Enhancing User Interaction and Efficiency with Structural Summaries for Fast and Intuitive Access to XML Databases

Felix Weigel

Centre for Information and Language Processing (CIS)
University of Munich (LMU), Germany
weigel@informatik.uni-muenchen.de

Abstract. This paper describes an ongoing Ph. D. project whose goal is to improve both the user interaction and the efficiency of XML databases. Based on *structural summaries* for indexing and visualizing the structure of XML data, we describe a highly interactive, intuitive GUI targeted at non-experts, and present sophisticated algorithms and data structures for efficient query evaluation which enable a smooth user interaction. Preliminary results illustrate how XML exploration, indexing, querying, caching, ranking and user feedback in XML databases can benefit significantly from the structural summaries.

1 Introduction

XML has by now become a de-facto standard for modelling, querying, exchanging and storing a broad range of data with different characteristics. At the one end of the spectrum, there are text-centric documents with little structure to be queried, e.g., web pages, Wikis, Blogs, news feeds, e-mail and FAQ archives. At the other end, there is XML content with a much more rigid and meaningful structure and little text, e.g., product catalogues, tax payer's data submitted via electronic forms and bibliography servers. While traditional Information Retrieval (IR) techniques have been established for the former class of data, XPath and XQuery [1] are the languages of choice for the latter.

However, XML is most commonly used for a wide variety of truly semistructured data in between those two extremes, with complex and irregular structure adding significant information to the rich textual content. Examples are documents in digital libraries or publishing houses, electronic encyclopedias, on-line manuals, linguistic databases and scientific taxonomies. For querying such data neither database nor IR-style methods are well suited. On the one hand, we cannot expect unskilled users of these applications to express their information need in XPath or XQuery, languages which are also inapt for ranked retrieval in XML data with an irregular or (partly) unknown schema. On the other hand, flat-text IR disregards valuable structural hints which can help not only to formulate more precise queries, but also to rank and present query results in a meaningful form. Although there is much recent work on ranking structured documents, the user interaction in such systems mostly follows the database paradigm[1].

[1] See, e.g., the IR extensions to XPath (in the *INEX* benchmark [2]) or XQuery [3].

We argue that to make the full spectrum of XML applications accessible to non-experts, a new way of user interaction with XML databases is needed which helps them understand and use the schema of the data in an intuitive way. It is known from earlier experience with relational databases (RDBSs) and IR engines that sophisticated query languages alone are not enough for serving the information needs of unexperienced users. Making users *benefit* from the XML structure poses *challenges* to both the user interface and the query kernel:

Schema exploration. Exploiting the inherent document structure allows for more precise and useful queries and answers. However, sometimes users find the markup just meaningless, either because they ignore the underlying schema or because they are faced with instructions for layout, document processing, namespaces, etc.[2] Rather than to present the structure of the data one-to-one, interfaces to XML databases should therefore allow users to explore the schema and selected sample data, as well as to create views customized to their information need and degree of expertise.

Structured query results. When presenting query results to users, the XML structure relates distinct parts of the result and defines boundaries for highlighting matches in a meaningful context. Yet users also need to recognize how results relate to the query and schema. When exploring results, users often wish to change part of the query and reevaluate it. Ideally this could be done in a seamless iterative feedback without leaving the result view.

Efficient query evaluation. Encouraging vivid user interaction via integrated schema exploration, querying and result browsing makes sense only with a fast query kernel ensuring prompt system reaction. A major challenge is the incremental query evaluation based on previously retrieved results in an XML query cache, which is needed for smooth user interaction and feedback.

Structured ranking. Structural query hints can be used to present retrieved elements in order of relevance. However, computing relevance scores for structured documents is non-trivial. Open problems are, e.g., how to find elements which cover exactly a relevant portion of text, how to handle structural near-misses and how to avoid overlapping elements in the query result efficiently.

We believe that the key to easy, intuitive user interaction with XML databases is the tight integration of graphical views on the document schema, sample data, user queries and retrieval results on top of an efficient and scalable query kernel. The goal of the Ph. D. project is to address the above challenges with (1) a novel graphical user interface (GUI) providing integrated schema, query and result views, (2) index structures and algorithms for efficient, scalable and incremental query evaluation, (3) efficient and effective techniques for XML ranking. In a first phase, we have investigated how *structural summaries* visualizing the document schema also improve the efficiency of the query kernel. Most of this work, though targeting enhanced user interaction, applies to XML databases in general. Hence

[2] For instance, with its 30,000 distinct label paths, the schema of the INEX collection of scientific articles [2] is much more complex than the actual logical structure of the documents, and therefore hard to understand for non-expert users.

we present our contributions in the context of more recent approaches which have appeared since the beginning of the project in late 2003. To the best of our knowledge our approach to intuitive and efficient user interaction is unique in that we employ the same structural summary for (a) exploring the schema and samples of the data, (b) formulating, planning, evaluating and caching queries, (c) ranking query results and (d) enabling iterative user feedback.

The next section sketches the state of the art in XML retrieval, highlighting techniques we build upon. Section 3 explains in detail our solutions to the aforementioned challenges, presenting the main contributions of the project. These are summarized and briefly contrasted with existing approaches in Section 4, which also gives a snapshot of the accomplished work and remaining issues.

2 XML retrieval: state of the art and open problems

User interaction with XML databases. As mentioned in the introduction, XML retrieval is typically addressed either from an IR or a database viewpoint, which is also reflected in the main features of the systems. While the focus of XML databases is efficient XQuery processing, IR engines optimize ranking effectiveness (precision/recall). By contrast, user interaction has been somewhat neglected so far[3]. Most systems from both camps offer only a text interface to queries and results, rendering query results as XML fragments [4]. More sophisticated GUIs focus on visual query creation [5,6,7,8,9,10] while disregarding the exploration of query results. In particular, it is hard to figure out how results relate to the query and schema. [11] introduces a compact result view which reflects the query structure, but is not linked to the query view. All these tools assume prior knowledge of the document schema for creating meaningful queries. [8,10] support DTD-driven query assistance and restructuring, but lack schema and result browsing. In [6,12] structural summaries are used to visualize the document schema, provide sample content and facilitate manual query formulation to some extent. However, there is little work on adapting IR-style relevance feedback for smooth iterative XML retrieval. In particular, we are not aware of any solution to efficient incremental query evaluation based on user feedback.

Index and storage structures. As more and more multi-gigabyte data sets need to be stored and queried, efficiency and scalability of XML databases have received much attention. IR systems often use inverted lists built over tag names or keywords as index structures. This hinders scalability, joins of large node lists being expensive. Many database approaches are based on the *DataGuide* [6], a compact representation of the document structure where each distinct label path appears exactly once. Query paths are matched on this *structural summary* in main memory rather than the entire data set on disk, which avoids some joins and disk I/O. The DataGuide has been combined with different IR techniques for text matching, such as signature files, tries and inverted lists.

[3] No new approaches to user interaction were presented at the *Interactive* and *Relevance Feedback* tracks of INEX 2004 [2].

A complementary indexing technique are *labelling schemes*, which assign elements specific IDs from which part of the document structure is inferred without accessing the entire data set. Recently many new schemes addressing different query problems on XML trees have been proposed. For instance, *interval schemes* [13] encode the descendant axis by representing an element v as a numeric interval I_v such that v is a descendant of v' iff $I_v \subset I_{v'}$. They are space-efficient but sensitive to node insertions and join-intensive. *Prefix schemes* [14] represent an element as the sequence of sibling positions on its root path, similar to the section numbers in this paper. Simple label manipulations allow to infer ancestors and siblings. Binary encodings have been applied to reduce the space overhead of prefix schemes. Node insertions can be handled gracefully [15], whereas the problem of labelling graph-shaped XML remains open.

While earlier semistructured [16] and most IR engines are native systems, recent approaches treat XML retrieval as an application for RDBSs. Different methods of storing and querying XML in tables have been proposed [17,15,13]. Supporters of the native approach [18,4,19] argue that the best performance is achieved by tailoring the system to the XML data model. However, the question whether native or relational systems are more efficient, which largely affects indexing, joining as well as query planning and optimization, is still unresolved.

Caching of XML query results. Reusing cached query results is an instance of the more general problem of query processing in the presence of *views* on the database, which has been studied extensively for RDBSs. Major problems are *query containment/overlap* (compare the definitions of queries/views to decide whether their extension overlap or contain one another) and *query answering* (based on the view definitions and extensions, decide whether a given piece of data is part of the result of a specific query). Many papers have studied the theoretical complexity of these problems with different query languages for semistructured data. For instance, [20] shows that both are PSPACE-complete for tree-shaped and EXPSPACE-complete for arbitrary conjunctive queries of regular path expressions. Despite the high theoretical complexity, a number of different approaches strive to push the practical efficiency to its limits, building on native XQuery engines [21,22], two-way finite state automata [20], tree automata [23], incomplete trees [24,25], or LDAP servers [26].

Ranked XML retrieval. Unlike flat-text IR, XML ranking must cope with (1) relaxing query structure in order to capture near misses, (2) balancing structural and textual query conditions when computing relevance scores, (3) defining an inverted element frequency, (4) choosing suitable elements to return (rather than entire documents), and (5) avoiding result overlap. Since the first INEX benchmark [2] in 2002, many performance metrics and new ranking models have been proposed, most of which adopt flat-text methods such as *tf·idf* [27,28,29]. A web-inspired technique is to exploit the link structure in document collections [30,31]. Recently the use of ontologies for query expansion has been studied more thoroughly [32,31]. However, little work is concerned with the efficient implementation of XML ranking techniques.

3 Contributions of the project

3.1 Indexing and exploring XML data with the CADG index

At the core of our efficiency enhancements is the *Content-Aware DataGuide (CADG)*, a path summary inspired by the DataGuide [6] which tightly integrates IR techniques for efficient keyword search. Those parts of the schema which do not lead to occurrences of query terms are pruned early during path matching. Experiments show that this considerably reduces disk I/O, improving the performance of the original DataGuide by up to a factor 200 [33]. As a first step toward enhanced user interaction (Sect. 3.6), we created a graphical CADG [34] which extends the DataGuide visualization in [6] (Fig. 1). The schema is rendered as a tree with highlightable nodes. The user can view sample keywords occurring under specific label paths and statistical information about their distribution in the data. For complex schemata (e.g., INEX benchmark), the user may simplify the CADG by hiding subtrees based on tag names or textual content. Unlike [6], we also use the CADG for creating complex tree queries in a semiautomated manner.

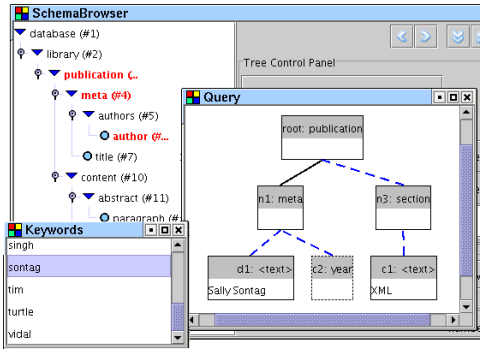


Fig. 1. CADG visualization.

3.2 Ranked XML retrieval with the CADG index

As mentioned above, the question how to rank structured documents both efficiently and effectively is still open. Rather than to commit ourselves to a particular model, we therefore classified existing approaches w.r.t. the path and term frequencies they use for computing relevance scores. In [35] we show how to adapt the CADG to indexing precomputed frequencies for different classes of ranking model, which speeds up the retrieval process. To evaluate our approach experimentally, we implemented the *S-Term* ranking model [36] and tested it with the INEX 2004 benchmark [2]. The system scaled up well to over 500 MB in terms of retrieval speed, but we discovered that even with precomputed frequencies the scoring of certain queries takes quite long [37]. Clearly this is due to deficiencies inherent to the S-Term model. Although not central to the Ph. D. project, we might therefore develop a simplified model which avoids too complex scoring and at the same time addresses some of the challenges mentioned in Sect. 2.

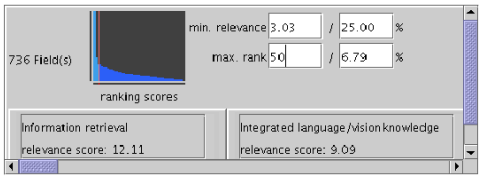


Fig. 2. Ranking visualization.

We also use ranking in our preliminary GUI for rendering result lists, which may become large for unselective queries. With a new *threshold histogram* (Fig. 2) the user specifies a minimum relevance score (or alternatively, a maximum rank) for items to be displayed, dynamically adapting the number of items in the result list below the widget.

3.3 The BIRD labelling scheme

In a third step we developed the *BIRD* labelling scheme [38] for avoiding joins of large node sets, a common bottleneck in XML query evaluation. Although the CADG matches leaves of query paths without such joins, BIRD achieves huge benefits when retrieving nodes higher on the path, or common ancestors in tree queries. The key to reducing the join effort is that BIRD not only *decides* all XPath axes for two given elements, but also infers ancestors, siblings and children of a single element from its label alone, for which we coined the term *reconstruction*. Combined with the CADG, reconstruction avoids additional disk I/O for looking up matches to the branching nodes of a tree query. As shown in Fig. 3, BIRD labels (small numbers) are multiples of integer *weights* (large numbers), which are stored in the CADG. For instance, to reconstruct the parent of element 43, we look up the parent weight (5) in the CADG and simply compute the parent label as $43 - (43 \bmod 5) = 40$, and likewise for other ancestors.

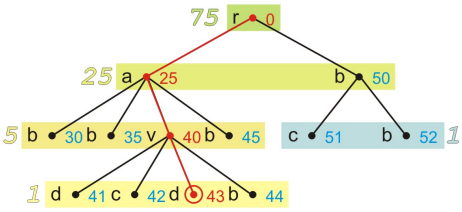


Fig. 3. BIRD ancestor reconstruction. Small numbers denote BIRD node labels, whereas large numbers indicate BIRD weights. Here all children of the same node have the same weight regardless of their label paths.

In [38] we study (1) the benefit of reconstruction over decision and (2) how BIRD compares to other labelling schemes in terms of expressivity (reconstruction/decision of different axes), processing time, space consumption and robustness against updates. In our experiments with five labelling schemes BIRD outperforms all competitors with equal expressivity in terms of space and time. The only comparable approach, μPID [39], generates smaller labels but is less expressive than BIRD. Although BIRD is not as robust against updates as other schemes such as ORDPATH [15], further experiments illustrate that a simple strategy minimizes the impact of node insertions at least for a certain class of data set. Still we would like to investigate advanced updating techniques for BIRD sketched in [38]. Finally, a comparison of different query algorithms in the paper confirms that (1) reconstruction is indeed most effective for speeding up query evaluation, (2) schemes with excessive label size may incur a performance overhead due to inefficient node comparison, and (3) labelling schemes respecting document order (such as BIRD, e.g.) benefit from extra optimization techniques. These encouraging results motivated the combination of BIRD and the CADG in a relational setting (see the next section).

Currently we are surveying a multitude of new approaches with distinct features which have appeared meanwhile, including an extensive comparative analysis and evaluation of more than twenty labelling schemes in terms of time and space efficiency, robustness against node insertions and expressivity. While a previous survey [40] classifies a small number of approaches into *bit-vector*, *interval* and *prefix* schemes, we subsume BIRD and a few other encodings under a fourth class, *multiplicative schemes*.

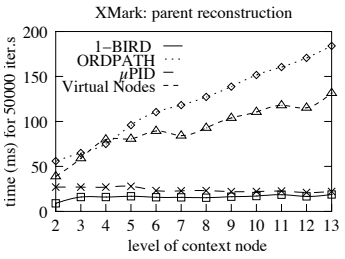


Fig. 4. Reconstruction speed of BIRD and other approaches.

3.4 Relational XML retrieval with the RCADG index

pid	par	max	lab	type	lev	weight	csig	gsig	keys	elts
#0		#5	people	elt	0	27	000000	111111	0	1
#1	#0	#5	person	elt	1	9	000000	111111	0	1
#2	#1	#2	name	elt	2	3	010010	010010	2	2
#3	#1	#5	profile	elt	2	3	000000	101111	0	2
#4	#3	#4	edu	elt	3	1	101100	101100	2	2
#5	#3	#5	gender	elt	3	1	001011	001011	2	2

Fig. 5. Relational CADG (RCADG).

The following work contributes to the discussion of native vs. relational XML retrieval systems (see Section 1). Given that both approaches have been established and pursued without much cross-fertilization taking place so far, we examined how our native XML indexing techniques can boost the retrieval of XML stored in an RDBS. The goals in the context of the Ph. D. project were (1) to improve the scalability of our native prototype system whose performance degraded for unselective queries, (2) to benefit from RDBS features such as concurrency or recovery and (3) to store intermediate results temporarily during query evaluation, in anticipation of a future incremental query kernel (see the next section). In [41] we show how to migrate the CADG and BIRD to the relational data model, applying interval labelling [42] to CADG nodes and BIRD labelling to document nodes. The resulting *Relational CADG (RCADG)* replaces the structural summary in main memory with a single table containing one row for each CADG node (see Figure 5), including its interval-scheme labels (columns pid, max), BIRD weight (*weight*), CADG-specific keyword signatures (*csig*, *gsig*) and statistical path information for query planning and ranking (*keys*, *elts*).

XML queries against the RCADG are evaluated entirely within an RDBS as a sequence of SQL statements. The translation algorithm makes heavy use of BIRD's reconstruction capabilities to minimize the number and the size of intermediate result node sets to be joined. For instance, to evaluate the query in Figure 6 only q_3 and q_5 are matched by joining the RCADG table with the element table, whereas matches to q_1, q_4 and q_6 are obtained via reconstruction. Thus the number of joins is reduced by 50% compared to other relational approaches. Subsequent evaluation steps each produce a more complete intermediate result table from prior results, which also enables relational index support

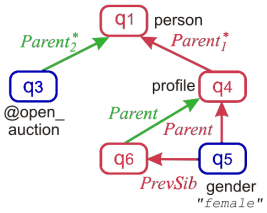


Fig. 6. RCADG query.

for elements reconstructed on the fly. This technique is rewarded in the experiments where we compare the RCADG to the native CADG/BIRD system, a relational version of the interval scheme and another relational DataGuide variant that uses string matching on paths [17]. Unlike the two relational competitors, the RCADG fully preserves the underlying document schema in the RDBS. This avoids false hits for certain queries on recursive data sets (which we observed for the string-matching approach) and also enables query optimization techniques that take into account XML path statistics ignored by the relational optimizer.

From the study on relational XML retrieval with the RCADG, we learned that (1) exploiting native indexing techniques such as BIRD and the CADG in an RDBS boosts the query performance by up to three orders of magnitude compared to both native and relational approaches, (2) the proposed techniques significantly improve the scalability both in terms of the query complexity and selectivity, (3) these benefits are achieved with only a negligible space overhead, but (4) the performance gains may be deteriorated by inappropriate query planning and rewriting. Our preliminary planning algorithm needs to be refined to cope with more involved cases where the selectivity of a particular query node on the one hand and the analysis of applicable reconstruction steps on the other hand favour conflicting query plans.

3.5 Incremental query evaluation with an XML query cache

The user interaction described below encourages the continuous modification of prior queries in an iterative relevance feedback process. To this end, queries need to be evaluated *incrementally*, i.e., common subsets of results to different queries should not be retrieved repeatedly from scratch but reused with the least possible computational effort. This means that (1) final or intermediate results to previous queries must be materialized at least temporarily and (2) subsequent queries must be analyzed to find out which parts of the results they share with previous ones. The first requirement is satisfied in a natural way by the RCADG (see above). To meet the second requirement, we have developed a novel XML query cache [43] from which reusable result subsets can be retrieved efficiently with the help of schema information. A new query is first matched on the schema level (which can be done very fast in the RCADG's path table). The resulting *schema hits* are decomposed into pairs of label paths which are then looked up in the query cache to find prior queries with overlapping schema hits. The results of these queries are available in their RCADG result tables for further processing.

This approach has three benefits, which distinguish it from the earlier work we know of. First, comparing query *extensions* (i.e., results) rather than *intensions* (i.e., tree patterns or XQuery expressions) works around the high complexity of query containment (Sect. 2). Second, comparing schema hits before accessing the full query results helps to discard useless cached queries efficiently. Third, by caching intermediate and final matches to all parts of a query we can reuse partial query results and create new query plans to compute the complete result incrementally. Note also that only schema hits are kept in the main-memory part of the cache, whereas the full query extensions reside in the RDBS backend.

3.6 Integrated schema exploration, querying and result browsing

The current GUI described in [34] provides separate views on the schema, queries and results. Once a query has been formulated and evaluated, the retrieved hits are explored in a graphical representation reflecting the query structure. While browsing the result view, users often wish to modify the query, realizing mismatches with their information need. Currently this requires re-editing and re-running the query outside the result view (perhaps after consulting the schema again). This not only causes needless computations to find data that is already known, but also makes it hard for the user to keep track of updates to the query result.

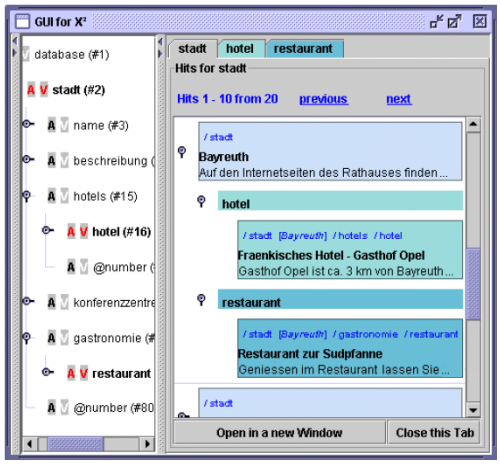
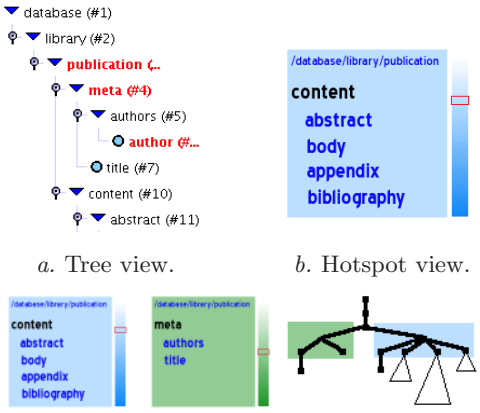


Fig. 7. Preliminary GUI.

The most salient feature of the new GUI to be developed is the tight integration of the schema, query and result views. Ideally the user would silently issue new queries or modify previous ones while browsing the document schema or query result, as follows. First the user activates interesting label paths, possibly with keyword constraints as before. The occurrences of these label paths span a substructure in the documents which in turn induces a partial schema specific to the current activation. In the schema view, the CADG is immediately updated, e.g., by hiding paths outside the reduced schema. Note that finding the current substructure requires efficient tree matching in the documents, just like for processing explicit user queries.

At any point in time the user can either narrow down or expand the CADG by changing the path activation. Moreover, distinct paths can be *merged*, i.e., treated as equivalent both in query evaluation and in the GUI. Conversely, occurrences of the same label path can be distinguished, based on their textual content or statistics such as subtree size, by *splitting* the corresponding node in the schema view. To some extent this blurs the distinction between the schema and the actual data. However, query results for user-specified parts of the schema



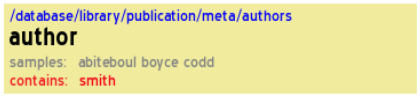
a. Tree view. *b.* Hotspot view.

c. Bird's eye overview.

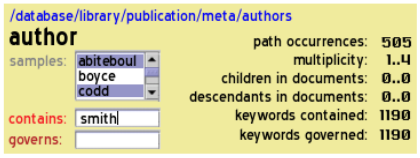
Fig. 8. Alternative schema views.

are still displayed in a dedicated view (see Figure 7, right-hand side), using different profiles determining the desired level of detailedness as well as backlinks into the schema view for locating the corresponding label paths.

Inspired by the capabilities of common file system browsers, we intend to provide alternative views on the schema. Fig. 8 sketches two possible schema presentations, the *tree view* (*a.*) introduced before and the *hotspot view* (*b.*) which displays the root path and children of a single schema node. The *level widget* to the right in Fig. 8 *b.* indicates how deep the node is buried in the document hierarchy. Multiple hotspot views may be opened in separate frames or integrated with the tree view. A *bird's eye overview* locates the hotspots in the document hierarchy (Fig. 8 *c.*). Individual schema nodes may be rendered at distinct levels of detailedness. Two alternatives to the simplistic rendering in Fig. 8 *a.* are illustrated in Fig. 9. In a concise read-only profile (Fig. 9 *a.*), sample keywords and user-specified query keywords are shown. By contrast, the full profile (Fig. 9 *b.*) provides widgets to edit these properties and also displays some path statistics.



a. Concise schema node profile.



b. Full schema node profile.

Fig. 9. Alternative schema node profiles.

Meanwhile we have specified the intended user interaction with the new GUI in terms of a clean formal algebra. More specifically, we compiled a set of *A-operations* for activating label paths in the schema view, and *E-operations* for exploring the query results. A preliminary implementation covers some A- and E-operations in the GUI, but does not yet trigger the appropriate evaluation steps. Further remaining tasks include the merging and splitting of label paths and an analysis of the expressiveness of the interaction algebra.

4 Summary and discussion

The Ph. D. project presented here aims to improve both the user interaction and the efficiency of XML databases. Major contributions are (1) a highly interactive, intuitive *GUI for non-expert users*, (2) *index and storage structures* for efficient exact or ranked XML retrieval and (3) algorithms for the *incremental evaluation of XML queries*. The work on index and storage structures is finished. We have thoroughly analyzed our techniques in comparison with existing methods and demonstrated their practical use in extensive experiments, including scalability tests for various data sets up to 9 GB in size. The use of structural summaries such as the CADG has been shown to be particularly apt for the envisaged user interaction, because it provides an intuitive graphical access to the document schema and at the same time accelerates query evaluation. This twofold benefit was already discussed in the context of the *Lore* system [6]. We significantly

extend that work by (1) combining the DataGuide with IR techniques for keyword search and ranking (*CADG*), (2) boosting its performance by means of a novel labelling scheme (*BIRD*), (3) migrating the resulting index to an RDBS (*RCADG*), (4) using it for similarity search in an XML query cache, (5) integrating it with query and result views for iterative feedback-driven search.

The studies and experiments with XML ranking have shown that our indexing techniques support the efficient query evaluation with different models, and have given us a clear picture of what needs to be improved. As far as the quality of the S-Term ranking is concerned, the performance at INEX 2004 is encouraging but leaves room for optimizing the specificity of the results. For better response times, the computation of relevance scores needs to be simplified.

References

1. Boag, S., Chamberlin, D., et al.: XQuery 1.0. W3C Cand. Rec. (2005)
2. INEX: (Initiative for the Evaluation of XML Retrieval)
3. Amer-Yahia, S., et al.: XQuery 1.0 and XPath 2.0 Full-Text. W3C W. Dr. (2005)
4. Paparizos, S., Al-Khalifa, S., Chapman, A., Jagadish, H., et al.: TIMBER: A Native System for Quering XML. In: Proc. SIGMOD Conf. (2003)
5. Gemis, M., Paredaens, J., Thyssens, I.: A Visual Database Management Interface based on GOOD. In: Proc. Int. Worksh. on Interfaces to Database Systems. (1993)
6. Goldman, R., Widom, J.: DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In: Proc. VLDB Conf. (1997) 436–445
7. Munroe, K.D., Papakonstantinou, Y.: BBQ: A Visual Interface for Browsing and Querying of XML. In: Proc. Conf. on Visual Database Systems. (2000) 277–296
8. Comai, S., Damiani, E., Fraternali, P.: Computing Graphical Queries over XML Data. Trans. Inf. Syst. **19** (2001) 371–430
9. Berger, S., et al.: Xcerpt and visXcerpt: From Pattern-Based to Visual Querying of XML and Semistructured Data. In: Proc. VLDB Conf. (2003) 1053–1056 (Demo).
10. Braga, D., Campi, A., Ceri, S.: XQBE (XQuery By Example): A Visual Interface to the Standard XML Query Language. Trans. Database Syst. **30** (2005) 398–443
11. Meuss, H., Schulz, K.U.: Complete Answer Aggregates for Tree-like DBs: A Novel Approach to Combine Querying and Navig. Trans. Inf. Syst. **19** (2001) 161–215
12. Chawathe, S.S., Baby, T., Yeo, J.: VQBD: Exploring Semistructured Data. In: Proc. SIGMOD Conf. (2001) 603 (Demo).
13. Boncz, P., Grust, T., van Keulen, M., Manegold, S., et al.: Pathfinder: XQuery—The Relational Way. In: Proc. VLDB Conf. (2005) 1322–1325
14. Tatarinov, I., Viglas, S., Beyer, K.S., et al.: Storing and Querying Ordered XML Using a Relational Database System. In: Proc. SIGMOD Conf. (2002) 204–215
15. O'Neil, P., O'Neil, E., Pal, S., Cseri, I., et al.: ORDPATHs: Insert-Friendly XML Node Labels. In: Proc. SIGMOD Conf. (2004) 903–908
16. McHugh, J., Abiteboul, S., Goldman, R., Quass, D., Widom, J.: Lore: A Database Management System for Semistructured Data. SIGMOD Record **26** (1997) 54–66
17. Yoshikawa, M., et al.: XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases. Trans. Int. Tech. **1** (2001) 110–141
18. Fiebig, T., Helmer, S., Kanne, C.C., Moerkotte, G., Neumann, J., et al.: Anatomy of a Native XML Database Management System. VLDB Journal **11** (2002) 292–314
19. Galax: Open-source XQuery reference implementation. (`www.galaxquery.org`)

20. Calvanese, D., et al.: View-based Query Answering and Query Containment over Semistructured Data. In: Proc. Int. Worksh. Database Prog. Lang. (2002) 40–61
21. Chen, L., Rundensteiner, E.A.: XQuery Containment in Presence of Variable Binding Dependencies. In: Proc. Int. Conf. World Wide Web. (2005) 288–297
22. Shah, A., et al.: Improving Query Perf. using Materialized XML Views: A Learning-based Approach. In: Proc. Int. Worksh. XML Schema Data Mgt. (2003) 297–310
23. Chen, L., Rundensteiner, E.A., Wang, S.: XCache: a Semantic Caching System for XML Queries. In: Proc. SIGMOD Conf. (2002) 618–618 (Demo).
24. Hristidis, V., Petropoulos, M.: Semantic Caching of XML Databases. In: Proc. Int. Worksh. Web and Databases. (2002) 25–30
25. Abiteboul, S., Segoufin, L., Vianu, V.: Representing and Querying XML with Incomplete Information. In: Proc. Symp. Principles of Database Systems. (2001)
26. Marrón, P.J., Lausen, G.: Efficient Cache Answerability for XPath Queries. In: Proc. Int. Worksh. Data Integration over the Web. (2002)
27. Wolff, J.E., Flörke, H., Cremers, A.B.: XPRES: A Ranking Approach to Retrieval on Structured Documents. Technical Report IAI-TR-99-12, Univ. Bonn (1999)
28. Fuhr, N., Großjohann, K.: XIRQL: A Query Language for Information Retrieval in XML Documents. In: Proc. Int. Conf. Research Developm. IR. (2001) 172–180
29. Theobald, M., Schenkel, R., Weikum, G.: An Efficient and Versatile Query Engine for TopX Search. In: Proc. VLDB Conf. (2005) 625–636
30. Guo, L., Shao, F., Botev, C., et al.: XRANK: Ranked Keyword Search over XML Documents. In: Proc. SIGMOD Conf. (2003) 16–27
31. Graupmann, J., et al.: The SphereSearch Engine for Unified Ranked Retrieval of Heterogeneous XML and Web Documents. In: Proc. VLDB Conf. (2005) 529–540
32. Theobald, A., Weikum, G.: The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking. In: Proc. EDBT Conf. (2002) 477–495
33. Weigel, F., Meuss, H., Bry, F., Schulz, K.U.: Content-Aware DataGuides: Interleaving IR and DB Indexing Techniques for Efficient Retrieval of Textual XML Data. In: Proc. Europ. Conf. Information Retrieval. (2004) 378–393
34. Meuss, H., Schulz, K.U., Weigel, F., et al.: Visual Exploration and Retrieval of XML Document Collections with the Generic System X^2. Dig. Lib. **5** (2005) 3–17
35. Weigel, F., Meuss, H., Schulz, K.U., Bry, F.: Content and Structure in Indexing and Ranking XML. In: Proc. Int. Worksh. Web and Databases. (2004)
36. Schlieder, T., Meuss, H.: Querying and Ranking XML Documents. Journ. American Society for Information Science and Technology **53** (2002) 489–503
37. Weigel, F., Meuss, H., Schulz, K.U., Bry, F.: Ranked Retrieval of Structured Documents with the S-Term Vector Space Model. In: Proc. INEX Worksh. (2004)
38. Weigel, F., Schulz, K.U., Meuss, H.: The BIRD Numbering Scheme for XML and Tree Databases – Deciding and Reconstructing Tree Relations using Efficient Arithmetic Operations. In: Proc. Int. XML Database Symposium. (2005) 49–67
39. Bremer, J.M., Gertz, M.: Integrating XML Document and Data Retrieval Based on XML. VLDB Journal (2005) Online First.
40. Christophides, V., Scholl, M., Tourtounis, S.: On Labeling Schemes for the Semantic Web. In: Proc. Int. Conf. World Wide Web. (2003)
41. Weigel, F., Schulz, K.U., Meuss, H.: Exploiting Native XML Indexing Techniques for XML Retrieval in Relational Database Systems. In: Proc. Int. Worksh. Web Information and Data Management. (2005)
42. Li, Q., Moon, B.: Indexing and Querying XML Data for Regular Path Expressions. In: Proc. 27th VLDB Conf. (2001) 361–370
43. Weigel, F., Schulz, K.U.: Caching Schema Information and Intermediate Results for Fast Incremental XML Query Processing in RDBSs. Submitted for publ. (2006)