Data Management for XML: Research Directions

Jennifer Widom
Stanford University
widom@db.stanford.edu, http://www-db.stanford.edu/˜widom

Abstract

This paper is a July 1999 snapshot of a "whitepaper" that I've been working on. The purpose of the whitepaper, which I initially drafted in April 1999, was to formulate and put into prose my thoughts on the research opportunities XML brings to the general area of data management. It is important to know that **this paper is not a survey**. *It offers my personal opinions and thoughts on Data Management for XML, fully incorporating my biases and ignorances. Related work is not discussed, and references are not provided with the exception of a handful of URLs. Furthermore, I expect the whitepaper to evolve over time; please see [1] for the latest version.*

1 The XML Revolution

XML—the *eXtensible Markup Language*—has recently emerged as a new standard for data representation and exchange on the Internet [2]. The basic ideas underlying XML are very simple: tags on data elements identify the meaning of the data, rather than, e.g., specifying how the data should be formatted (as in HTML), and relationships between data elements are provided via simple nesting and references. Yet the potential impact is significant: Web servers and applications encoding their data in XML can quickly make their information available in a simple and usable format, and such information providers can interoperate easily. Information content is separated from information rendering, making it easy to provide multiple views of the same data. (XML data files can be rendered via specifications in *XSL*, the *eXtensible Stylesheet Language* [3].) Laborious, error-prone, and unmaintainable "screen-scraping" as a method for extracting useful data from HTML Web pages is greatly reduced, since XML is designed for data representation—XML is simple, easily parsed, and self-describing.

As an example, consider the following HTML fragment (extracted from my own publications Web page [4] without modification), describing two publications:

```
<UL>
<LI>
R. Goldman, J. McHugh, and J. Widom.
<A href="ftp://db.stanford.edu/pub/papers/xml.ps">
From Semistructured Data to XML: Migrating the Lore Data Model
and Query Language
```


Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

```
</A>.
Proceedings of the 2nd International Workshop on the Web and Databases
(WebDB '99), pages 25-30, Philadelphia, Pennsylvania, June 1999.
<LI>
T. Lahiri, S. Abiteboul, and J. Widom.
<A href="ftp://db.stanford.edu/pub/papers/ozone.ps">
Ozone: Integrating Structured and Semistructured Data
</A>.
Technical Report, Stanford Database Group, October 1998.
</UL>
```

One way of encoding the same information in XML is:

```
<Publication URL="ftp://db.stanford.edu/pub/papers/xml.ps" Authors="RG JM JW">
   <Title>From Semistructured Data to XML: Migrating the Lore Data Model
       and Query Language</Title>
   <Published>Proceedings of the 2nd International Workshop on the Web
       and Databases (WebDB '99)</Published>
   <Pages>25-30</Pages>
   <Location>
      <City>Philadelphia</City>
      <State>Pennsylvania</State>
   </Location>
   <Date>
      <Month>June</Month>
      <Year>1999</Year>
   </Date>
</Publication>
<Publication URL="ftp://db.stanford.edu/pub/papers/ozone.ps" Authors="TL SA JW">
   <Title>Ozone: Integrating Structured and Semistructured Data</Title>
   <Published>Technical Report</Published>
   <Institution>Stanford University Database Group</Institution>
   <Date>
      <Month>October</Month>
      <Year>1998</Year>
   </Date>
</Publication>
<Author ID="SA">S. Abiteboul</Author>
<Author ID="RG">R. Goldman</Author>
<Author ID="TL">T. Lahiri</Author>
<Author ID="JM">J. McHugh</Author>
<Author ID="JW">J. Widom</Author>
```

Clearly the XML encoding, although more verbose, provides the information in a far more convenient and usable format from a data management perspective. Furthermore, the XML data can be transformed and rendered as desired using simple XSL specifications.

There is great excitement in industry over XML. True believers think XML will radically change the face and uses of the Web. Leading software vendors are committed to XML and are quickly moving towards using XML internally as well as creating XML-oriented tools and products. XML technology startups are proliferating. Commercial enterprises and scientists alike are generating their data in XML, and we expect that the amount and variety of data made available in XML form, and the tools to accompany that data, will grow rapidly.

2 Commercial Perspective (A Disclaimer)

The remainder of this paper is written from a true research standpoint. For better or worse, I've ignored or cast aside certain important considerations from a commercial perspective. For example:

- Although it is clear that XML will have a significant impact on Internet information management, it is still unclear precisely how XML will be used. Will XML be used primarily as a data *exchange* format, or will it also be used as a data *storage* format? Will most XML documents be governed by *Document Type Definitions* (*DTDs*) or will many be without? For that matter, how important will the concept of an XML "document" be? It may be years before the answers emerge. Nevertheless, we researchers can enjoy the freedom of attacking problems associated with any or all of the possible outcomes. As illustrated in the next section, I believe database-style technology applied to XML can play an important role in all of them.

- Despite all of the hype, including my own, XML will not be the proverbial magic bullet that solves all of the problems associated with application interoperation and data integration on the Internet. Some applications, even if they encode their data in XML, may not wish to expose it that way, so "screen-scraping" will not disappear entirely. (In fact, some information providers purposely make their current HTML pages difficult to parse, in order to protect the information from being copied—these providers certainly won't expose their data in XML even if they store it that way.) Furthermore, there is the very significant issue of ensuring that applications use commonly-understood tags for data, or at least have practical methods and tools for detecting and resolving discrepancies among tags. Fortunately, in many cases Web information providers *will* have a vested interest in cooperating with each other, so we anticipate that such methods and tools, as well as agreed-upon, domain-specific DTDs, will proliferate.

- This paper is based on the core XML specification. It does not consider the various proposed mechanisms for inter-document references (e.g., *XLink*, *XPointer*), although it's my feeling that the differences among these mechanisms will have little significant impact on database-oriented XML research. The various proposed extensions or alternatives to DTDs for richer schema definitions (e.g., *DCDs*, *XML-Schema*, and others) also are not discussed here. These schemes obviously can have an impact on database-oriented XML research (and database researchers probably should try to have an impact on these schemes), but the field is too crowded at this point to choose a particular winner. Also, I believe that most of the technical issues discussed below with respect to DTDs are equally valid for richer schema definition languages.

3 Database Research Opportunities

In addition to the promise of greatly facilitating information integration on the Internet, as discussed in Section 1, another exciting promise of XML is that it "turns the Web into a database." Migrating Web information to XML is a significant first step in enabling efficient execution of ad-hoc, expressive queries over large amounts of Web data—a core feature of traditional database management systems. Consider the current state of query processing over information on the Web:

- Data embedded within HTML pages needs to be preprocessed by special-purpose, page-specific parsers before meaningful queries can be posed, a limited technology at best. Otherwise, ad-hoc queries are limited to simple keyword-based searches (as provided by search engines, for example) that understand documents as streams of words and little more.

- Data stored within traditional database management systems generally is accessed on the Web only through simple and rigid forms-based interfaces.

Most of us are familiar with Web sites that contain vast databases of useful information, but whose query and search facilities are surprisingly primitive. As a specific query scenario, consider a large collection of publication information such as that in the examples of Section 1, drawn from one or more data sources.

- If the information is provided in HTML, we could attempt to parse out the relevant data elements, provided the formats don't change and our parsers are amenable to the inevitable inconsistencies and omissions across publications. Otherwise, we are limited to keyword-based searches.
- If the information is provided via a traditional DBMS, then simple, fixed, parameterized queries are the usual mode of external access. Query capabilities aside, much of the information made available on the Web is not well-structured (even our tiny examples in Section 1 illustrate some of the problems), and thus the data may not be amenable to using a traditional DBMS.
- Suppose the information is provided in XML, and let us be optimistic and assume that if multiple data sources are involved, the XML encodings are compatible (see Section 2). In this case, the structure and "meaning" of the data (at least to the extent that meaning can be embodied in tags), as well as the data itself, is readily parsable and available, setting the stage for powerful queries. Some relatively simple examples of such queries are:
  - *Find all authors with two or more SIGMOD publications in the same year.*
  - *Find the earliest publication with "semistructured data" in its title.*

Encoding information in XML is a first step to enabling expressive, database-like queries over the information, but many query processing issues still need to be addressed, as discussed in the bullets below. Furthermore, the tendency to mix traditional data elements with free text in XML, the ability to encode data ranging from fully structured to highly unstructured, and the inherent dichotomy between documents and databases, poses new challenges in combining techniques from database systems and information retrieval.

A few sample research topics for the database community follow, with a primary focus on database-like treatment of XML (as opposed to focusing on XML-based information integration, clearly a very important topic as well). There has been preliminary work in several of these areas, while some topics are still virtually untouched.

- Since XML is a document format and not a data model, we need the ability to map XML-encoded information into a true data model.
- More generally, we need to resolve the various conflicts that arise when we try to mix the concepts of documents and databases. For example, while some applications may wish to view a large set of XML documents as exactly that—a set of documents—other applications may prefer to think of each document as a database "load file," where all document contents are merged into a single large database. In fact, we may wish to simultaneously view a body of XML information in both ways.
- Theoretical results and practical techniques for designing XML databases are needed, to the extent possible within the relatively free-form nature of XML. For example, when should attributes be used and when should subelements be used? Is a one-to-one relationship best represented using element nesting or IDREFs? Is there an analogy to relational functional dependencies in the XML world? There are a number of questions of this nature that arise when translating a conceptual model of a database into an XML encoding.
- The relationship between XML's optional Document Type Definitions (DTDs) and traditional database schemas needs to be understood and exploited.

- An appropriate query language (or set of languages) for XML needs to be defined. This task is made particularly difficult because the true requirements for XML query languages will not be known until a significant number of data-intensive XML applications are built. Here too, there is an inherent conflict between the document and database view of XML-encoded information, and thus an opportunity to merge formerly separate technologies.

- Database updates in an XML setting must be considered, with a focus on environments that are heavily read-oriented.

- Efficient physical layout and indexing mechanisms are required for large stores of XML data. At the same time, we should be able to provide the illusion of an XML data store when the data actually is stored elsewhere (such as in a traditional DBMS), and make the two modes work together.

- All facets of traditional query processing must be considered, from semantic checking (when appropriate) through plan generation and optimization to efficient access methods. We also need to consider non-traditional query processing that can meaningfully and efficiently handle a mixture of data elements (both structured and semistructured) and free text.

- *View* mechanisms are important in conventional databases, and are likely to be important in XML databases as well. Virtual views, involving query rewriting techniques, are likely to be quite a bit more complex in XML than in the relational world. Similarly, the incremental maintenance problem for materialized views is likely to pose new problems when we consider XML data. Even the view definition language itself needs to be considered carefully. At one extreme, *XSL* [3] could be used as a view definition language, posing especially challenging view management problems due to its expressive power and procedural nature.

- Everything needs to scale to Web proportions (!).

4 The Lore Project at Stanford

The *Lore* project at Stanford [5] began around 1995, with the premise of building a complete database management system for *semistructured data*. We defined semistructured data as data that may be irregular or incomplete, and whose structure may change rapidly and unpredictably. Information integrated from heterogeneous sources, structured text, and "screen-scraped" HTML pages were our original motivating sources of data for Lore.

By 1998 we had largely achieved our goal. We had built a complete, robust (as university prototypes go), multi-user database system based on a fairly traditional DBMS architecture, with a number of extra features such as *dynamic structural summaries* (*DataGuides*), *keyword and proximity search*, and an *external data manager*. Our schema-less, self-describing data model, called the *Object Exchange Model* (*OEM*), was essentially a directed labeled graph—or equivalently, nested tagged data with references. Our query language, *Lorel*, was based on OQL, with modifications and extensions suitable for semistructured data. Many aspects of building a DBMS top-to-bottom needed to be revisited in the context of semistructured data. We published papers, distributed our system, and continued to work on a variety of issues.

When XML came along, the similarity between OEM and XML was striking, and exciting. In late 1998 and early 1999 we migrated Lore to be based on a true XML-oriented data model and modified our query language accordingly. (See our short paper [6] on the topic, which also happens to be one of the publication examples in Section 1.) A number of subtleties were involved in both design and implementation, and the first public release of the XML version of Lore was made in May 1999.

While some aspects of Lore could still benefit from refitting and tuning for XML, by and large we have one of the only complete database systems designed specifically for storing and querying "native" XML. We believe that Lore provides an ideal testbed for further research in this area.

5 Personal Research Agenda

As outlined in Section 3, there is a broad range of research issues to be explored in XML data management. Here I list a number of more specific topics that are on my personal research agenda. In some cases the topics are obvious follow-ons to previous work in Lore, in other cases the topics are just plain interesting.

5.1 Storage and Indexing

Lore uses a simple storage manager that parses XML into the basic units of elements, attributes, and text strings, and stores them using a straightforward depth-first clustering heuristic. We can build a wide variety of indexes on Lore databases. The types of indexes we support were designed for Lore's original data model, OEM, but with minor changes the indexes also work for XML. There are several further avenues to pursue in storage and indexing:

- Different clustering schemes for storing XML data, preferably customizable for different databases and applications.
- New index types designed specifically for XML data, taking into account element ordering, new kinds of comparison operations, and a few other subtle differences between XML and our original data model.
- More radically, we plan to explore using XML documents themselves as a storage medium in Lore, augmented with auxiliary structures such as: (1) an indexing scheme for quickly finding certain elements, attributes, and more complex structural patterns in the document; and/or (2) a data store containing some of the XML data parsed to some level (and possibly created "lazily", i.e., on an as-needed basis), with a mapping maintained between the database and documents.
- Depending on how applications tend to encode their data in XML, we may find that we can take data that is primarily encoded as tree structures and convert it to a more graph-structured representation, e.g., by merging identical text values or entire subelements, then inserting appropriate IDREFs. It will be interesting to see whether any significant compression is achieved, but more importantly to observe the effect on query processing and even query semantics.
- Another opportunity for compressing XML data is to exploit regularity in the structure, in the extreme case storing the XML data essentially as relations. (See also Sections 5.5 and 5.6 below.)

5.2 DataGuides and DTDs

Lore builds and dynamically maintains a *DataGuide* for every database, which is a summary of the current structure of the database and serves some of the functions a schema serves in a traditional DBMS. Since an XML DTD (Document Type Definition) is a set of grammar rules that restrict the form of an XML document, there is a close relationship between DataGuides and DTDs. Note that a DTD acts more as a traditional schema, since it restricts the allowable XML data, while a DataGuide infers rather than imposes structure.

Currently we can store DTDs in Lore databases, and we can use a DTD to build an "approximate" DataGuide. There can be a significant performance advantage to building DataGuides from DTDs instead of from the database itself: in some cases, particularly in highly connected and cyclic databases, building a DataGuide can be prohibitively expensive. However, the DTD does not capture the structure of a database as accurately as a DataGuide: attributes and elements included in a DTD need not actually appear in the database, and DTDs cannot specify restrictions on the types of elements referenced by IDREF attributes. Furthermore, the lack of accuracy in DTDs inhibits other ways we use DataGuides in Lore, such as storing statistics and encoding path indexes.

There are several avenues to pursue in the general area of DTDs, DataGuides, and their relationship, listed here in no particular order:

- Validating parsers can check that an XML document conforms to its DTD, and we can build the same functionality into an XML database system. However, there are some interesting related problems: Can we perform validation incrementally as portions of an XML database are updated? Can we perform validation on the update statements themselves, instead of on the database? When we pose a query to an XML database, can we infer a DTD for the query result?

- Currently we do not encode subelement ordering in our DataGuides (even the DataGuides we build from DTDs). If subelements of a given element type always appear in the same order, it would be nice to reflect that ordering in the DataGuide. However, if subelements do not always appear in the same order, we probably do not want to expand the size of the DataGuide to encode that fact. One possibility is to order subelements in the DataGuide based on a "most frequent" ordering from the database.

- We would like to further investigate the performance and functionality tradeoffs between using (exact) DataGuides inferred from the database versus (possibly inexact) DTDs.

- There may be scenarios where DTDs are available for specific portions of an XML database, but not for all of the data. In that case, for maximum flexibility we might want to build a DataGuide for the portion without a DTD, and link to DTDs in the appropriate places.

- Our DataGuide serves as the basis of a convenient interface by which users can browse the current structure of the database and even formulate queries "by example." Independent of XML, for some time we've had the idea of trying to blur the distinction between browsing structure (the DataGuide) and data (the database). We envision interactions where predicates are specified within the DataGuide, which produces a smaller DataGuide based on the constrained database, through which additional predicates may be specified, and so on. When the constrained database becomes sufficiently small, we should automatically begin browsing the data along with the structure.

- Along similar lines, when browsing the structure of a database through the DataGuide, we might want to specify certain updates that should propagate to the database. For example, we might "cleanse" the data by modifying or merging tags, or by identifying portions of the database that are uninteresting or erroneous and can be deleted.

5.3 Databases and Information Retrieval

The typical paradigm for querying document collections is to use information retrieval style searches based on keyword matching and word position within documents. By contrast, the typical paradigm for querying databases is through an expressive, declarative query language (such as SQL) that relies on database structure.

Lore implements the declarative OQL-based query language *Lorel*, along with a *keyword and proximity search* feature. Keyword search in Lore is straightforward: we find all objects (elements or attributes) whose tag, name, or value contains the specified keyword. Proximity search is more interesting. In a document collection, a typical "*X near Y*" search might return all documents containing both *X* and *Y*, ranked by how near *X* and *Y* are to each other within the document. There are two problems with this approach:

1. If the documents are structured, as XML documents are, then nearness based on the linear encoding of the document, rather than on the document's structure, may not work well. For example, in our sample XML data of Section 1, the year of a publication is closer in a document sense to the title of the following publication in the list than it is to its own title, and author names are nowhere near the relevant references.

2. Document divisions may be artificial: items that are near each other semantically may not even appear in the same document, and the concept of separate documents may be lost when loading XML into a database.

In Lore we solve both of these problems by encoding all XML structure in the database, and by measuring proximity based on (weighted) path length in the database graph.

We're excited about how the techniques we've developed for proximity search in Lore might be used to improve searching over XML on the Web. Still, there is more work to be done on proximity search in Lore, as well as generally integrating database and information retrieval concepts:

- While we've developed some novel indexing structures and algorithms designed to make Lore's proximity search feature scale to very large databases, scaling to Web proportions is still a significant leap away.
- In Lore, keyword/proximity searches versus Lorel queries so far have been completely separate. We would like to integrate searching into Lorel queries, which (among issues) requires us to combine standard set-based query results with ranked results. A related idea is to use proximity search "under the covers" as a pre-filtering step during query processing.
- Our experience so far has been that Lore's proximity search feature yields intuitive and useful results, and that keyword and proximity search in general are a good way for casual users to interact with Lore databases. One feature that's missing, however, is the ability to explain why objects rank where they do in proximity search results. Identifying and saving the relevant "near" objects is both a technical issue during search processing (primarily an efficiency problem), as well as a user-interface issue of presenting the explanations in an intuitive fashion.
- We have begun investigating the related problem of *similarity search* in Lore. While proximity search is useful for identifying objects that are related based on closeness in a graph sense (due to shared subobjects, for example), it doesn't identify objects that are similar structurally but distant within the database structure. In similarity search, given a set of XML objects (elements and/or attributes), we want to rank them based on how similar they are to one or more other objects. Similarity of objects might be based on values, substructure (children, outgoing IDREFs), and/or super-structure (parents, incoming IDREFs). Needless to say, coming up with a workable definition of similarity measure, much less developing efficient evaluation algorithms, is a significant challenge. In addition, the issues of scalability, integration with the Lorel query language, and providing an "explain" feature to the user, described in the previous three bullets for proximity search, are relevant here as well.

5.4 Other Database Features

Three useful features provided by most traditional database management systems are *views*, *constraints*, and *triggers*. We have done some initial work on materialized views in Lore, and we also have done some work on *change management*—tracking, representing, and querying changes in semistructured databases. However, full view support for XML, including both virtual and materialized views, is a largely unexplored area. The same is true of constraints; in fact, even the notion of keys has not been introduced into XML or semistructured database work in a significant way, as far as I know. With respect to triggers (or active database capabilities in the Web/XML context in general), it is important to consider the relationship with recent developments in publish-subscribe technology.

5.5 Mixing Semistructured and Structured Data

A frequent and valid criticism of work in the area of semistructured data management is that *all* data is assumed to be semistructured in nature, i.e., (as defined earlier) the data may be irregular or incomplete, and its structure may change rapidly and unpredictably. As a result, when structure does happen to be present and stable in a semistructured database, the structure is not exploited to the advantage of the user or the system. Lore is certainly guilty on this count.

There are two issues to address: (1) finding the structure; (2) exploiting the structure. There has been some work on issue (1), although with XML the presence of a DTD certainly alleviates this problem to a great extent. The second issue—how to exploit structure when it is present, but not require structure for effective or efficient processing—is largely yet to be addressed. Structure might be exploited at all levels of the system: object layout, indexing, query processing, query formulation, etc. We have done some preliminary work in this general area by "marrying" the ODMG and OEM data models (and corresponding query languages) in a system called *Ozone*, but clearly there is more to be explored.

5.6 XML in/on a Traditional DBMS

We decided to build Lore from scratch so that we could explore every nook and cranny of building a complete DBMS for semistructured data (now XML). In retrospect, we probably should have used somebody else's low-level page or object storage manager, since we haven't done much work at that level, and writing a transaction manager was a pain. More generally, it would be interesting to explore just how much of an XML-savvy DBMS actually needs to be built from scratch, and how much can be lifted from existing DBMSs. (For example, at the other extreme, we and others have considered schemes for translating semistructured XML data to relations and translating Lorel queries to SQL accordingly.) While I expect there are some interesting research problems to explore all along the spectrum, it's also a certainty that database companies are working fast and furious to figure out how XML data and queries can fit into their systems.

5.7 Performance Evaluation

We have performed some preliminary evaluations of Lore's performance, particularly in the context of query optimization. However, we have had great difficulty figuring out what an appropriate benchmark for XML data should look like, in terms of the data itself (e.g., regular versus irregular, tree versus DAG versus general graph) as well as the type of queries and mix of queries and updates. So far we have not found any large, semistructured XML data sets to work with other than those we have constructed ourselves, although this dearth of interesting XML data surely will change. Meanwhile, flexible synthetic data generation for graph-structured data is a hard and interesting problem.

6 More Information and Related Work

Information related to many of the topics discussed here, ranging from how to download Lore to research papers to links to commercial product sites, can be found by starting at the Lore project's home page [5]. There's also a lot of good stuff to be found from the W3C's QL '98 Web site [7]. I have not attempted to cover the very interesting and fast-growing body of great work from other researchers in XML and databases—I hope I haven't stepped on anyone's toes.

Alon Levy has created an interesting follow-up document to this one, *More on Data Management for XML* [8], covering two important topics omitted here: the role of XML in data integration, and new measures of complexity and scalability for XML query processing.

Acknowledgements

Many people have had a direct influence on my thinking and work on XML, including but not limited to (in alphabetical order): Serge Abiteboul, Adam Bosworth, Roy Goldman, Jason McHugh, Michael Rys, and Frank Tompa. For comments on this paper, thanks go to Roy Goldman, Alon Levy, Jason McHugh, and Dan Suciu.

Funding for my research in semistructured data has come in the past from DARPA and the Air Force Rome Laboratories, and currently is supported by NASA and by the National Science Foundation under grant IIS-9811947.

References

[1] http://www-db.stanford.edu/˜widom/xml-whitepaper.html

[2] http://www.w3.org/XML

[3] http://www.w3.org/TR/WD-xsl

[4] http://www-db.stanford.edu/˜widom/pubs.html

[5] http://www-db.stanford.edu/lore

[6] ftp://db.stanford.edu/pub/papers/xml.ps

[7] http://www.w3.org/TandS/QL/QL98/

[8] http://www.cs.washington.edu/homes/alon/widom-response.html