A dynamic warehouse for XML data of the Web

Lucie Xyleme*

Abstract

Xyleme is a dynamic warehouse for XML data of the Web supporting query evaluation, change control and data integration. We briefly present our motivations, the general architecture and some aspects of Xyleme. The project we describe here was completed at the end of 2000. A prototype has been implemented. This prototype is now being turned into a product by a start-up company also called Xyleme [14].

Xyleme: a complex tissue of wood cells, functions in conduction and storage ...

1 Introduction et motivation

The current development of the Web and the generalization of XML technology [13] provides a major opportunity which can radically change the face of the Web. We have developed a prototype of a *dynamic warehouse for XML data of the Web*, namely Xyleme. In the present paper, we briefly present our motivations, Xyleme's general architecture and its main aspects.

The Web is huge and keeps growing at a healthy pace. Most of the data is unstructured, consisting of text (essentially HTML) and images. Some is structured, mostly stored in relational databases. All this data constitutes the largest body of information accessible to any individual in the history of humanity. The lack of structure and the distribution of data seriously limit access to the information. To provide functionalities beyond full-text searching ala Alta Vista or Google, specific applications are being built that require heavy investments. A major evolution is occurring that will dramatically simplify the task of developing such applications:

XML is coming!

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Lucie Xyleme is a nickname for a large group of people who worked on the project: S. Abiteboul, V. Aguilera, S. Ailleret, B. Amann, F. Arambarri, S. Cluet, G. Cobena, G. Corona, G. Ferran, A. Galland, M. Hascoet, C-C. Kanne, B. Koechlin, D. Le Niniven, A. Marian, L. Mignet, G. Moerkotte, B. Nguyen, M. Preda, M-C. Rousset, M. Sebag, J-P. Sirot, P. Veltri, D. Vodislav, F. Watez and T. Westmann.

XML is a standard to exchange semistructured data [1] over the Web. It is widely adopted. It is clear that progressively more and more Web data will be in XML form and that DTDs, or XML Schemas will be available to describe it, see, e.g., Biztalk and Oasis. Communities (scientific, business, others) are defining their own DTDs to provide for standard representations of data in specific areas.

Given this, we decided to study and build a *dynamic Warehouse for massive volume of XML data* from the Web. The number of accessible XML pages is marginal for the moment, less that 1% of the number of HTML pages. However, we believe that this number will grow very rapidly soon. Thus we want a warehouse capable of storing huge volumes of pages and a major issue is *scalability*. The problems we address are typical warehousing problems such as change control or data integration. They acquire in the Web context a truly distinct flavor. More precisely, the research directions we studied and that we briefly discuss here are as follows:

- *efficient storage* for huge quantities of XML data (hundreds of millions of pages).
- *query processing* with indexing at the element level for such a heavy load of pages.
- *data acquisition strategies* to build the repository and keep it up-to-date.
- *change control* with services such as query subscription.
- *semantic data integration* to free users from having to deal with many specific DTDs when expressing queries.

These goals are technically very challenging. To handle such a volume of data (terabytes) and workload, we rely on parallelism. More precisely, we are distributing data and processing on a local network of Linux PCs.

Certain functionalities of Xyleme are standard and are or will soon be found in many systems. For instance, from a query viewpoint, search engines will certainly take advantage of XML structure to provide (like Xyleme) attribute-equal-value search and some will certainly support XML query languages. Other features such as change monitoring are already provided (to some extent) by systems, e.g., MindIt [10]. What is specific to Xyleme? Perhaps, the main distinguishing feature is that *Xyleme is based on warehousing*. The advantages of such an approach may best be illustrated by a couple of examples. First, consider query processing. Certain queries requiring joins over pages distributed over the Web are simply not conceivable in the current Internet context, for performance reasons. They become feasible with a warehousing approach. To see another example, consider monitoring. It is trivial to notify users when some pages of interest change. Now, if the pages are warehoused, one can support more precise alerts, e.g., when some particular elements change. More examples of new services supported by Xyleme will be encountered in the paper.

The Xyleme Project functioned as an open, loosely coupled network of researchers. Serge Abiteboul, Sophie Cluet (INRIA researchers) and F. Bancilhon (CEO of Ariso) initiated the project. Xyleme was rapidly joined by researchers from the database groups from INRIA-Rocquencourt (http://www-rocq.inria.fr/verso/), Mannheim U. (http://pi3.informatik.uni-mannheim.de/mitarbeiter.html), the CNAM-Paris (http://sikkim.cnam.fr/), the IASI Team of University of Paris-Orsay (http://www.lri.fr/iasi/introduction.fr.html), and a few individuals from other places. The project we describe here was completed at the end of 2000. A prototype has been implemented. This prototype is now being turned into a product by a start-up company also called Xyleme [14].

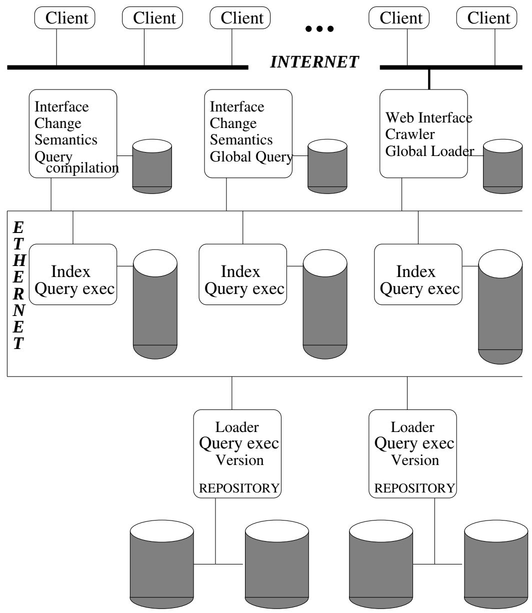

Figure 1: Architecture

The paper is a short survey. We provide references to some longer reports describing specific aspects of the work. References for related works may be found there. The paper is organized as follows. Section 2 introduces the architecture. The repository is discussed in Section 3. Section 4 deals with query processing and indexing. Data acquisition is considered in Section 5 and change control in Section 6. The topic of Section 7 is data integration.

2 The Architecture

In this section, we present the architecture of the system.

The system is functionally organized in four levels: (i) physical level (the Natix repository tailored for tree-data and an index manager); (ii) logical level (data acquisition and query processing); (iii) application level (change management and semantic data integration); (iv) interface level (interface to the web and interface with Xyleme clients).

As already mentioned, the system runs on a network of Linux PCs. All modules are written in C++ and communicate using Corba. The Xyleme clients run on standard Web browsers using Java applets. As shown in Figure 1, we (logically) distinguish between several kinds of machines. This aspect will be ignored here.

3 The Repository

In this section, we discuss the repository. More details may be found in [6].

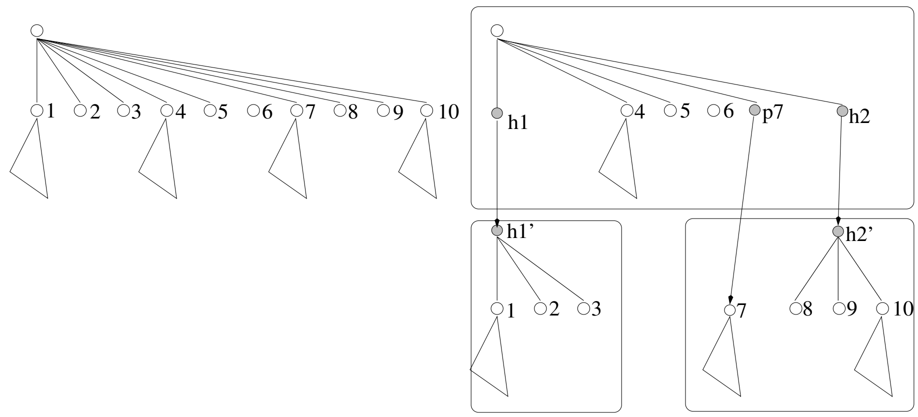

Figure 2: Distributing nodes on records

Xyleme requires the use of an efficient, update-able storage of XML data. This is an excellent match for Natix developed at the U. of Mannheim. Typically, two approaches have been used for storing such data: (i) store XML pages as byte streams or (ii) store the data/DOM tree in a conventional DBMS. The first presents the disadvantages that it privileges a certain granularity for data access and requires parsing to obtain the structure. The second artificially creates lots of tuples/objects for even medium-size documents and this together with the effect of poor clustering and the additional layers of a DBMS tend to slow down processing. Natix uses a *hybrid approach*. Data are stored as trees until a certain depth where byte streams are used. Furthermore, some minimum structure is used in the "flat" part to facilitate access with flexible levels of granularity, Also, Natix offers very flexible means of choosing the border between the object and flat worlds.

Natix is built, in a standard manner, on top of a record manager with fixed-size pages and segments (sequences of pages). Standard functionalities for record management (e.g., buffering) or fast access (e.g., indexes) are provided by Natix. One interesting aspect of the system is the splitting of XML trees into pages. An example is shown in Figure 2. Observe the use of "proxy" nodes (p-nodes) and "aggregate" nodes (h-nodes). One can control the behavior of the split by specifying at the DTD level, which splits are not allowed, are forced, or are left to the responsibility of the system.

The performances of Xyleme heavily depend on the efficiency of the repository. We believe that, by taking advantage of the particularities of the data, Natix does offer appropriate performances.

4 Query Processing

In this section, we consider processing of static queries. Change queries are studied in Section 6. A more detailed presentation of this work can be found in [2].

The evaluation of structured queries over loosely structured data such as XML has been extensively studied during the last ten years. The techniques differ slightly depending on the system that is used to store the data, object-oriented, dedicated or relational. However, none of the existing approaches scale to the Web. We are considering here billions of documents (Google recently indexed more than one billion HTML documents) and millions of queries per day. So, query processing acquires in Xyleme a truly distinct flavor.

As is often the case, we represent queries using an algebra. The novelty of our approach lies in (i) a distribution pattern that scales to the Web (see Figure 1) (ii) the efficient implementation of a complex algebraic operator, named *PatternScan*, that captures so-called tree queries, i.e., queries that filter collections of documents according to some tree pattern and extract information from the selected documents. We capture the added expressive power provided by database-like XML query languages using standard operators (e.g., *Join*, *Map*, etc.).

The *Patternscan* operation is implemented using an index mechanism, named *XyIndex*, that is an extension of the full text index (FTI) technology. Whereas a standard FTI returns the documents in which a word occurs, a *XyIndex* adds annotations to position each occurrence of a word within a document relatively to the other words. Also, a *XyIndex* respects an order relation that allows to use efficient and pipelinable merge algorithms to combine the various sets of words occurrences corresponding to a tree query.

As will be explained in Section 7, Xyleme partitions documents into clusters of documents, each corresponding to a domain of interest (e.g., *tourism*, *finance*, etc.). Also, Xyleme provides *a view mechanism*, that enables the user to query a single structure summarizing a cluster rather than the many heterogeneous DTDs it contains.

A view in a relational system builds one relation by combining information coming from others. Similarly, a view in Xyleme combines several clusters into a virtual one. However, whereas the tuples of a relation share a common type, a cluster is a collection of highly heterogeneous documents. Thus, a view cannot be defined by one relatively simple join query between two or three collections but rather by the union of many queries over different sub-clusters (Section 7 explains how views are (semi-)automatically generated by the system). As a matter of fact, there is no *a priori* limit to the size of a view definition in Xyleme since it depends on the number of DTDs (that may be implicit in documents that are only well-formed) that one can find on the Web. Obviously, techniques that are used to maintain and query relational views have to be seriously re-considered to fit Xyleme's needs.

The view information in Xyleme is decomposed into two parts corresponding to the standard query processing steps: compilation and execution. The compilation part of a view is replicated on each upper-level machine (see Figure 1) and is mainly used to understand which index and repository machines are concerned by a query [4]. The size of this information is usually small, depending on the number of domains and the size of the view schema. The execution part of a view is distributed over the index machines (see Figure 1). Each stores only the view information that concerns its indexed cluster (or sub-cluster). This data is used to translate tree-queries just before they are evaluated by XyIndexes and is order of magnitude smaller that the indexes.

5 Data Acquisition

In this section, we consider the issue of data acquisition. More details may be found in [9].

We *crawl* the Web in search of XML data. We also need to refresh pages to keep the repository up to date. We implemented a crawler that can read millions of pages per day. Several crawlers can be used simultaneously to share the acquisition work. Note that we store only XML pages. For HTML, we read them to discover new links. A critical issue is the strategy to decide which document to read/refresh next. First, a Xyleme administrator is able to specify some general acquisition policies, e.g., whether it is more urgent to explore new branches of the Web or to refresh the documents we already have. The system then cycles around all pages it knows of (already read or not) deciding for each page whether this page has to be refreshed/read during this particular cycle. The decision for each page is based on the *minimization of a global cost function under some constraint*. The constraint is the average number of pages that Xyleme is willing to read per time period. The cost function is defined as the dissatisfaction of users being presented stale data. More precisely, it is based on criteria such as:

- *Subscription and Publication:* A customer may specify a desired refresh frequency for some particular documents. Also, some query subscription may request the regular refreshing of some pages. Finally, the owner of a document may let Xyleme know when a change occurs and request a refresh of the warehouse for this document.

- *Temporal information* such as last-time-read or change rate: These allow to estimate the number of updates that were missed for a particular page.

- *Page importance*: We want to read in priority documents that, we believe, are important and refresh them more often than others. To estimate page importance, we use the structure of the Web and the intuition that important pages carry their importance to pages they point to. This leads to a fixpoint computation.

Note that each page that is known by Xyleme is eventually read/refreshed (no eternal starvation) unless there is an explicit decision by the administrator to leave a portion of the Web out, e.g., pages below a certain importance threshold. Finally, observe that page importance can also be used to rank pages in query results as done, e.g., in Google.

6 Change Control

In this section, we discuss change control. Users are often not only interested in the current values of documents but also in their evolution. They want to see changes as information that can be consulted or queried. Indeed, change detection and notification is becoming an important part of Web services.

The first aspect of change control we considered is a subscription mechanism [11]. At the time we load or refresh a document, we detect whether it verifies patterns specified by some subscriptions. Examples of changes that can be monitored include, for instance, a modification of a particular document, the discovery of new documents with a given DTD. We also support monitoring conditions at the element level, e.g., monitoring of the newly discovered documents with an element tagged *address* containing the word "Marseille". The monitoring system we developed allows to monitor, on a single PC, the loading of millions of documents per day, with millions of subscriptions.

The second aspect of change control we considered is a versioning mechanism [8]. For each page, Xyleme gets snapshots of the page. It is possible to version some pages. To represent versions, we opted for a *change-centric* approach as opposed to a data-centric one that is more natural for database versioning. We use a representation based on *deltas* in the style of Hull's deltas. Given two consecutive versions of a page, we compute the delta using a very efficient *diff* algorithm we developed, tailored to our needs [5]. As often done in physical logs, we use *completed deltas* that also contain additional information so the deltas can be reversed and composed. We also assign persistent identifiers that we call Xyleme IDs (XIDs) to the elements of the documents. These

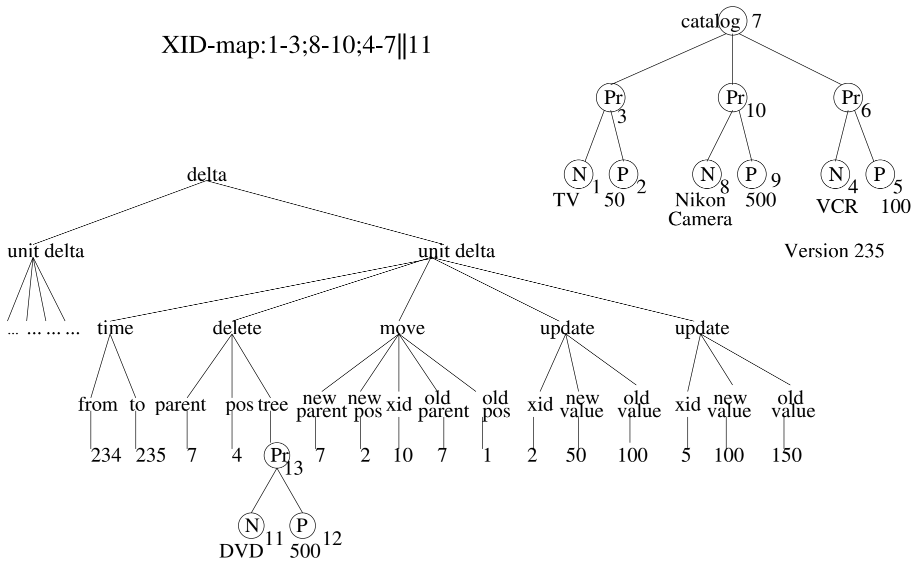

Figure 3: Storage of a versioned document

identifiers are essential to represent and control changes. One of the roles of the *diff* is to assign XIDs to elements in consecutive versions of a document. An example of our logical representation of versions is shown in Figure 3. In the figure, XIDs are shown in the XML tree although in reality, they do not exist inside the document but only in an auxiliary structure, the XID-Map. Note that a delta is also an XML document and thus, that it can be queried like any XML document.

We use deltas for versioning documents. We believe they can also be very useful to version results of continuous queries, i.e., queries that are evaluated regularly as in the Conquer [7] or Niagara [3] projects. Deltas may be used for other services as well; see [8].

7 Semantic Data Integration

In this section, we address the issue of providing some form of semantic data integration in Xyleme. More details may be found in [12].

Queries are precise in Xyleme because, as opposed to keywords searches, they are formulated using the structure of the documents. In some areas, people are defining standard documents types or DTDs, but most companies publishing in XML often have their own. Thus, a modest question may involve hundreds of different types. Since, we cannot expect users to know them all, Xyleme provides *a view mechanism*, that enables the user to query a single structure, that summarizes a domain of interest, instead of many heterogeneous schemata.

Views can be defined by some database administrator. However, this would be a tedious and never ending task. Also, metadata languages such as RDF may be used by the designer of the DTD or by domain experts to provide extra knowledge of a particular DTD. We believe that this will become common in the long range. However, for now, the field is too young, standards are missing and such knowledge is not available. Thus, we studied how to automatically extract it using natural language and machine learning techniques.

The first task is to classify DTDs into domains (e.g., *tourism*, *finance*) based on a statistical analysis of the similarities between the words found in different DTDs and/or documents. Similarity is defined based on ontologies or thesauri such as Wordnet.

Once an *abstract DTD* has been defined to structure a particular domain, the next task is to generate the semantic connections between elements in the abstract DTD and elements in concrete ones. Each element is identified by a path from the root of its DTD (e.g., *hotel/address/city*). The problem is then to map paths to paths. Obviously, all tags along a path (i) are not always words and (ii) do not have the same importance. Concerning (i), we use standard natural language techniques (e.g., to recognize abbreviations or composite words) and a thesaurus. As for (ii), we mainly rely on human interaction. Notably, the abstract DTD designer can indicate to the mappings generator that some tag is not particularly meaningful in a path, or, on the contrary essential. For instance, if we consider *hotel/address/city*, we can imagine that $hotel$ is meaningful but $address$ not that much. In this fashion, we will be able to map *auberge/town* to *hotel/address/city*, but not *company/address/city*.

Conclusion Working in the Xyleme Project was a fascinating experience. Many problems were encountered that still require to be further investigated. The work is of course continuing in the various research groups and in the Xyleme start-up.

Acknowledgments: We want to thank those who discussed many aspects of this work with us or those, in particular within INRIA's management, who supported us.

References

[1] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web*. Morgan Kaufmann, California, 2000.

[2] Vincent Aguilera, Sophie Cluet, Pierangelo Veltri, and Fanny Watez. Querying XML documents in xyleme. *ACM SIGIR Workshop on XML and information retrieval*, 2000.

[3] Jianjun Chen, David DeWitt, Fend Tian, and Yuan Wang. Niagaracq: A scalable continous query system for the internet databases. *ACM SIGMOD*, page 379, 2000.

[4] Sophie Cluet, Pierangelo Veltri, and Dan Vodislav. Views in a large scale xml repository. In *VLDB'01*, 2001.

[5] Grgory Cobna, Serge Abiteboul, and Amlie Marian. Detecting changes in XML documents. Technical report, Verso Group, INRIA-Rocquencourt, 2001. `http://www-rocq.inria.fr/verso/`.

[6] Carl-Christian Kanne and Guido Moerkotte. Efficient storage of XML data. Technical Report 8/99, University of Mannheim, 1999. `http://pi3.informatik.uni-mannheim.de/`.

[7] Ling Liu, Calton Pu, Wei Tang, and Wei Han. Conquer: A continual query system for update monitoring in the www. *International Journal of Computer Systems, Science and Engineering*, 2000.

[8] Amlie Marian, Serge Abiteboul, Grgory Cobna, and Laurent Mignet. Change-centric management of versions in an XML warehouse, September 2001. VLDB'01.

[9] Laurent Mignet, Mihai Preda, Serge Abiteboul, Sbastien Ailleret, Bernd Amann, and Amlie Marian. Acquiring XML pages for a webhouse, October 2000. BDA'00.

[10] Mind-it web page. http://mindit.netmind.com/.

[11] Benjamin Nguyen, Serge Abiteboul, Gregory Cobena, and Mihai Preda. Monitoring XML data on the web. In *ACM Sigmod*, 2001.

[12] C. Reynaud, J.P. Sirot, and D. Vodislav. Semantic integration of xml heterogeneous data sources. In *International Database Engineering and Applications Symposium, IDEAS*, 2001.

[13] World Wide Web Consortium pages. http://www.w3.org/.

[14] Xyleme Home Page. http://www.xyleme.com/.