ToX – The Toronto XML Engine[†]

Denilson Barbosa[a] Attila Barta[a] Alberto Mendelzon[a]
George Mihaila[b] Flavio Rizzolo[a] Patricia Rodriguez-Gianolli[a]

[a]Department of Computer Science
University of Toronto
{dmb,atibarta,prg,mendel,flavio}@cs.toronto.edu

[b]IBM T.J. Watson Research Center
mihaila@us.ibm.com

We present ToX – the Toronto XML Engine – a repository for XML data and metadata, which supports real and virtual XML documents. Real documents are stored as files or mapped into relational or object databases, depending on their *structuredness*; indices are defined according to the storage method used. Virtual documents can be *remote documents*, defined as arbitrary WebOQL queries, or *views*, defined as queries over documents registered in the system. The system catalog contains metadata for the documents, especially their schemata, used for query processing and optimization. Queries can range over both the catalog and the documents, and multiple query languages are supported. In this paper we describe the architecture and main of ToX; we present our indexing and storage strategies, including two novel techniques; and we discuss our query processing strategy. The project started recently and is under active development.

1 Introduction

XML was originally designed for allowing the representation of user-defined structure in Web documents. However, it turned out to be also an elegant textual representation for semistructured data [1]. As a consequence, it blurs the notions of data and document and enables the joining of efforts from the Web and database management communities. Many XML related projects have been conducted, both in academia and in industry: data models and query languages been defined; tools and applications, mostly open source code, have been developed; and a considerable standardization effort has been done. As a result, XML is becoming *the* standard for representing information on the Web. Nevertheless, this is still a very active field. Given the current size and growth tendency of the Web, efficient storage, indexing and query processing of XML documents is of paramount importance.

In this paper we introduce ToX, a repository for XML data and metadata. As such, ToX storage units are documents[1], which are registered, modified and eventually removed from the system. For each document we typically store its *data* (i.e., its contents) and a set of related *metadata*. The latter includes standard metadata (e.g., [7]), but we are particularly interested in storing the schemata of the documents in the system. A schema defines a *document type*; we assume that there will be more documents than types in the system. We maintain a catalog containing entries for all documents and types registered in the system. ToX supports heterogeneous data storage and indexing as well as multiple XML query languages. Virtual documents, for which only metadata is stored, are defined as queries over either remote data sources or documents in the system. In this setting, ToX can be used as a data integration environment, where the data sources correspond to XML documents.

[1] For clarity, hereafter we refer to *document* simply as an XML document, regardless of its contents.

[†]This work is supported by the National Science and Engineering Research Council of Canada, Bell University Laboratories, and the IBM Centre for Advanced Studies.

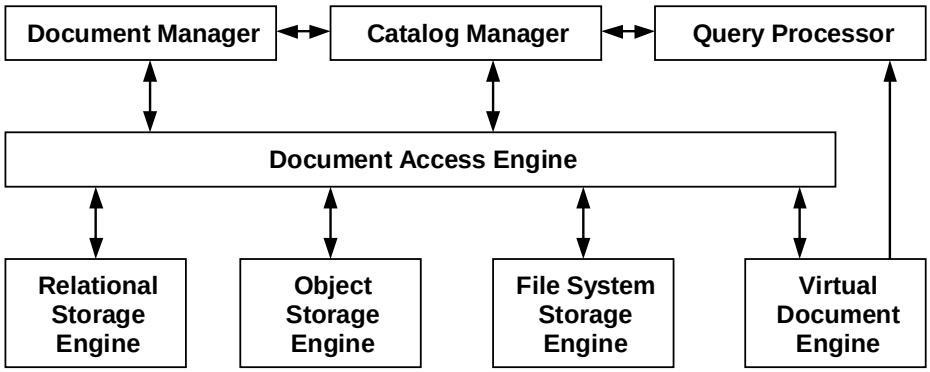

Figure 1 – ToX architecture.

The remainder of this document is organized as follows. In Section 2, we present the notion of document *structuredness*, a key aspect for the efficient storage of XML documents. ToX architecture and its main components are presented in Section 3. In Section 4 we briefly discuss the automatic characterization of documents, one of our main goals in this project. Section 5 presents EXIP, an executive information portal, which is our first data intensive application for ToX. Finally, Section 6 draws some conclusions.

2 Document *structuredness*

Although XML can represent both *textual documents* (e.g., books) and *data* (e.g., a catalog of books), documents in these categories differ greatly, not only in their characteristics but also in the way they are indexed and queried. Consider an XML representation of a book. Most elements are strings with varying length (e.g., a few words for the title and many sentences for the paragraphs). Typical queries involve searching for words occurring in a given context (Structured Information Retrieval [9]), thus requiring a full-text index. Other typical query is to retrieve a large sequential portion of the document (e.g., a chapter of the book). A book catalog, on the other hand, is expected to have many entries with a somewhat similar, record-like structure involving different data types. Typical queries in this case access many fragments of the data (e.g., aggregate queries) and yield small result sets (e.g., determining the lowest price among all Database books). For this scenario, traditional DBMS indexing techniques are more adequate.

Given these differences, characterizing the documents is crucial for determining the best storage and indexing strategy. We use the document *structuredness*, a fuzzy measure that determines how close a document is to each extreme of this spectrum. We say a document has low structuredness if it is similar to a textual document, for storing and indexing purposes. Documents of the same type are expected to have equivalent structuredness. We also need to determine a threshold value for deciding which storage and indexing approach to use for a given document. Note, however, that structuredness is not the same as regularity in the structure (i.e., a document might have low structuredness and a regular structure).

3 ToX architecture

The ToX architecture is depicted in Figure 1. The modules visible to the user are the Document Manager and the Query Processor. The former allows the user to register, check-in, check-out and remove documents in the repository. This module is also responsible for determining the storage and indexing strategy for incoming documents. The core of the system is the Document Access Engine, which mediates the requests of the other modules and the actual data access engines. We next describe the other ToX components.

3.1 The catalog manager

This module is responsible primarily for maintaining the metadata associated with the documents registered in the system. We intend to store typical metadata (e.g., [7]) and also the schemata of the documents. There are many schema formalisms for XML [16], but initially we intend to support DTDs, given their popularity, and XML Schema. The latter is preferred since it allows the representation of more detailed information about the structural properties of the documents, which can improve the storage, indexing and query processing in the system. Note that structural queries can be answered by accessing the catalog alone, provided that every document registered in the system has a schema associated with it. The catalog itself is an XML document and as such can be queried by the user.

Other functions of the Catalog Manager include maintaining statistics on the use of the documents and a version control mechanism. One important problem here is how to represent and manage multiple versions of XML documents. We also intend to extend this module with schema integration support, for allowing reasoning about schemata in the repository.

All ToX modules depend heavily on the catalog for performing their functions. For this reason we define a set of *system queries*, using one of the languages supported by ToX. One of our goals in this project is to identify the most useful set of system queries and an efficient way of implementing them.

3.2 Storage and indexing

Current storage methods for XML documents fall into three broad categories: files, DBMS mappings and native storage engines. Intuitively, files are more suited for textual documents, since they allow fast sequential access while DBMS mappings are better for documents with high structuredness. Native storage engines (e.g., [17]) are starting to flourish and their performance might depend on the document structuredness as well[2]. Although differing in many aspects, each of these methods relies on a single storage mechanism, which may not be optimal for all types of documents. ToX, on the other hand, uses different mechanisms, thus allowing more flexibility in the storage of documents. Next, we briefly describe the storage engines supported by ToX (Figure 1).

The Relational Storage Engine maps the contents of the documents into relational databases; traditional RDBMS indexing is used in this module. ToX will support a generic mapping [11], which is insensitive to the structure of the document and, thus, can be used even for documents whose schema is unknown; structure dependent mappings (e.g., STORED [8]) can be used as well. DB2 is our relational storage engine; we also intend to use its own storage mechanisms for XML content [13] also. A generic engine, in which all data accesses will be done via ODBC for enabling the use of any RDBMS, will also be provided.

The Object Storage Engine can be used in two ways: for defining a mapping and for storing a DOM tree corresponding to the XML document. We use Ozone [18], an open source persistent storage engine for Java objects, which has already some support for storing XML documents. Currently, the indexing mechanisms available in Ozone are those provided by the Java language (e.g., hash tables). The third storage strategy allowed in ToX is to keep the documents as files, and use a text database system for indexing and querying these files. We intend to use MultiText [6] for this purpose.

Hybrid storage. In Section 2 we discussed two types of XML documents: text and data. Note, however that it is customary to have "hybrid" documents exhibiting characteristics of both categories. An example of a hybrid document is a catalog of books, containing data about the books annotated with comments from readers, editor or publishers, as usually found in most online bookshops.

[2] In fact, another goal of this project is to characterize the performance of different storage engines w.r.t. different types of XML documents (see Section 4).

A hybrid document can be viewed as a join of a *data component* and a *set of textual components*. In our example, the data component would contain the data about the books and each textual component would be a single comment about a book. The join attribute can be any unique element in the data component[3] (e.g., ISBN in our example) or a system generated value; we index both data sources on the join attribute, for efficiency reasons. In this scenario, two new document types are registered in the catalog, corresponding to the documents that are actually stored, while the original is represented as a virtual document. This physical storage design decision can be hidden from the user by differentiating among the corresponding document types in the catalog.

ToXin – the ToX indexing mechanism. ToXin is a main-memory indexing mechanism inspired by DataGuides [12] and Access Support Relations [14]. It was designed for allowing fast access to elements in the XML tree both for forward and backward navigation. It consists of two separate indices: a *path index*, for allowing the efficient evaluation of regular path expressions, and a *value index*, used for efficiently locating nodes in the document that satisfy certain criteria. Thus, ToXin allows a more flexible query processing. Another interesting feature of ToXin is that, unlike DataGuides, the total size of the index is always linear with respect to the size of the document. Preliminary experiments indicate promising results [19].

In its current implementation, the index is built from a DOM tree, which is kept in memory for actual references to the document. We are currently working on eliminating this dependency on the DOM structure and on providing persistence to the indices. As a result, ToXin will be used as an efficient native storage method for XML content.

3.3 Query processing

The query processor is primarily responsible for providing an interface for the issuing of user queries over the documents in the repository. This module is also used for processing the system queries (see Section 3.1). Different XML query languages are allowed: currently, we are integrating ToX with Kweelt [20], Xpath [5] and XSLT [4].

Note that different data storage mechanisms have different data representation and access patterns. We adopt an algebraic approach for query processing and optimization in ToX, thus hiding this heterogeneity and reducing the complexity of the query processor. All that is needed is that each data storage engine supports this algebra, which also simplifies the addition of new storage engines. We intend to use the W3C XML Algebra [10].

3.4 Virtual documents

ToX was designed to deal transparently with both real and virtual documents. We distinguish two kinds of virtual documents: *remote* documents, which the user is unable or does not care to store, and *views*, which are queries ranging over documents registered in the catalog. A remote document is specified by an arbitrary WebOQL [3] query; views are specified in any language supported by ToX.

Note that as far as the user is concerned, ToX does not distinguish between real and virtual documents. However, optimizing queries that access remote documents is a process that depends on the access patterns of the remote data source. Efficient view materialization and maintenance are also important issues here.

4 Automatic document characterization

As we have seen in the previous sections, the key aspect for the storage, indexing and querying of a document in ToX is its structuredness. Therefore, one of our goals in this project is to establish an algorithm for computing it. Intuitively, this measure should be a weighted average of factors including characteristics of the document and its usage in a

[3] Note that uniqueness constraints can be expressed by a DTD or an XML Schema specification.

```
<tox-file name="books.xml">
  <element name="books">
    <element name="book" tox-minqtty="1000"
     tox-maxqtty="1000">
      <tox-lookup list="book_list"
        method="sequential">
        <element name="isbn" type="isbn_type">
          <tox-lookup-item path="book/isbn"/>
        </element>
        <element name="title" type="title">
          <tox-lookup-item path="book/title"/>
        </element>
      </tox-lookup>
      <element name="price">
        <complextype>
          <attribute name = "currency"
           type="string">
            <tox-value>CDN</tox-value>
          </attribute>
          <tox-number type="real" minvalue="10"
           maxvalue="100" decimals="2">
        </complextype>
      </element>
    </element>
  </element>
</tox-file>
```

```
<tox-file name="reviews.xml">
  <element name="reviews">
    <element name="review" tox-minqtty="500"
     tox-maxqtty="2000">
      <tox-lookup list="book_list"
        method="random">
        <element name="isbn" type="isbn_type">
          <tox-lookup-item path="book/isbn"/>
        </element>
      </tox-lookup>
      <element name="comments">
        <complextype>
          <tox-alternatives>
            <tox-option odds="75">
              <attribute name="type" type="string">
                <tox-value>reader</tox-value>
              </attribute>
              <tox-string type="paragraph"
                tox-minqtty="1" tox-maxqtty="5"/>
              …
            </tox-option>
            <tox-option odds="25">
            …
</tox-file>
```

Figure 2: ToXgene template definition of two XML documents.

typical query mix. However, one could also take into account the structuredness of similar documents (e.g., with a similar type) in the system. We plan to investigate the use of schema mapping techniques for inferring such similarities.

Another point worth mentioning here is that we need a more realistic classification of XML documents. In Section 2 we discussed textual and data documents, which seem to be the most common. However, XML has also been used for representing Java code and UML diagrams, for instance. Since one can expect a higher frequency of recursive elements in these classes of documents, it might be the case that none of the storage strategies discussed here yield optimal performance. In summary, properly characterizing XML documents is an extremely relevant task.

4.1 ToXgene – generating synthetic XML content

ToXgene is a generator of synthetic XML content based on templates. Each template consists of XML Schema specifications annotated with the probabilities of occurrence and characteristics of each element (or attribute) in the document. Our goal with ToXgene is to be able to generate large collections of documents for evaluating the storage, indexing and query processing mechanisms in ToX. Having total control over the structure of the benchmarking documents is a requirement in this case.

One of our motivations in designing ToXgene was the necessity of generating large collections of documents sharing both structure and values for some elements (i.e., documents that can be joined). Figure 2 shows a fragment of a template defining two XML documents: `books.xml`, representing a book catalog with 1000 entries, and `reviews.xml`, containing at least 500 and at most 2000 book reviews. In order to be able to join these documents, we have to make sure that all `ISBN` values in the reviews appear in some entry in the catalog. This is accomplished in ToXgene by keeping all ISBN values in a `tox-list` (not shown in the figure), which are later referenced in the actual XML documents; the ISBN element is declared to be unique in that list. Note that in our example, the books for which a review is written are chosen randomly, reflecting the fact that some books being more popular will get

more reviews[4]; it is also specified that 75% of the reviews are written by readers (as opposed to publishers, for instance). The values of the elements can be generated by ToXgene (e.g., a `tox-string`) or specified in the template (e.g., CDN as the currency for the prices).

5 EXIP – one motivating application for ToX

Strategic business analysts keep track of trends that are relevant to their organization and its strategic objectives. To accomplish their mission, they monitor news stories and other reports as they become available, looking for evidence that these objectives remain on track, or have encountered obstacles. The Executive Information Portal (EXIP) is a prototype intended to support this type of knowledge work. Because EXIP has to handle news stories and reports, it is designed to manage thousands of documents that have multiple links among them. EXIP supports complex search operations on these documents, involving both full-text and metadata search. Documents are indexed according to a multidimensional index with respect to the semantic model. Last but not least, the system supports online updates. In order to address all these requirements the documents are internally stored as XML documents. The first release of the document management component of EXIP was build using a relational DBMS. Currently we are in the process of porting this component to ToX.

EXIP is a good test bed for the ToX engine. One of the major issues that ToX has to address is querying large collections of XML documents. Although there are quite a few implementations of XML query languages, not many of them handle query inputs of thousands of documents. Furthermore, EXIP's query requirements range from very complex XML queries to very simple ones. To accommodate these requirements, EXIP employs different XML query languages for different tasks. For example, Kweelt is used for complex queries while our own lightweight XML query implementations are used for simple ones.

6 Conclusions

In this paper we introduced ToX, a heterogeneous repository for XML content that can be used for data integration. We discussed its architecture, and presented some of its features; namely, ToXin, ToXgene and a hybrid storage scheme. The main goal of this project is to understand how the characteristics of a document affect its storage and query processing. We expect to define ways of characterizing XML documents; define guidelines for their storage and indexing; design efficient native storage mechanisms; and, finally, better understand the factors that affect the performance of the various storage approaches.

There are already a number of methods for storing and querying XML documents, using files (e.g., Kweelt [20]); schema independent mappings (e.g., Edge Tables [11]); schema dependent mappings (e.g., [21]); hybrid mappings (e.g., STORED [8] and Ozone[5] [15]); and native storage mechanisms (e.g., Lore [17] and Xyleme [2]). ToX differs from all of them in the following ways: it allows heterogeneous data storage and indexing; it supports multiple query languages; it can be used as a data integration environment; and, most importantly, it uses the characteristics of the documents as the criterion for determining the way they are stored and indexed. Unlike query processors (e.g., Kweelt), ToX provides persistent storage for indices and other query optimization data. Note that although STORED and Ozone offer hybrid mappings, they rely on a single storage engine (a RDBMS and an ODBMS, respectively). Xyleme is intended to be a warehouse for all XML data on the Web. As such, it deals with the crawling and refreshing of documents, which are irrelevant in our context.

ToX is a recent project and is under active development.

[4] One can define different probability distributions when choosing items from the lists.
[5] This method should not be confused with [18].

References

[1] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web*. Morgan Kaufman, 1999.

[2] Vincent Aguilera, Sophie Cluet, Pierangelo Veltri, Dan Vodislav, and Fanny Wattez. Querying XML documents in Xyleme. In *Proceedings of the ACM SIGIR 2000 Workshop on XML and Information Retrieval*, Athens, Greece, July 2000.

[3] Gustavo Arocena, and Alberto Mendelzon. WebOQL: Restructuring documents, databases, and Webs. In *Proceedings of the Fourteenth International Conference on Data Engineering*, pages 24-33, Orlando, Fl, USA, February 1998.

[4] James Clark, editor. XSL transformations (XSLT) Version 1.0 - W3C recommendation, November 16 1999. http://www.w3.org/TR/xslt.html.

[5] James Clark, and Steve DeRose, editors. XML path language (XPath). W3C recommendation, November 16 1999. http://www.w3.org/TR/xpath.

[6] Gordon V. Cormack, Charles L. A. Clarke, Christopher R. Palmer, and Robert C. Good. The MultiText retrieval system. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, page 334, Berkeley, CA USA, August 1999.

[7] The Dublin Core Metadata Initiative. http://purl.org/dc/.

[8] Alin Deutsch, Mary F. Fernandez, and Dan Suciu: Storing semistructured data with STORED. In P*roceedings ACM SIGMOD International Conference on Management of Data*, pages 431-442, Philadephia, PA USA, June 1999.

[9] Daniel Egnor, and Robert Lord. Structured Information Retrieval. In *Proceedings of the ACM SIGIR 2000 workshop on XML and Information Retrieval*, Athens, Greece, July 2000.

[10] Peter Fankhauser, Mary Fernández, Ashok Malhotra, Michael Rys, Jérôme Siméon, and Philip Wadler, editors. The XML Query Algebra - W3C working draft, December 4 2000. http://www.w3.org/TR/query-algebra/.

[11] Daniela Florescu, and Donald Kossman. Storing and querying XML data using an RDMBS. *IEEE Data Engineering Bulletin* 22(3):27-34, September1999.

[12] Roy Goldman, and Jennifer Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 436-445, Athens, Greece, August 1997.

[13] IBM DB2 Universal Database XML Extender – Administration and programming. http://www-4.ibm.com/software/data/db2/extenders/xmlext/, 1999.

[14] Alfons Kemper, and Guido Moerkotte. Advanced query processing in object bases using access support relations. In *Proceedings of the 17th International Conference on Very Large Databases*, pages 294-305, Brisbane, Australia, August 1990.

[15] Tirthankar Lahiri, Serge Abiteboul, and Jennifer Widom. Ozone: Integrating structured and semistructured data. In *Proceedings of The 8th International Workshop on Database Programming Languages*, Kinloch Rannoch, Scotland, September 1999.

[16] Dongwon Lee, and Wesley Chu. Comparative analysis of six XML schema languages. *SIGMOD Record*, 29(3):76-87, September 2000.

[17] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass, and Jennifer Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54-66, September 1997.

[18] Ozone: The open source Java ODBMS. http://www.ozone-db.org/.

[19] Flavio Rizzolo. ToXin: an indexing scheme for XML data. Master Thesis, Department of Computer Science, University of Toronto. January 2001.

[20] Arnaud Sahuguet. Kweelt – Querying XML in the new millennium. http://db.cis.upenn.edu/Kweelt, September 2000.

[21] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David DeWitt, and Jeffrey Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *Proceedings of 25th International Conference on Very Large Data Bases*, pages 302-314, Edinburgh, Scotland, UK, September 1999.