threshold both as an absolute value and relatively to the total number of fields in this slot. The upper two displays, labelled “min. relevance”, represent the *relevance score threshold*, i.e. the minimal relevance score of any field displayed in the slot, again both as an absolute value and as the percentage of the greatest relevance score of any field in the answer set, respectively. The relevance threshold is visualized as a horizontal line in the histogram, as shown in Figure 12, and triggers highlighting of those bars reaching above the threshold.

In order to reduce the number of fields to be displayed, the user sets either the rank threshold or the relevance score threshold by typing an appropriate absolute or relative value into the corresponding text field. When the upper text boxes are updated with a new relevance score threshold, the corresponding rank threshold is displayed automatically in the lower boxes, and vice versa. Alternatively, the user may drag the red line in the histogram to a new position. To this end, mouse-sensitive threshold selectors appear next to the histogram, a vertical one for setting the relevance score threshold and a horizontal one for the rank threshold. Histogram and text fields are synchronized and update each other when receiving user input. No matter which way the thresholds are modified, the slot is updated immediately to show the appropriate number of fields in descending order of relevance.

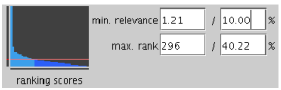

Fig. 12. Threshold selection by relevance score

In our sample result from the INEX collection, the user decides first to explore the 50 most relevant sections about the vector space model and Information Retrieval (Figure 11), and then expands the view to see all section reaching at least 10% of the maximal relevance score (Figure 12). Using these two variants of *top-k* queries, X^2 allows for an intuitive and dynamic adaption of the result size which is most useful when searching particular information in large answer sets. By contrast, the presented methods like ranking, reordering and clustering offer only limited support for data mining purposes on XML documents. Here suitable interaction and visualization techniques are yet to be developed.

7 Conclusion

7.1 Related work

Before surveying related approaches to visual retrieval and exploration, a brief comparison of common form- or field-

based query interfaces to native XML systems like X^2 is in order. In the more database-oriented query facilities for structured documents which are currently most common especially in a web context, retrieval is restricted to occurrences of search terms in a static set of predefined fields. While field-based search interfaces are easy to use, they cannot capture the full hierarchical structure of XML documents which they therefore hide rather than exploit in the way X^2 does: (1) to increase retrieval precision and (2) to present the search results in a structured, explorable form that relates them to the exact part of the query they matched. Moreover, field search must be tailored specifically to the schema of a given document collection (like e.g. the Entrez search interface for MEDLINE [14], or the various systems related to computational biology surveyed in [3]) and therefore needs manual adaption to accomodate new collections or changes to the schema of a collection. We argue that the hierarchic nature of XML documents calls for a generic treatment in query systems in order to expose its powerful features in retrieval and maintenance.

An early predecessor of graphical query languages in the field of semistructured data is *GOOD* [9], a retrieval system for graph databases. In the retrieval model employed by GOOD, a query result is a subgraph of the database graph which matches the query. The structure of the query, however, is lost in the answer, and GOOD does not offer exploration techniques for the answer subgraph.

Most query languages for XML exploit visualization techniques for query construction and representation only [17, 7, 23]. Closest in spirit to our work are the *DataGuide* (when used for schema browsing as in [10], or for query generation as described in [11]) and *XML-GL* [5]. The latter is a full-fledged XML query language for retrieval and manipulation of XML documents that uses visual syntax elements only. XML-GL does not support interactive query formulation and result exploration. The DataGuide as a means for schema browsing is introduced in [10]. A simple and intuitive formalism in [11] sketches how to use the DataGuide to construct a path query incrementally and explore the document collection at the same time. The DataGuide browser relies on a graphical tree representation with expandable nodes like the one presented in Section 4. In contrast to our work, this model cannot handle queries that are more complex than simple paths or contain descendant (i.e., soft) edges. Besides, the CADG [25] underlying our SchemaBrowser exploits its additional keyword information to provide a content-specific view on the database schema. Neither [11] nor [25] covers techniques for exploring the query result. [15] introduces CAAs in this context, but does not investigate schema browsing or query creation. *BBQ* [17] provides interactive and visual query generation. Users select elements from the DTD underlying a given document collection and arrange these elements into queries. Compared to the query generation approach presented here, this has the disadvantage that DTDs, being normative schemata (see Section 4.1), generally allow for content models that do not occur in the current document collection. In addition, queries generated with BBQ can only be post-edited in a textual form.

Obviously, ranking mechanisms are another means to simplify the inspection of large answer sets. Dedicated ranking schemes for structured document retrieval currently attract much attention in IR research [21, 22, 8, 27, 24, 26]. X^2 is a system that is to a large extent independent of the ranking mechanism used, hence the research papers mentioned above are complementary to the issues discussed in the paper at hand.

Interactive and iterative techniques have a strong tradition in IR. For instance, various methods for iterative query refinement and relevance feedback stem from IR research. However, since "mainstream" IR focusses on flat textual documents, visual exploration techniques are less common. Systems like *Scatter/Gather* [19], being based on visualizations of document distance metrics, are completely complementary in their visual metaphors, so that they can hardly be compared to the approach presented here.

Related work from the field of database systems has been collected and carefully analyzed in a survey by Catarci et al. [4]. Visual retrieval systems are categorized according to (1) the graphic representations that are used for data and queries on the one hand and for query results on the other hand, (2) the kind of interaction strategies that are offered to help users in understanding the database content and in formulating appropriate queries, and (3) syntax, semantics and expressivity of the query languages they support.

As to the kernel languages and functionalities, there are also similarities between X^2 and wrapper generation tools such as *Lixto* [1]. Lixto comes with a sophisticated GUI to allow users to interactively construct wrappers that locate and collect valuable pieces of information in structured documents.

7.2 Discussion and Future Work

The contributions of X^2 can be summarized as follows:

- full-fledged and seamlessly integrated *visual* access to the XML documents, schemata, queries, and retrieval results
- novel exploration, aggregation and rudimentary data mining techniques for visualizing the structure of retrieval results and the entire document collection
- use of CAAs for relating structured search results to the structure of the query and documents in an interactive exploration process
- integration of database and IR techniques with special consideration of the hierarchic nature of the underlying XML data, to access the more database-like parts of a document
- improved precision w.r.t. flat-text retrieval

We tested the applied techniques with a wide spectrum of applications, among others: (1) linguistic data containing XML representations of syntax trees, (2) touristic data containing descriptions of cities with hotels, sightseeing sites, restaurants, traffic information etc., and (3) the INEX 2004 collection of the IEEE Computer Society's journal publications from 1995 to 2002 [13]. We observed that although our query and answer representations are flexible and generic

enough to cope with such heterogeneous domains, some applications needed special query constructs not foreseen in our initial design. Among these constructs added later were e.g. a variety of mechanisms for querying order relations which proved helpful in linguistic research. Similarly, the way element content is displayed in a CAA must be tailored to a given document collection sometimes. However, a generic solution which does not alter the collection is indispensable for obvious reasons. We therefore developed a simple and generic framework for specifying how element descriptions are generated, which is yet to be improved (see Section 6.2).

The development of X^2 is still work in progress. Accompanying the system development, prototypes have been presented and used in Information Retrieval courses at the University of Munich in 2002 and 2003, and positive and negative aspects have been discussed. The main features approved by the students include (1) the way search results are explored interactively and systematically, (2) the fact that users can choose between different views on individual parts of the search results, and new domain-specific views can be integrated into the system, and (3) the flexible and expressive query formalism which can refer to all parts of the XML structure, unlike retrieval systems allowing access to a predefined set of fields only. Major occasions for criticism were the following:

1. It is difficult to formulate queries if the XML structure is unknown or only partially known to users.
2. The visualization should be based more upon standardized techniques and widgets, e.g. for displaying trees.
3. When faced with a large search result, users easily get lost, even if techniques to scale down the number of displayed elements (see Secions 6.1 and 6.2) are used.
4. The system's performance degrades when exploring (not computing) large answer sets, causing exploration of big CAAs to become sometimes tedious.

The first problem was solved to a large extent with the integration of the SchemaBrowser into X^2 (see Section 4). We plan to address the second issue by using well-known visual metaphors in the near future. We would also like to investigate whether a form-based web query interface could be adapted to offer at least part of the flexibility of structured document retrieval with X^2. In response to the third objection, we equipped the system with ranking mechanisms recently proposed for XML documents (see Section 6.3). Combined with thresholding facilities for hiding elements whose relevance score is too low, they provide the most intuitive way for a user to organize the answer space. Finally, we are still struggling to decrease the response time of the GUI in the case of big CAAs, which is partly due to the Java GUI libraries we use.

Apart from the points mentioned above, we currently concentrate on the following major goal:

Fully interleaving query formulation and result exploration. In the current version of X^2, users first formulate their queries, supported by the SchemaBrowser, and afterwards browse the query result. We plan to depart from this strictly sequential model, visualizing query and result in the form of a single query/result object that is extended, manipulated and explored at the same time. The development of such a model is facilitated by the fact that CAAs inherit their macro-structure from the query.

Acknowledgements

We would like to thank Marcus Schilling and all other students who helped with their programming skills and motivation to make X^2 a working system. We also thank the anonymous referees for their detailed and helpful comments on a preliminary version of this paper.

References

1. R. Baumgartner, S. Flesca, and G. Gottlob. Visual Web Information Extraction with Lixto. In *The VLDB Journal*, pages 119–128, 2001.
2. T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation, 2000.
3. F. Bry and P. Kröger. A Computational Biology Database Digest: Data, Data Analysis, and Data Management. *Distributed and Parallel Databases*, 13(1):7–42, 2002.
4. T. Catarci, G. Santucci, and J. Cardiff. Graphical Interaction with Heterogeneous Databases. *The VLDB Journal*, 6(2):97–120, May 1997.
5. S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. XML-GL: a graphical language for querying and restructuring XML documents. *Computer Networks*, 31(11-16):1171–1187, May 1999.
6. J. Clark and S. DeRose. XML Path Language (XPath), version 1.0. W3C Recommendation, November 1999.
7. S. Cohen, W. Nutt, and A. Serebrenik. EquiX - Easy Querying in XML Databases. In *WebDB'99, Proc. Int. Workshop on the Web and Databases*, 1999.
8. N. Fuhr and K. Großjohann. XIRQL: A Query Language for IR in XML Documents. In *Research and Development in IR*, pages 172–180, 2001.
9. M. Gemis, J. Paredaens, and I. Thyssens. A visual database management interface based on GOOD. In *Proc. Int. Workshop on Interfaces to Database Systems*, 1993.
10. R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured DBs. In *Proc. 23rd Int. Conf. on Very Large DB*, 1997.
11. R. Goldman and J. Widom. Interactive Query and Search in Semistructured Databases. In *International Workshop on the Web and Databases*, pages 52–62, 1998.
12. G. Gottlob, C. Koch, and K. U. Schulz. Conjunctive queries over trees. In *PODS'04 Symposium on Principles of Database Systems*, 2004.
13. Initiative for the Evaluation of XML Retrieval (INEX). Available at `inex.is.informatik.uni-duisburg.de:2004`, 2004. Organised by the DELOS Network of Excellence for Digital Libraries.

14. MEDLINE – Querying with PubMed. Available at `www.ncbi.nlm.nih.gov/PubMed`. A service offered by the U.S. National Library of Medicine.
15. H. Meuss and K. U. Schulz. Complete Answer Aggregates for Tree-like Databases: A Novel Approach to Combine Querying and Navigation. *ACM Transact. on Information Systems*, 19(2):161–215, 2001.
16. S. Mizzaro. How Many Relevances in Information Retrieval? *Interacting with Computers*, 10(3):303–320, 1998.
17. K. D. Munroe and Y. Papakonstantinou. BBQ: A Visual Interface for Browsing and Querying of XML. In *5th IFIP Working Conf. on Visual Database Systems*, pages 277–296, 2000.
18. D. Olteanu, H. Meuss, T. Furche, and F. Bry. XPath: Looking forward. In *Proc. of the EDBT Workshop on XML Data Management (XMLDM)*, volume 2490, pages 109–127. Springer, 2002.
19. P. Pirolli, P. Schank, M. A. Hearst, and C. Diehl. Scatter/Gather Browsing Communicates the Topic Structure of a Very Large Text Collection. In *ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*.
20. PubMed Central. Available at `www.pubmedcentral.nih.gov`. A service offered by the U.S. National Library of Medicine.
21. T. Schlieder. Similarity Search in XML Data using Cost-Based Query Transformations. In *Proc. 4th Intern. Workshop on the Web and Databases*, 2001.
22. T. Schlieder and H. Meuss. Querying and Ranking XML Documents. *JASIS Spec. Top. XML/IR 53(6):489-503*, 2002.
23. A. Sengupta and A. Dillon. Query by templates: A generalized approach for visual query formulation for text dominated databases. In *Conf. on Advanced Digital Libraries (ADL'97)*, 1997.
24. A. Theobald and G. Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In *Proc. 8th Int. Conf. on Extending DB Technology*, pages 477–495, 2002.
25. F. Weigel, H. Meuss, F. Bry, and K. U. Schulz. Content-Aware DataGuides: Interleaving IR and DB Indexing Techniques for Efficient Retrieval of Textual XML Data. In *Proc. 26th Europ. Conf. on IR*, 2004.
26. F. Weigel, H. Meuss, K. U. Schulz, and F. Bry. Content and Structure in Indexing and Ranking XML. In *Proc. 7th Intern. Workshop on the Web and Databases*, 2004.
27. J. E. Wolff, H. Flörke, and A. B. Cremers. Searching and browsing collections of structural information. In *Proc. IEEE Forum on Research and Technology Advances in Dig. Lib.*, pages 141–150, 2000.