Some Requirements for XML Repositories

Laurent Mignet, Anne-Marie Vercoustre
INRIA-Rocquencourt BP 105 78153 Le Chesnay France
{laurent.mignet, anne-marie.vercoustre}@inria.fr

Abstract

XML applications are developing rapidly. There is an urgent need for XML repositories to store XML documents, to retrieve and restructure them. This paper studies some of the requirements for such XML repository, i.e. the requirements for a data model, a query language and an architecture. We discuss various models and propose using Ozone, a model developed at Stanford, and an extension for managing links. We also present an architecture that couples smoothly the repository with a cache for managing external or virtual documents.

Keywords *XML, repositories, Data model, Query language, Links, Virtual Documents, cache.*

Introduction

XML is a new standard that has been designed to support data exchange on the World-Wide Web. It will replace HTML for the delivery of information for a large variety of applications. The advantage of XML relative to HTML is the semantic description of information or data through meaningful tags which may be described by a DTD, although it is not mandatory.

XML is already used in many applications. It may be not very visible to the end-users since information is still delivered on the Web mostly using HTML. Once XML browsers will become widely available, the number of XML sources will increase dramatically. XML is also been used to deliver data from databases where the data are not native XML since it is as easy as exporting HTML from a database. These XML documents could in turn be imported, physically or virtually, into other repositories and used by other applications.

The subject of this paper is to identify requirements for an XML repository, i.e. a management system for XML documents. The system should support storing, searching for documents, querying collections of documents, creating new documents, updating old ones or restructuring them, navigating within documents and across documents, just for a start.

Storing XML documents is easy. It has already been done with SGML (of which XML is an instance), for managing large SGML documentation or TEI literary collections. Many database vendors have made recent announcement for XML repositories.

Problems that arise are: i) the choice of the data model used in the storage system, and ii) the functionalities provided by the system.

The models that have been proposed and used can fit into two broad categories: The first one maps a DTD into a database schema, possibly to a limited level of tags. Then it is possible to query documents according to their structure and their content. We call this approach the *structured approach*. The alternative is to store documents using a very generic model, independently of any DTD, and to combine it with full text indexing both on content and structure for search purpose. In this approach the model used is what is now called a *semi-structured model* [MAG+97].

This paper will present advantages and disadvantages of both approaches and claim that we should get the best of the two worlds. We propose to use a new data model, called Ozone, introduced in [LAW98] and based on the ODMG model, as an appropriate model for storing XML documents. At one extreme, the model allows to represent a document by strictly respecting the DTD. At the other, it may ignore the DTDs and handle documents in a generic way.

We believe that an XML Repository must also support i) error recovery, ii) transaction, iii) concurrent access as standard requirements for DBMS [ABD+89]. We think that an XML Repository must be an OODBMS and in a rest of this paper we work above this affirmation. Nevertheless it is possible to build an XML Repository above a relational DBMS.

Section 1 presents the various models that can be used for storing XML documents and introduces Ozone. An example is given in Section 2 and used in the rest of the paper. Section 3 discusses some extensions of the data model for XML links. Section 4 gives the requirements for the query language. Section 5 introduces the notion of Virtual Documents as XML views upon the repository as well as upon external XML documents, and finally Section 6 proposes an architecture where the XML Repository is strongly coupled with a cache for managing external linked resources.

1 Modelling XML Objects

Literature about XML often refers to the *XML model* without defining it precisely. It is implicitly understood as a tree model defined by the hierarchy of tags. It is the *logical model* chosen by the DOM (Document Object Model) Recommendation [WA98] for XML documents. The DOM is an application programming interface (API) and a data model for building, modifying documents, or navigating their structure.

DOM does not specify that documents must be implemented as a tree, nor does it specify how the relationships among objects be implemented. The DOM is a logical model that may be implemented in any convenient manner. However the model is inherently hierarchic in nature which makes OO models better candidates than relational models. An XML repository should actually implement the DOM API as a minimal function set. Implementing the API through methods gives another argument for building the XML Repository using an OODBMS system.

This section studies the possible suitable data models for *implementing* the DOM in an XML Repository . Regarding links, the DOM in its current state only defines a model for simple HTML 4.0 links that includes a fixed number of predefined attributes (href, media, target, pre, rev, etc.) Models for XML links will be considered in 3.2

1.1 XML Document

An XML Document [BPMM98] includes three main aspects. The first one is the content of the document. The XML recommendation introduces only one data type #PCDATA (i.e. Text) for content, but the note XML-Data [LJM+98] insists on the need for many other data types (e.g. integer, float, dateTime.iso8601, ...). The second part of an XML document describes the model of a class of documents (Declaration Type Document), that gives a structure to the various parts of the corresponding documents. The third part concerns the links, both internal or external links to other documents or document subparts. The notion of link is introduced in Xlink [MR98], only a W3C working draft but in an advanced stage. This draft defines *simple* links, basically similar to HTML links, and *extended* links which can have more than one source and one target, and are often defined outside the XML documents they connect (*off-line links*).

Our vision of an XML document is represented in figure 1.

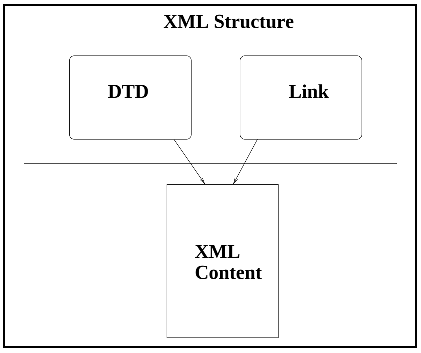

Figure 1: XML Document

Before presenting the possible data models for an XML Repository it is important to recall that an XML distinguishes between :

- valid documents which conforms to a DTD
- well-formed documents in which tags are only organized in a strict tree hierarchy.

1.2 Structured approach

A possible way of storing an XML document in an OO database is to map the DTD into a database schema and to store the content as value of the data. We consider here the ODMG model [Cat94, RC97]. This is the approached followed by [CACS94] and strongly supported by [MMA98]. A difficulty here is the absence in the ODMG model of *union* in the type system which does not make the mapping quite natural.

The advantages of using this model are the following: i)a rich type system useful for XML Data ii) closely control updating: parts of the XML document can be updated without reparsing the entire document iii) query optimisation. The disadvantages are i) only valid documents can be stored, or ii) DTD must be inferred for documents that are just well-formed iii) DTD updates entail expensive schema and database updates.

Point ii) is of strong concern since a collection of very heterogeneous documents may result in a schema nearly as big as the data itself.

1.3 Semi-structured approach

Another way to store XML documents is to use a semi-structured approach [MAG+97]. In a semi-structured model we can map directly an XML document in a (non typed) labeled directed graph where nodes describe the structure of the document (sub-elements, attributes, linking elements) and the leaves are the values (#PCDATA). Transversal edges between nodes may represent XML links between documents or parts of documents. An alternative to this model is to not instanciate links (as in [VP98]) and to represent XML documents as trees instead of graphs. We will discuss below our motivation for instanciating links whenever it is possible.

The advantages of this model are: i) it is an XML generic model, so we don't need to define a new schema whenever we get a new DTD ii) it allows for integrating heterogeneous XML documents in the same repository iii) it is possible to store well-formed documents as well as valid documents iv) the results of a query can be stored as a well-form document in this format after being wrapped in a result root element[1].

The disadvantages are i) the lack of stucture limits the possiblities for query optimisation; ii) less control of updates; iii) it makes it not easy or expensive to handle complex queries involving the structure of the document.

1.4 A hybrid Model: Ozone

The previous discussions suggests that we need to support both modeling structured and semi-structured documents. Ozone has been designed to support this approach. In summary Ozone permits to extend the ODMG structured object database model and its query language OQL with the ability to handle semistructured data based on the OEM model [PGMW95] and Lorel language [AQM+97]. With Ozone, structured data, as typed objects, may contain entry point to semistructured data, and vice-versa. A short description of the model will be given in 3.

Although Ozone was not defined exclusively for XML, it offers the primitives to manage well-formed XML documents as well as valid documents and to integrate them smoothly in the XML Repository while keeping the advantages of both approaches. Therefore we believe Ozone is the best approach, at this time, for modeling XML documents. In fact when a part of a document is valid we can map it with ODMG structure and when a part of document doesn't validate the DTD we can use the OEM object model to store it.

Moreover Ozone can support representing the DTD in the XML repository. Most OODBMS offer some capabilities from accessing the database schema (through an API), but OQL itself does not support querying or navigating the schema. The XML repository would need to access the DTD, if only to check the default values for attributes (e.g. xml:link attribute). For supporting DTD queries we can store the DTDs in the XML repository using the Ozone model. This permits to query, to see and to manipulate DTD as the same way as XML documents. This is in line with the current

[1]querying on XML attributes may also require some wrapping of the results

trends at W3C that try to define XML schemas using the XML syntax rather than the SGML syntax for DTDS (cf. XML-DATA schemas, RDF schemas).

A research direction is to use this schema representation to optimise queries on semistructured data.

Ozone also extends OQL (standard query language in ODMG) with the capabilities of Lorel language in a language called OQL^S. This extention will be discussed in the section 4.

2 Example

This example is inspired by the one presented in [LAW98] but is expressed under the form of XML documents and DTDs. Note that the advantage of the Ozone model is that it can support a flexible implementation where data can be native XML or native ODMG.

Suppose that we have a retail agency that sells products on the Web on behalf of different companies. The agency maintains their own catalog of vendors with their products, a product description list, as well as a directory of companies.

The current XML document describing the vendors could be the following (see table 1).

```
<vendors>
   <vendor>
       <details xml:link="simple" href="directory#ID-x"> company-x </details>
       <prod xml:link="simple" href="products#prod1"> product1 </prod>
       <prod xml:link="simple" href="products#prod3"> product3 </prod>
   </vendor>
   <vendor>
       <details xml:link="simple" href="directory#ID-y"> company-y </details>
       <prod xml:link="simple" href="products#prod2"> product2 </prod>
       <prod xml:link="simple" href="products#prod3"> product3 </prod>
   </vendor>
   ....
</vendors>
```

Table 1: vendor.xml (Extract)

The directory of companies (not shown here) contains the details (eg. name, address, size of the company) for each company, and there is an unique identifier (eg. ID-x) that can be referred to using Xpointers. The way these identifiers are created is shown in the product description (see table 2).

The example of the XML description for *Products* is given in Table 2. It has a flexible structure since the *competing* element is either a link element or a normal element. It is not possible to create a DTD that validates this document. This document could not be represented fully in the structured model.

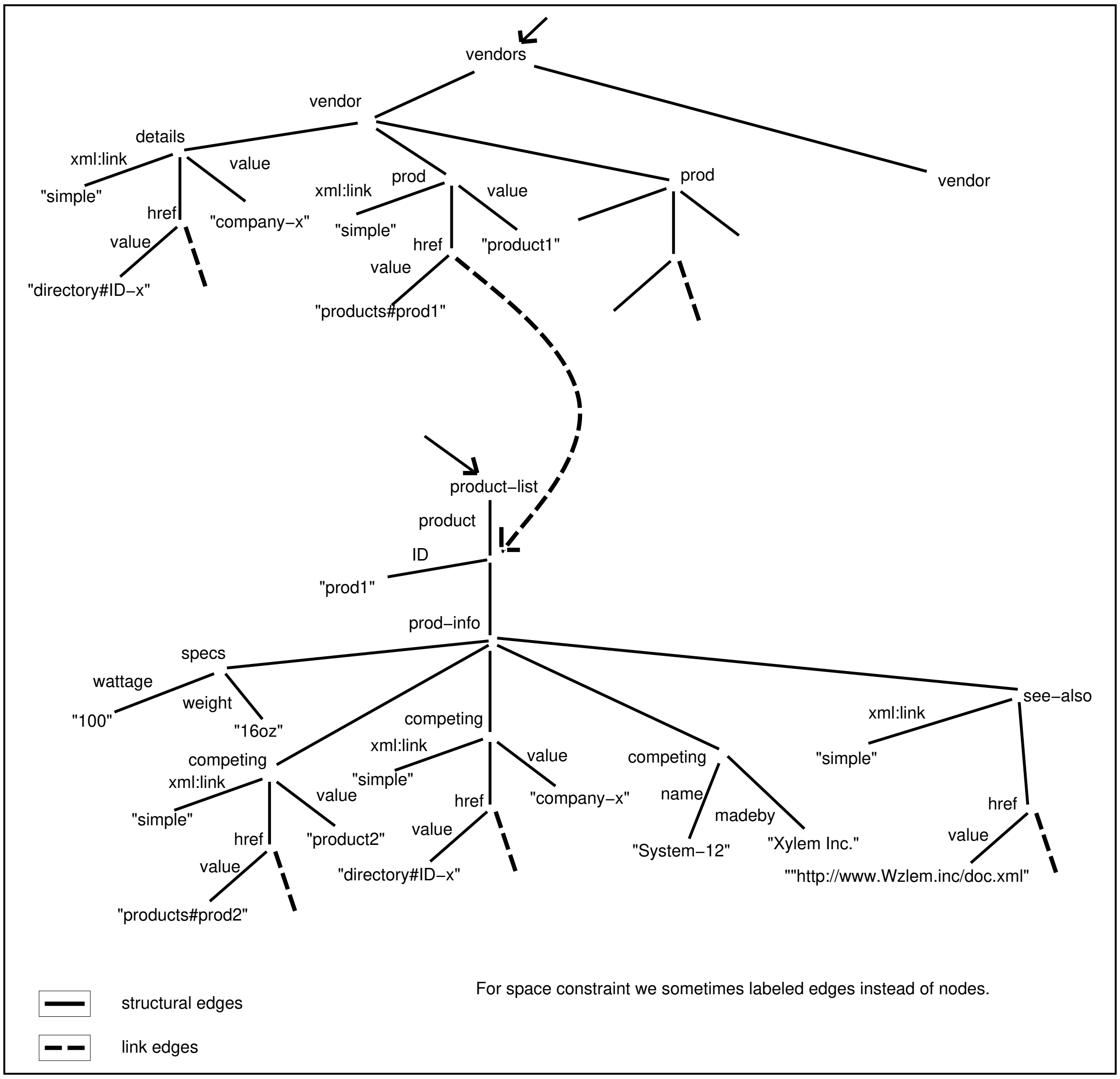

Figure 2: Tree of vendors and products

```
<product-list>
   <product ID=”prod1”>
      <prod-info>
         <specs>
            <wattage> 100 </wattage>
            <weight> 16oz </weight>
         </specs>
         <competing xml:link=”simple” href=”products#prod2”> product2 </competing>
         <competing xml:link=”simple” href=”directory#ID-x”> company-x </competing>
         <competing>
            <name> System-12 </name>
            <madeby> Xylem Inc. </madeby>
         </competing>
         <see-also xml:link=”simple” href=”http://www.Wzlem.inc/doc.xml”/>
      </prod-info>
   </product>
   ...
</product-list>
```

Table 2: products.xml (Extract)

Figure 2 gives a graphic representation for the Vendors and Products documents.

XML documents may or may be described with an explicit DTD. Even if they refer to an explicit DTD they could be stored in Ozone according to either the structured model, or the semi-structured one, or a mixed approach (e.g. using the ODMG model for the Vendors and the semi-structured model for the Products). The structured data (e.g. a vendor) can refer to a semistructured data (e.g. a product) and vice versa. The default loading could be using the semi-structured model while ODMG loading functions could be implemented and used for specific DTDs, or on request for specific DTDs or applications.

The above example also introduces XML links which we are studying in more details in the following section.

3 Ozone and Extensions to support XML links

3.1 Ozone: a brief overview

The Ozone data models extends ODMG and has all the necessary features of object-orientation: classes, encapsulating attributes and methods, subtyping, and multiple inheritance allowing complex class hierarchy. All values in an ODMG database must have a valid type defined in the schema. The possible types in ODMG are: atomic types (integer, float, string, etc., structured types (records), collection types (sets, bags, lists and arrays), and user-defined classes.

Ozone extends ODMG with a new class XML to represent untyped XML information. Objects in the class XML are two categories : XMLcomplex and XMLatomic.

(Note that these categories do not represent subclasses in the strict sense of the term, since XML objects are untyped). An XMLcomplex objects encapsulates i) a list of (tag,value) pairs and ii) a list of (attribute,value) pairs, where values are XML objects. Allowing structured, strongly-typed ODMG objects to have components of type XML provides "crossover points" from structured to unstructured information.

The value of an XMLatomic object may have any valid ODMG type. Thus, in addition the ODMG atomic types integer, float, string, etc., the value of an XMLatomic object may for example be of type Product (a class in the schema), list(tuple(a:integer, b:Product, c:XML)), etc. Intuitively, atomic XML objects are untyped containers for typed values. Note that an XMLatomic object containing an object of type XML is interpreted as a reference (IDREF). Allowing atomic XML objects to encapsulate arbitrary structured data provides "crossover points" from unstructured to structured information.

The query language for the Ozone system is OQL^S. OQL^S is not a new query language–it is syntactically identical to OQL. Furthermore, the semantics of OQL^S on structured data is identical to OQL on standard ODMG data. However, OQL^S extends OQL with additional semantics (based largely on the Lorel query language for semistructured data [AQM+97]) that allow it to access unstructured XML data. Furthermore, the semantics of OQL^S allows smooth navigation from structured data to unstructured data and vice-versa, and provides intuitive evaluation of expressions that mix the two kinds of data. OQL^S provides navigation of unstructured data without the possibility of run-time errors, while providing full type-checking support for the structured portions of the data. A complete description of the model and the system can be found in [LAW98]. An implementation above on top ODMG compliant system – O_2 of Ardent Software – has been realized at Stanford, in the database group.

3.2 Ozone extension for X-Link

Although XML links are not yet fully defined, and for this reason most XML Query languages proposals ignore them completely, we would like to make some requirements and proposals for XML repositories to support them in the future. The Working Draft XLink [MR98] introduces first the notion of *linking element* which asserts the existence and describes the characteristics of a link. *Simple* links are one-directional and most often in-line while *extended* links are used for multidirectional and off-line links. A linking element is an XML element that has specific reserved attributes, e.g. the attribute xml:link indicates that the element is a link, the mandatory attribute href precises a locator for the target of the link.

Example of simple links have been shown in the previous section. For example :
<competing xml:link="simple" href="directory#ID-x"> company-x </competing>
defines a link to the company-x description (eg. the element within the document "directory" that has an ID attribute equals to "ID-x"). A DTD description for this link can be found in annexe A page 18. If the xml:link attribute has a fixed default value it does not need to be present in the element which can be reduced to :
<competing href="directory#ID-x"> company-x </competing>.

Model and Typing Since XLINK are primarily XML elements they could be modeled in an XML Repository as any XML elements. Finding links is then done by checking the presence of the xml:attribute, either in the DTD definition or in the actual documents.

As we did for elements we can map the DTD for a link into a schema, or represent the links in the semi-structured model. However links are different in nature than element-subelements relationship. Unlike in [MMA98] XML Links are weakly typed. They are typed by their name, role and behavior, but not by the typed of objects they connect. This is not in the philosophy of strong typing in ODMG. Links need to be managed differently than other elements for various reasons. The requirements regarding links for an XML Repository are: i)it must be easy to find and follow links from or towards a document; ii)it must avoids chasing of dangling links. Finding specific links within a document of course can be done by searching such elements from the root of a document. However finding all the links that point to or from a document may be very costly if we do not introduce a type "link" in the object model. With offline links it will be the only feasible way of managing links for a specific document. In this case the XML Repository must be able to act as a link server as in Microcosm [CDDH98].

We propose to extend the Ozone model with a new class of XML objects called XLINK to represent XML links. XLINK objects will have the same structure as the XMLcomplex objects. A more specific structure could be defined as required when XML links are fully standardised. In the same way a generic class LINK must be created in the ODMG part of Ozone using the standard ODMG type system. Various types of links will be defined as subclasses of LINK in this model.

Internal and external links Links can be classed into two categories : i) simple links which have their source element embedded into an XML document through a *linking element* ; ii) extended links which have both their source and target defined outside any XML document.

Regarding the XML repository, we can also classify links according to whether they have their targets stored in the repository or not. We will refer to the first case as *internal* links, and the second case as *external* links. We don't need to distinguish between links within one documents or between different documents at the model level: i) internal links are stored as references in the Ozone model, whenever a document is loaded into the repository; external URI however may be kept in the repository for interfacing with the external world. ii) external links are stored as string values (URI), and external documents are managed through the cache.

Although the XML repository must deal with internal and external documents, we do not see it as mostly a virtual XML repository (unlike [MMA98]).

4 Query Language

There is currently a lot of activity concerning the definition of query language for XML. W3C organized a workshop in December 1998 devoted to the subject, and there already more than fifty proposals. We do not intend to propose yet another one, but want to identify some of the requirements in the context of an XML repository. A first statement is that such a language should draw upon various communuties who have been working on it from different point of view for many years, namely Web, IR and

DB communities.

Although rich operators for querying text are mandatory for some applications, we enumerate some basic requirements for an XML Repository query language :

- extracting from document collections (not just from a single document),
- navigating along the structure,
- navigating through links.

The SGML community has been more concerned with searching on the content of large collections (large SGML documentation, or large TEI linguistic corpus [GBY91, MMR95, BR]) using a little structure but a rich set of textual search operators. The database community and more specifically the OO database community has developed OQL [Cat94], a standard for querying and navigating data according to the structure. The Web and hypertext community have focused on following or querying links [MMM96, VP98, AMMT97] together with full text indexing.

We give three examples in the OQL^S syntax for illustrating the above requirements for a query language.

Query 1 This query searches for the competitors of prod1 (see table 3).

Select p.product.prod-info.competing From p in Products Where p.ID = "prod1"

Table 3: query 1

In this example OQL would return an error, because the last *competing* tag is not of the same type than others, while OQL^S returns all the expected results.

Query 1 bis In OQL^S this query can be rewritting using a star operator:

Select c From p in Products, c in p.*.competing Where p.ID = "prod1"

Table 4: query 1 with star operator

Query 2 This query (see table 5) returns all the competitors of products sold by "company-x". The example shows the problem of navigating following the links.

Prod is the linking element, and we probably need a more convenient and powerful operators for expressing following links.

```
Select c
From v in Vendors, c in v.prod.href.prod-info.competing
Where v.details.value = “compagny-x”
```

Table 5: query 2

```
Select tuple (product: v.prod,
              detail : (
                        Select v2.details
                        From partition v2) )
From v in Vendors
Group by v.prod
```

Table 6: query 3

```
<result>
    <prod xml:link=”simple” href=”products#prod1”> product1 </prod>
        <details xml:link=”simple” href=”directory#ID-x”> company-x </details>
    <prod xml:link=”simple” href=”products#prod2”> product2 </prod>
        <details xml:link=”simple” href=”directory#ID-y”> company-y </details>
    <prod xml:link=”simple” href=”products#prod3”> product3 </prod>
        <details xml:link=”simple” href=”directory#ID-x”> company-x </details>
        <details xml:link=”simple” href=”directory#ID-y”> company-y </details>
</result>
```

Table 7: result of query 3

Query 3 This query (table 6) creates the list of products together with the companies selling them.

The result could be expressed in XML as in table 7.

More generaly the result of a query should be an XML document. It is easy to construct a well-formed XML documents, but returning a valid document is an harder problem especially when it involves many DTDs.

Query 1 and query 2 has shown the need for more flexible syntax, allowing incomplete paths and link operators. It could be based on previous research works such as:

- the work on POQL [CACS94], that involves queries with complex paths and introduces path variables.
- the work on WebSQL language [MMM96] or Rio [VP98], they allow for following simple links from a document without necessarely giving the paths to the linking elements. This is very important in XML where extended links will not have embbeded linking elements. WebSQL offers regular expressions on link operators. It also distinguishes between internal ($\mapsto$), local ($\rightarrow$) and global ($\Rightarrow$) links. In the XML Repository we will distinguish between internal ($\mapsto$,$\rightarrow$) or external ($\Rightarrow$) links.

In conclusion, queries involving links must work the same for inline or offline links. It requires an appropriate syntax and processor.

5 Virtual document

Section 4 has introduced an example (Query 4) of complex query to create a views on the vendors/products data. As observed in [BLP$^+$98], Views on XML documents should be faithful to XML, i.e. they should always produce XML output. Complex XML views would therefore not be different than other XML document except that they may be kept virtual. XML virtual documents, seen as XML views, can benefit proved database techniques for creating and maintaining database views.

Views may or may not be refreshed whenever the data involved in the view have been updated. Active views extends the notion of views to allow specification of views that can be selectively refreshed at a very fine granularity level. It is built upon a notification mechanism that must be supported by the repository.

More generally XML Virtual documents can be defined as documents who contain parts not instantiated, and possibly not stored, in the XML Repository . This parts can be generated by the inclusion of external links or the result of requests to external sources. The definition of an XML virtual document will be stored in the XML repository, while the instantiated virtual document may be kept only in the cache. The cache is also in charge of managing external sources not stored into the XML repository. In the next section we are going to describe the architecture and the requirements for the XML cache.

6 Caching data

In the system, local data may reside in the repository or in the cache of the repository. We are not concerned here with this distinction and assume that the repository and its cache somehow are consistent. By caching data, we mean here that the entire system (cache+disk) is also used to keep (i) external data that has been loaded and (ii) results of the evaluation of virtual data.

Consider, for instance, the access to a particular very popular catalog that is available on some external sources. We may want to keep it in the repository. However, the catalog may be quite large, so we may prefer not to keep everything. So, we have to be able to specify exactly what should be kept. Clearly, the catalog probably changes regularly. We need to be able to specify when such changes should be propagated to the repository, e.g., every 1st of January, April, September. The prices of products in the catalog may change more often or we may be more concerned with such aspects. Thus we need to be able to specify that such prices are "refreshed" before any order is issued. When accessing the catalog, we may run into information that is not kept in the repository. If the catalog is available, a simple access fixes the issue. Otherwise, we still have to answer user requests to the best of our knowledge even when the information in the repository is incomplete or not up-to-date.

The specification of the external and materialized data that we want to keep in the repository will rely on the query language. We consider next some issues, the management of changes, the expiration policy [GMLY98], and the management of incomplete information.

6.1 Managing changing data

The management of changes introduces a number of challenges.

First, for local data, we may have computed virtual data, e.g., virtual pages. When the local data changes we need to detect that the view is potentially affected and maintain the materialized view. This can be achieved by recomputing the view. It is often preferable for performance reason to incrementally maintain it, a technique that has been intensively studied in databases, (see [AMR+98]). Note that such recomputation or maintenance may be quite expensive and many applications do not require that the view adapts in real time to local data. It is sometimes preferable to perform this only periodically.

For external data, the management of changes depends a lot on the capabilities of data sources. Some sources may be able to send notifications of changes messages and incremental updates. Many sources on the Web are incapable to do so. In most cases, the only feasible solution is to use expiration. Expiration is also sufficient when the source is not expected to change much and when access to stale (but reasonably recent data) is acceptable .

So, an XML repository must also offer the possibility to specify in an application a number of essential information on external and materialized data for the management of changes and in particular how data validity is defined (e.g., how much staleness is acceptable), how data invalidity is detected (e.g., notification, polling, expiration), how data is maintained (incrementally or by re-computation)

6.2 Document Expiration

We saw in the previous section that we should be able to specify the expiration strategy for data. We argue here for a fine grain in the expiration definition.

Expiration does not have to be brutal: e.g., expire all or nothing of a document. Suppose we have a large collection of reports. The strategy may be to expire reports that are more than 10 years old, and for those, we may want to keep the title, list of authors and keywords. So, the specification of expiration should be based on a rich language that combines features of a query language for XML and temporal aspects. (An example of a language with such features is DOEM, see [CAW98]).

6.3 Management of incomplete information

As mentioned in the previous section, the XML repository may refer to data that has been (partially) expired. Because of the natural unreliability of the Web, this data may be inaccessible. To continue with the report example. Suppose we ask for the title and abstract of all papers of Knuth with some keywords. The system may find that one such paper has been partially expired and is at the moment unreachable. It would be appropriate to still return the information about that paper even if we cannot provide the abstract. Indeed, the expiration we described entails the creation of incomplete documents which leads to the management of queries of incomplete information. The repository should be able give as accurate an answer as possible even in the presence of incomplete data.

6.4 Persistence

To conclude this section, note that persistence of such materialized view and external data is important in many applications. For instance, suppose we computed some virtual XML document at a high cost because they entail access to many Web sites and complex statistical computation. We would like this data to be persistent and resist computer failures. This XML document could acquire persistence by being stored in files. The use of an object database system more accurately serves that purpose: we have a better control of errors that occur in the middle of a document production, or in changes to an existing document. Also, the OODBMS garbage collector would detect the presence of useless data and thereby avoid storing data unnecessarily. (See the architecture in Figure 3).

Conclusion

This paper has presented some requirements for an XML Repository repository, focusing on the data model and the query language. We have shown that such a repository should be an OODBMS to offer the required functionalities and benefit the existing ODMG standard and its query language OQL, as well more advanced research works in OO databases. However XML requires a flexible model to support well-formed and valid documents, as well as extension to the query language for supporting querying or following links.

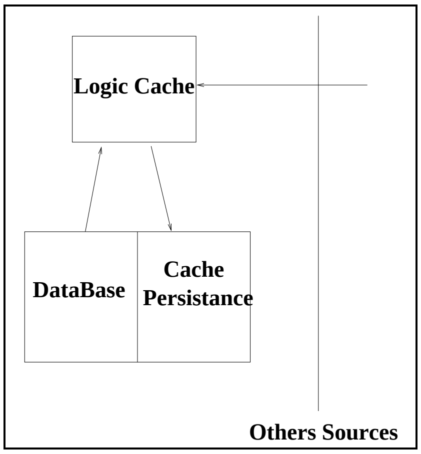

Figure 3: XML Document

We have also described an architecture using an associated cache that blurs the distinction between external and local documents, as well as between virtual and real ones.

Some problems has been just raised and need more work, e.g. a language for defining XML active virtual documents, based on a standard XML-QL language. For some applications we also need to define formally the type of the answer to a query, or to build tools for validating it against a specific DTD.

An XML Repository for electronic commerce is currently being implemented according to these requirements using the Ozone model and link extension. This repository will manage only a few DTDs (3 or 4) with very huge documents (catalogs), but may include a variety of external sources.

References

[ABD+89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The Object-Oriented Database System Manifesto. In *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, Kyoto, Japan, December 1989.

[AMMT97] Paolo Atzeni, Giansalvatore Mecca, Paolo Merialdo, and Elena Tabet. Structure in the Web. Technical report, Universita' di Roma Tre, 1997.

[AMR+98] Serge Abiteboul, Jason McHugh, Michael Rys, Vasilis Vassalos, and Janet L. Wiener. Incremental Maintenance for Materialized Views over Semistructured Data. In *VLDB*, 1998.

[AQM+97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, April 1997.

[BLP+98] C. Baru, B. Ludascher, Y. Papakonstantinou, P. Velikhov, and V. Vianu. Features and Requirements for an XML View Definition Language: Lessons from XML Information Mediation. *The*

W3C Query Languages Workshop, December 1998. Available from `www.w3.org/TandS/QL/QL98/pp/xmas.html`.

[BPMM98] Tim Bray, Jean Paoli, and C. MSperberg-McQueen. Extensible Markup Language (XML) 1.0. W3C Recommendation 10-Feb-98, World Wide Web Consortium, 1998. Available from `http://www.w3.org/TR/1998/REC-xml-19980210`.

[BR] P. Bonhomme and L. Romary. Silfide. Available from `http://www.loria.fr/projets/XSilfide`.

[CACS94] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From Structured Documents to Novel Query Facilities. In *SIGMOD*, pages 313–324, Minneapolis, Minnesota, May 1994.

[Cat94] R.G.G. Cattell, editor. *The Object Database Standard : ODMG-93 Release 1.2*. Morgan Kaufmann, San Francisco California, 1994.

[CAW98] S. Chawathe, S. Abiteboul, and J. Widom. Representing and Querying Changes in Semistructured Data, 1998.

[CDDH98] Les A. Carr, David C. DeRoure, Hugh C. Davis, and Wendy Hal. Implementing an Open Link Service for the World Wide Web. *World Wide Web Journal*, 1(2), 1998.

[GBY91] Gaston H Gonnet and R. Baeza-Yates. Lexicographical indices for text: Inverted files vs pat trees. Technical Report TR-OED-91-01, University Of Waterloo, 1991.

[GMLY98] Hector Garcia-Molina, Wilburt Juan Labio, and Jun Yang. Expiring Data in a Warehouse. In *VLDB*, 1998.

[LAW98] Tirthankar Lahiri, Serge Abiteboul, and Jennifer Widom. Ozone: Integrating Structured an Semi-structured Data. 1998. Available from `ftp://db.stanford.edu/pub/papers/ozone.ps`.

[LJM+98] Andrew Layman, Edward Jung, Eve Maler, Henry S. Thompson, Jean Paoli, John Tigue, Norbert H. Mikula, and Steve De Rose. XML Data. W3C Note 5-Jan-1998, World Wide Web Consortium, 1998. Available from `http://www.w3c.org/TR/1998/NOTE-XML-data-0105`.

[MAG+97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, September 1997. See `http://www-db.stanford.edu/lore`.

[MMA98] Giansalvatore Mecca, Paolo Merialdo, and Paolo Atzeni. Do we really need a new query language for XML. *The W3C Query Languages Workshop*, December 1998. Available from `www.w3.org/TandS/QL/QL98/pp/MeMAql98.html`.

[MMM96] Alberto O. Mendelzon, George A. Mihaila, and Tova Milo. Querying the World Wide Web. In *Proceedings of PDIS'96*, 1996. Available from `http://www.cs.toronto.edu/~websql/`.

[MMR95] Jaques Le Maitre, Elisabeth Murisasco, and M. Robert. SgmlQL, un langage d'interrogation de documents structurés. In *Poceedings* *of BDA'95*, Nancy, August 1995. see also SgmlQL reference manuel at`http://www.lpl.univ-aix.fr/projects/multext/MtSgmlQL/MQL2.html`.

[MR98] Eve Maler and Steve De Rose. XML Linking Language (XLINK). W3C Working Draft 3-March-1998, World Wide Web Consortium, 1998. Available from `http://www.w3c.org/TR/1998/WD-xlink-19980303`.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 251–260, Taipei, Taiwan, March 1995.

[RC97] Douglas K. Barry R.G.G. Cattell, editor. *The Object Database Standard : ODMG 2.0.* Morgan Kaufmann, San Francisco California, 1997.

[VP98] Anne-Marie Vercoustre and François Paradis. Reuse of linked Documents through virtual document Prescriptions. In *Electronic Publishing '98*, Saint Malo, France, 1-4 April 1998.

[WA98] Lauren Wood and Al. Document Object Model (DOM) Level 1 Specification. W3C Recommendation 1 October 1998, World Wide Web Consortium, 1998. Available from `http://www.w3c.org/TR/1998/REC-DOM-Level-1-19981001`.

Annexe A: DTD Extract

```
<!ELEMENT vendors (vendor+) >
<!ELEMENT vendor (details, prod+) >
<!ELEMENT details (#PCDATA) >
<!ATTLIST details xml:link CDATA #FIXED ”simple”
        href CDATA #REQUIERED
        >
<!ELEMENT prod (#PCDATA) >
<!ATTLIST prod xml:link CDATA #FIXED ”simple”
        href CDATA #REQUIERED
        >
<!ELEMENT product-list (product+) >
<!ELEMENT product (prod-info) >
<!ATTLIST product ID CDATA #REQUIERED >
<!ELEMENT prod-info (specs,competing+, see-also) >
<!ELEMENT specs (wattage,weight) >
<!ELEMENT wattage (#PCDATA) >
<!ELEMENT weight (#PCDATA) >
<!ELEMENT competing (ANY) >
<!ATTLIST competing xml:link CDATA #FIXED ”simple”
        href CDATA #REQUIERED
        >
<!ELEMENT name (#PCDATA) >
<!ELEMENT madeby (#PCDATA) >
<!ELEMENT see-also EMPTY >
<!ATTLIST see-also
        xml:link (simple | extended) #REQUIRED
        href CDATA #REQUIERED
        >
```

Table 8: Catalog.dtd