On Views and XML

Serge Abiteboul
I.N.R.I.A., 78153 Le Chesnay, France
Serge.Abiteboul@inria.fr

1 Introduction

The notion of views is essential in databases, see for instance [29, 30, 5]. It allows various users to see data from different viewpoints. In the present paper, we informally present works of the author on the topic. Instead of addressing the issue of views in a classical database setting, the paper focuses on XML [32] and looks at these various works in the context of a system of views for XML.

The Web has revolutionized the electronic publication of data. It has relied primarily on HTML that emphasizes a hypertext document approach. More recently, XML, although originally a document mark-up language, is promoting an approach more focused on data exchange. In XML, explicit structuring is enforced and presentation is separated from the data content. For data sources containing information with some structure, it is therefore more appropriate to use XML rather than HTML to export their data to the Web. When data is exported via XML, the problem of views becomes essential. Indeed, views in this setting are even more crucial than in standard database applications because (i) one often has to integrate heterogeneous sources and also (ii) views provide the means to add a structured interface on top of some otherwise (more chaotic) semistructured data.

In some sense, a language already allows to define views for XML documents, namely XSL. XSL is the current (still unstable) W3C proposal for expressing stylesheets. Although primarily targeted towards presentation, XSL allows to transform/restructure XML documents using *templates rules*. We are discussing such restructuring here. However, we will ignore presentation issues and will consider more general views than offered by XSL.

A view specification for XML data[1] will primarily rely, like for relational views [14], on a data model and a query language.

We will argue that the data model should be as the ODMG model [11] based on objects. In general, we will argue that XML view technology should borrow a lot from the object database view technology. However, XML data is not regular like in the relational or object models, which leads to considering semistructured data models [3, 1, 10, 28]. We will argue that this should not be to the detriment of regularity and structure, when it is known. Furthermore, we will argue that the data model should allow the management of incomplete information.

A central issue for the definition of views for XML data is the query language. Unfortunately, there is no standard yet for such a language although the activity invested towards obtaining one is rather intense [33] and a standard should emerge soon. This is a complex issue that we will partially address here. We could have stated some desiderata for an XML query language. However, since we could not add much to [21], we prefer to simply endorse that proposal.[2]

Finally, we believe that a declarative specification of XML views should encompass aspects that are typically not found in relational or object database views. This comes from Web applications that are by nature distributed. So, for instance, a view should specify aspects such as replication and provide active features such as change notifications.

Although the main purpose was not to survey works of the author on the topic, the paper is clearly influenced by previous works at INRIA and Stanford. In

[1] XML data for us refers to data in XML and not to the precursor of DCD, a typing language for XML, that was called *XML data*. We prefer to use the term XML data instead of XML document to stress that our prime concern is in data exchange and not in document management.

[2] We perhaps could add that ODMG [11] provides answers to many problems that are raised by [21] and in particular to some of the modeling and query language issues. Thus, we favor an approach in the spirit of *Ozone* [19].

particular, we will briefly discuss *O_2-Views* [27], a view mechanism for ODMG databases, *Ozone* [19], a system allowing to mix structured and semistructured data, and *ActiveView* [2], a view system for XML data with active features.

In Section 2, we discuss views of XML data. We look at some existing relational database technologies for views in Section 3. Section 4 deals with views of object databases. Sections 5 and 6 deal, respectively, with semistructured and structured data in an XML context. Section 7 deals with the control of updates. Section 8 considers problems related to seeing the view as a workspace. Finally, in Section 9, we argue that a proper data model for views should be based on incomplete information.

2 Views for XML

XML is still in its infancy and it is hard to predict what it will become. For many people, XML is just a document mark-up language. For us, XML data consist in a forest of labeled (annotated) ordered trees with references and in the possibility to type portions of the data with DTD's/DCD's. More precisely, in each tree, the children of internal vertices are ordered, the edges are labeled, and the leaves may contain references to vertices of the same or another tree. Clearly, XML is much richer but, from a data exchange viewpoint, this simple model will suffice for our discussion. This view of XML elements with an object flavor is in the DOM [16] spirit. If some features we will implicitly assume are not yet supported by the standard, it is very likely that they will soon, as well as (based for instance on SQL3 experience) many more that will not be considered here.

An example of some (untyped) XML data is shown in Figure 1. Its representation under a graphical form is also given there. We will assume the existence of a "declarative" query language for XML in the style of SQL or OQL hoping that a standard for such a language will exist soon.

In this paper, we are interested in views of XML data. The need for such a concept is first like in traditional databases: different users sharing XML data may have different needs and may want to see the same data differently, and this not only at the presentation level. Furthermore, since XML data is primarily used as a common model for otherwise heterogeneous data, the use of views is even more essential than in classical databases.

When considering XML views, it is worth mentioning briefly the architecture. A possible architecture is shown Figure 2. It is based on three components:

- the data server that may be a database, an XML repository, or any (possibly wrapped) source capable of exporting XML data.

```
<states>
   <state id = "s1">
      <scode> ID </scode>
      <sname> Idaho </sname>
      <capital idref="c1"/>
      <cities-in idref="c1"/>
      <cities-in idref="c3"/> ...
   </state>
   <state id="s2">
      <scode> NE </scode>
      <sname> Nevada </sname>
      <capital idref="c2"/>
      <gambling> many casinos ...
                      </gambling>
      <cities-in idref="c2"/> ...
   </state>
...
</states>
<cities>
   <city id="c1">
      <ccode> BOI </ccode>
      <cname> Boise </cname>
      <personal> don't go </personal>
      <state-of idref="s1"/>
   </city>
...
</cities>
```

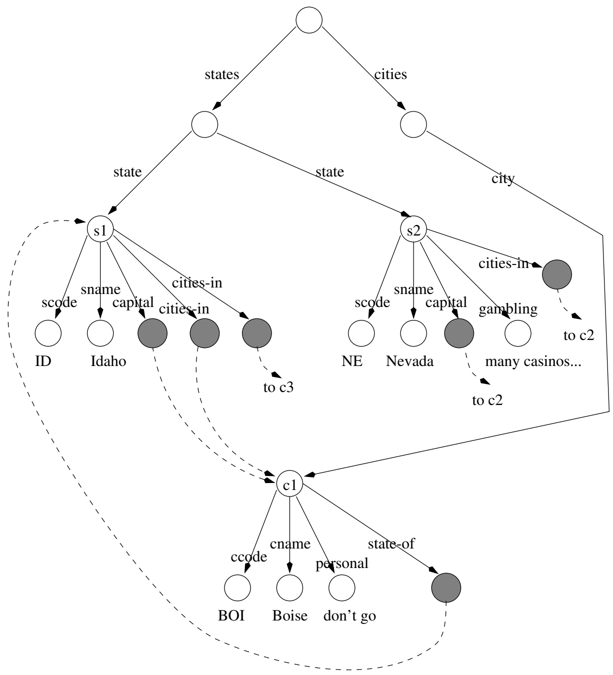

Figure 1: Some (rather regular) XML data

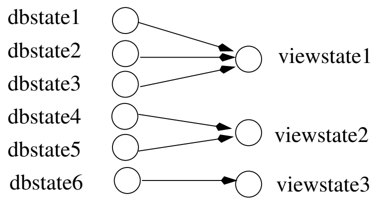

Figure 3: A view is a function

- the view server that restructures data to construct the view, possibly deals with access rights, and integrates data from several sources.
- an XML view document that is handled by a standard Web browser and interacts with the view server, e.g., to obtain data.

We will not elaborate on this architecture here. To illustrate it, we will mention the *ActiveView* system that is being developed at INRIA. In ActiveView, the database server is the Ardent Software XML repository (developed on the O_2 system); the view server is a Java application. The view document is for the moment in dynamic HTML with embedded Java applets and will move to XML as soon as XML browsers provide the desired dynamic features. The protocol between the repository and the view server used DOM. The protocole between the view server and the Web browser is via Java Remote Method Invocation. We will discuss further on some *active* features of the system.

Tag line Database folks should be interested in XML (views) and more and more are.

3 View = Query

One of my first papers in databases was on views [7]. At that time, I believed this simple definition: *a view is just a function*. (See Figure 3.) And yes indeed, this definition remains true many years after[3]. But, the devil is in the details and it is difficult to handle functions when the world is changing and when applications attempt to present rich data to demanding users.

In the relational world, a view is simply specified by a query. We will be led to enrich this specification quite a bit. However, a lot of the technology developed for relational databases remains meaningful in the context of views for XML data. Most of the work on relational views deals with the propagation of updates:

db → **v** In one direction, when data change, incremental techniques have been developed to avoid entirely recomputing the views. In a Web context, suppose we are publishing a Web catalog of thousands of articles. A customer may have loaded portions of the catalog. If the price of a single item changes, we do not want to reload the entire catalog.

v → **db** In the other direction, we find the problem of propagating view updates to the data sources. This is sometimes considered less of an issue since the Web is now viewed primarily as a "read-only" resource. However, many Web applications do allow updates. Consider for instance a bug management system. A programmer may consult bug reports, but may also edit them. If the programmer is presented with views of the bugs actually stored by the system, we are led to a view update problem.

The fact that the data is now in XML doesn't eliminate the problems although the use of (DOM) objects may facilitate the support of update propagation in some cases.

It is clearly possible to reuse results for relational views in an XML-view context. We mentioned update propagation but clearly query optimization for logically accessing large collections is an indispensable technology for Web applications. We should mention also more recent results on answering queries using views, e.g., [12, 20, 34, 4, 25], often developed already with Web applications in mind.

Tag line XML Views require standard database view technology, but much more.

Tag line What databases can bring to XML is query optimization and query rewriting.

4 View = World of Objects

We believe that XML data are by nature object-based which strongly supports the DOM viewpoint. This will be easily seen in an example. Suppose that a user checks out Paragraph 3 of Section 4 of a large document while others may be editing other portions of the document. The system needs the means to refer to that particular element. We are not talking about some identification of the tree vertex such as "MyDoc.Section[4].Paragraph[3]" since someone may be shuffling sections around, adding/removing paragraphs, etc. The problem is even more crucial in more "databasy" contexts with set collections, e.g., editing the design of a part in the engine of a car described as XML data.

Indeed, as discussed in [21], we need the means to designate locations within some XML data. Such a concept is called *locator* in [21]. This is really captured by object identity irrespective of the presence or not

[3] We have to be careful with the converse statement that *a function is just a view*, so almost anything you can think of is a view.

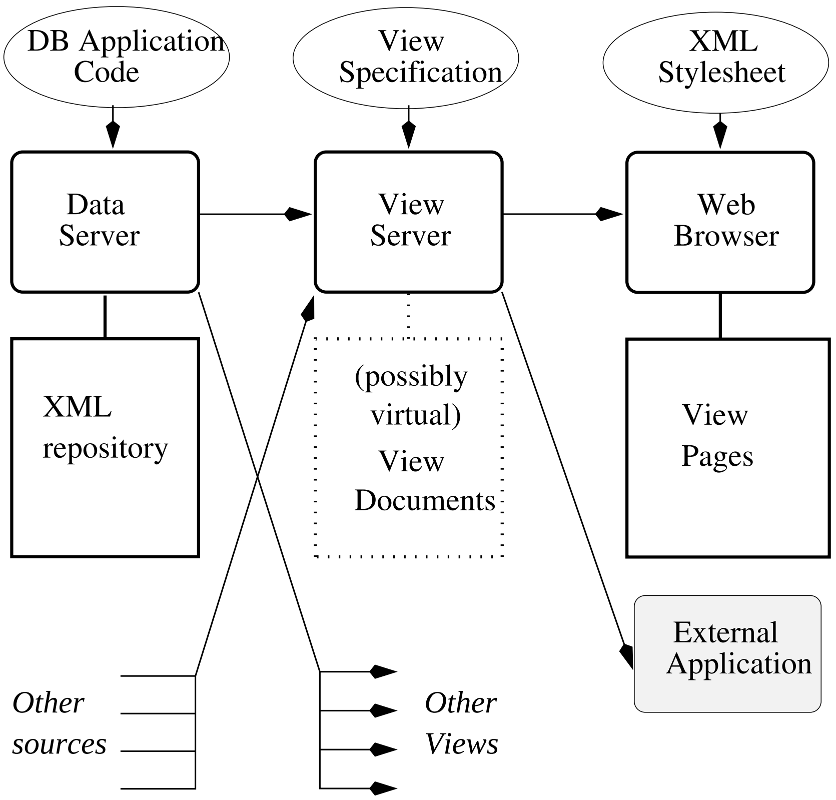

Figure 2: View Architecture

of some XML *name* to denote the element that is updated. Thus, we prefer to think of XML as an object model (in the DOM style) with object identities that may be *exported*.

Since we have objects, we may as well introduce code and methods. These are clearly useful, for instance, to define virtual XML data or to use conversion functions. So, based on the DTD/DCD, elements are seen as organized in classes. We will be less concerned here with notions such as inheritance and late binding although these are definitely nice to have around. This said we are in a world very similar to the object database world and most notions about views in object databases may be imported to views of XML data.

This is what we do next primarily influenced by works on O_2-Views [27], C. Souza's thesis and the system he implemented.

O_2-Views The goal of O_2-Views was to propose a rich view mechanism for ODMG databases. In O_2Views, three main mechanisms are used to define object database views that are relevant for XML views as well:

- virtual values: these are essentially like relational views. The notion carries immediately to having virtual data in an XML view, from entire documents to virtual elements. For instance, suppose that we have for each customer a list of the unpaid orders of that customer. We could add a virtual element that would contain the customer's outstanding balance.
- virtual classes: a set of database objects may be logically grouped into a view class, and, as such, they may acquire new interfaces. To see how this carries to XML, suppose that an electronic catalog offers products sold by several companies and each company uses a different DTD for describing its products. The view may specify the mappings between the products in original sources and the products exported by the view. Then the view provides a uniform access to all products using a single DTD. Note that each exported product element corresponds to an existing product element in the repository.
- imaginary classes: an imaginary class allows to (virtually) create a set of objects that exist only in the view and do not correspond to database objects. This may be used, starting from a relational database, to (virtually) create some XML data, with for instance, one element corresponding to each tuple in the join of two relations. Note that exported elements do not have corresponding elements in the repository.

These notions probably do not sound new in a Web context. It is more and more common to have HTML or XML pages produced on demand, e.g., to have queries embedded in documents. We wanted to stress that this can be done more cleanly within a formal model.

Tag line The underlying XML model is an object-based model and XML views should be founded on object database views.

5 View = Semistructured data

As advocated by [21], there is a need for a model for XML. We are not really talking here about all the bells and whistles already in XML but about a model that would capture the essence of data in XML. The use of such a model seems to us a prerequisite for being able to define queries with precise semantics, so views as well. In the present section and in the following, we propose elements of answers to this problem. We already mentioned that we see the model as object-based. It has been argued that semistructured data models are appropriate for XML. This is the point of view adopted in this section. In the next section, we also argue that the model should include structure as well, when some structure is known.

There have been many recent reports promoting semistructured data, and query languages for such data [1, 10, 28]. We will assume here that the reader is convinced of the need to manage such data, the prime motivation being data exchange formats, in particular XML, and irregular data resulting from the integration of heterogeneous sources. There has been a lot of work on querying semistructured data (e.g., [22, 9, 23, 15]) that will hopefully impact on query languages for XML [33]. We will use here the Lorel language [6] and not insist on an XML-flavored syntax. (This is just an issue of syntax relevant for a standardization committee but less so to the research community.)

Although the data structure in semistructured data models, roughly speaking a labeled graph, is certainly not new, the management of such data yields novel issues. Perhaps, a most important one is query optimization with exciting problems in physical organization, clustering or indexing. We will not address these issues here.

There will be no tag line in this section since there have been already too many beliefs expressed about semistructured data. In the next section, we will argue that semistructured data should go together with structured data as well, the corollary being that, for data exchange, XML should encourage the use of types ala ODMG even if totally untyped XML is fine.

6 View += Structured data

We want to insist here on the need for XML to support simultaneously structured and semistructured data with cross references between these two worlds. In short, the motivations for handling structured data are as follows:

1. if we know about some structure in some of the data we are managing (e.g., that we have a large collection of similar tuples), then not using such information may seriously damage performance.

2. we can use a regular structure as access structure over some existing XML data to boost performance. (For instance, if we manage a set of thousands of home-pages, it may be appropriate to introduce a relation[4] or a class *Person* with attributes name, address, phone number, picture as an access structure.)

3. the use of structure facilitates the programming of applications since languages such as Java or C++ do expect typed data.

In (1) and (2), we end up having to write applications where some data is structured and some semistructured. More motivations, a query language, and a system, namely Ozone, for such hybrid data are described in [19]. We next briefly present the Ozone approach.

The Ozone model Rather than starting from scratch, Ozone extends the ODMG model with a class for semistructured data inspired by the OEM model [24]. More precisely, the semantics of a semistructured object is either complex (a collection of pairs $\langle label, oem \rangle$), or is a container for a typed value (int, real, reference to an OEM). Collections in complex semistructured objects of Ozone may be sets or lists. The use of container allows to capture the distinction in XML between attributes/subelements and references. Consider the example of Figure 1. For instance, s_1 will be represented by a complex OEM object whose value is a list of 5 $(label, value)$ pairs. The first one is $(scode, o)$ where o is an atomic object containing the string ID. The last one is $(cities\text{-}in, o')$ where o' contains a reference to the object c_3.

The query language Here also, rather than starting from scratch, Ozone extends the OQL language. The language is in the style of many languages for semistructured data, and in particular can be viewed also as an

[4] This is not a departure from the XML world since the regular structure of a relation may be captured by a strict DTD and DCD-like typing.

extension of Lorel. The novelty of the language resides in the possibility to query hybrid data, with structured portions referencing semistructured ones, and vice versa.

A key notion towards this goal is that of *proxy* that is a structured interface to semistructured data, or a semistructured interface to structured data. In one direction, we may want to use objects with regular structure as proxies for irregular objects, e.g., have proxies with structure

```
struct( name:string,
        address:string,
        phone: string,
        picture:gif    )
```

for thousands of *Person* objects that may exist in the repository containing such information and more. This may allow (i) to improve performance, e.g., by having an index on *name*, and (ii) to facilitates the development of application code, e.g., in Java or C++. In the other direction, semistructured (logical) proxies to structured data allow to provide a semistructured view of structured data and thus allows ignoring (some of) the structure when querying such data.

Note that proxies as used in Ozone can be seen as *view* mechanisms that blur the distinction between structured data and semistructured ones.

What is the impact of this on XML views? Data sources do provide data organized in relations, in typed collections of objects, etc. We believe that ignoring such structure (when it exists) is a bad idea for performance as well as logical reasons. Thus XML (in its general context with DTD and DCD) should provide the means to export both structured and semistructured data.

Tag line XML should allow the exchange of structured data as in the relational and ODMG models.

7 View = Changing World

Users are now used to seeing HTML/XML data on the Web. To a certain extent, more and more want to see only such data, which is certainly a solution to many problems coming from data exchange format heterogeneity. The massive diffusion of data in the Web is often performed without the use of database systems. This may be because the data are relatively small and a heavy duty DBMS would be overkill or because the data are too large and for performance reasons, require tailored systems (e.g., indexes in Web search engines). However, data used by Web applications are by definition *accessed* by many users and it is often the case that data are also *updated* by many users. When we start dealing with changes in shared data, this becomes even more a database issue and database solutions become more compelling. Indeed, this explains why databases are used more and more in Web sites. The management of changes in an XML view environment is the topic of the present section.

We have been working recently on a system called *ActiveView* [2]. The idea is to offer a declarative definition of Web views of some XML data with change control and active features. We illustrate this work next.

ActiveView The following simple statement may be part of a definition of the customer view of an electronic catalog:

```
let monitored catalog : CatalogElem
be RepCatalog
with catalog.*
mode append catalog.product.opinions
```

The specification of view data is based on XML queries. The only query here is extremely simple, *RepCatalog*, i.e., the name of a document in the repository. The *with* clause specifies to import all the elements in this document. (The reader will ignore the particular syntax. We use a Lorel/OQL style syntax until a standard query language arises.) The type of the answer and thus of the catalog as viewed by a customer is a DTD, namely *CatalogElem*. The access mode for the data is defined by a *mode* clause. Here the entire catalog will be in read mode (the default), and the view user will also be allowed to add *opinions* about products. The keyword *monitored* specifies that the view has to be notified when the catalog changes, so that the view may request, if desired, an incremental update.

More generally, the ActiveView system can be seen as a *database application generator*. The system enables a *declarative* specification of *certain kinds* of views. By declarative, we mean here that there is little (or no program) to write and that the description of the application is in a high level language (or via a graphical user interface). The specification of an application includes definitions of the main actors involved in the application. For each actor, we specify:

1. the data and operations available to this particular actor (a *view* mechanism) and these with a sophisticate access control;
2. the activities this actor may be engaged in and the data and operations available in each;
3. some active rules that notably specify the sequencing of activities (a *workflow* component) but also the events this actor wants to be notified of (a *subscription* component) and those that have to be logged (a *tracing* component).

To see an example, suppose a product is added to the catalog. A notification is issued to all actors that are interested in this event, i.e., a change in the catalog. The specification may also include an active rule to specify that, when such an event occurs, their view of the catalog should be updated. Observe that both the detection of the event and the maintenance may take advantage of incremental techniques.

The focus in the ActiveView system is on the control of updates and on a declarative specification of views. Note also that users often want to query changes. For instance, one might want to ask a query such as *what are the books by South American authors entered since January 16th, 1998.* This introduces standard issues in temporal databases, see [26]. The management of temporal semistructured data is the topic of Chawathe's thesis [13] and the *DOEM/Chorel* framework. Such capabilities will probably be the basis of a number of new services such as query subscription systems.

Beyond the specific issues mentioned here, what we wanted to stress is that the control of changes in shared data yields a number of issues where technologies developed in databases fit nicely. New issues also arise. For instance, the introduction of workflows (see, e.g., [31]) to control these changes is a challenging issue.

Tag line What databases can bring to XML is also the control of updates.

8 View = Workspace

We are primarily interested here by *distributed* data. In particular, a view will typically be on a different machine than the data sources. Thus, a view should be thought of as a *workspace* that in particular may contain previous queries and the results of these queries. This raises a number of issues such as (again) the problem of answering queries using views and the management of replicated data. To illustrate the issues related the management of the view workspace, we consider in more detail the semantics of XML queries.

Consider the query

Who are the authors of papers on XML?:

```
Q:  select P
    from Mybiblio.paper P
    where P.keyword contains "XML"
```

Ignoring that we are using here an OQL syntax, suppose that the source consists of XML data and the query output is also in XML. (It seems well-accepted that an XML query returns some XML output – a closure property.)

In an "object-based" language (say OQL or Lorel), this is returning a collection of objects, each corresponding to one paper. In DOM terminology, the answer to this query may be thought of as an entry point to a set of elements stored in the repository (assuming some locators for these elements.) However, in a workspace context, the situation is somewhat different. We would like to specify the data that should be *transferred* together with the objects, i.e., to save on communications, we may wish the XML data returned by the query to be more than just a bunch of locators.

In ODMG, it is easy to specify that we want the query to return the value of each object as well. In a semistructured context, there is one notion of "value" of a complex object, i.e., a collection of (label,object) pairs. But this is not what we mean here, we would like to return specific data we know about each particular paper. A particular syntax for that is used in Lorel. One may use a *with* clause to specify what exactly to return [8] for each P element selected by the query. For instance, one could use the following query:

```
select P
from Mybiblio.paper P
where P.keyword contains ‘‘XML’’
with P.title, P.abstract, P.author.*
```

that requires to glue the *title*, *abstract*, and all data reachable from attributes *author*. (Note that the *with* clause is a non standard syntax. We believe that a query language for XML would have some syntax to express this notion of gluing data with the result of a query. This may be like here in a *with* clause or elsewhere, e.g., as part of the *select* clause.)

Thus, a first solution, is to specify explicitly what to return. Let us move closer to XML to see a possible solution XML suggests for this problem. In the underlying XML model, there are two kinds of links: (i) an element may be a *component* of another; and (ii) an element may *reference* another. Intuitively, elements together with their components and this recursively seem the natural atomic unit of transfer. So, if we consider query Q above, a natural XML solution may be to import the entire *paper* element. If the XML style solution is often appropriate, it may be too constraining in some situations. In many cases, we may prefer a tighter control of what is transmitted.

There are many other issues when we start considering a view as a workspace such as where some code is executed. To a large extent, a lot of the technology for that exists but needs to be reconsidered.

Tag line A query language and the specification of views should take into account distribution.

9 View = Incomplete World

Incomplete information is not a very active topic these days. Important results were obtained in the past, but the database field has more and more a tendency to go according to Web time and ignore them as "too old".

In this last section, we briefly argue (by example) that views (in particular in a Web context) should be approached with an incomplete information model and recall a model for incomplete information of Lipski and Imielinski that is somewhat typical of great tools that remain unfortunately mostly unused.

To be able to use directly previous works on incomplete information, we present here relational examples. Clearly similar examples can be given with XML data sources.

First, consider the following scenario. A view is built from a set of vendors that sell products on the Web. More precisely, the view consists of (i) the collection (i, v, p) such that item i is sold by v at price p, (ii) the collection (v, c) such that the vendor is located in city c. (Let us assume that all vendors publish prices for all their products.) Suppose we first ask for the vendors of some *Gismo45* product in Paris. We may obtain the table in Figure 4 (a). We decide next to ask for the price listed by each vendor. Suppose that the source that provides the prices for some vendors (say $v2, v5$) is temporarily unavailable. The answer should contain incomplete information as in Figure 4 (b) or otherwise would be inconsistent with the previous answer. (In the figure, variables start with capital letters.) Such incompleteness is easily captured with the simplest kinds of conditional tables [18] (as in Figure 4 (b)). Furthermore, suppose that the user wants to restrict the answer to Gismo45's under $100. Then again conditional tables allow the representation of that uncertainty. (See Figure 4 (c)). Depending on the application, we may want to see or not incompleteness in the answer. However, it is essential to consider it in the view since the Web naturally yields incompleteness because of the unavailability of sources.

To see another example, consider the problem of data *expiration* for instance studied in [17]. In [17], data is expired from a view explicitly, e.g., to save on storage space. One may also consider that some data is expired because some validity time is attached to it. Some of this data may only then exist off-line or may be temporarily unavailable. Suppose for instance that we have some large quantity of technical reports and that we decide to keep only the title and authors of tech reports before 1990. (The expiration policy may be much more complex, e.g., depend on the topic of the reports, the issuing institution, the authors, etc.) We need a model of incomplete information to represent such data and query it. The fact that we are dealing with possibly semistructured data instead of structured data does not change much the problem. A more essential difference is that incompleteness becomes a more dynamic notion since data may be constantly added/removed from the view.

company
$v1$
$v2$
$v3$
$v4$
$v5$

(*a*)

company	*price*
$v1$	109
$v2$	X
$v3$	99
$v4$	89
$v5$	Y

(*b*)

company	*price*	(*condition*)
$v2$	X	if X < 100
$v3$	99	
$v4$	89	
$v5$	Y	if Y < 100

(*c*)

Figure 4: Conditional Tables

Tag line Using a data model for views allowing incomplete information is necessary in a Web context.

Acknowledgments We want to thank D. Suciu, J. Widom, P. Buneman, S. Cluet, T. Lahiri, A. Mendelzon, L. Mignet, J. Simeon, C. Souza, R. Topor and A.M. Vercoustre for discussions or comments on (parts of) this paper.

References

[1] S. Abiteboul. Querying semistructured data. In *Proc. Int. Conf. on Database Theory (ICDT)*, 1997.

[2] S. Abiteboul, B. Amann, S. Cluet, T. Milo, and V. Vianu. Active views for electronic commerce. In *Conférence sur les Bases de Données*, 1998. www-rocq.inria.fr/verso/ACTIVEVIEWS/paper/av.pdf.

[3] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web*. Morgan Kaufmann Publishers, San Francisco, CA, to appear, 1999.

[4] S. Abiteboul and O. Duschka. Answering queries using materialized views. In *Proc. ACM SIGMOD/SIGACT Conf. on Princ. of Database Syst. (PODS)*, 1998.

[5] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, Reading-Massachusetts, 1995.

[6] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1, 1997.

[7] S. Abiteboul and N. Spyratos. Information theoretic aspects of databases. In *Proc. ACM SIGMOD/SIGACT Conf. on Princ. of Database Syst. (PODS)*, 1983.

[8] Serge Abiteboul, Jason Mc Hugh, Michael Rys, Vassilis Vassalos, and Janet Wiener. Incremental maintenance for materialized views over semistructured data. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 1998.

[9] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 505–516, 1996.

[10] Peter Buneman. Semistructured data. In *Proc. ACM SIGMOD/SIGACT Conf. on Princ. of Database Syst. (PODS)*, 1997.

[11] R. G. Cattell. *The Object Database Standard: ODMG 2.0.* Morgan Kaufmann, 1997.

[12] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proc. IEEE Intl. Conf. on Data Engineering*, 1995.

[13] S. Chawathe, S. Abiteboul, and J. Widom. Representing and querying changes in semistructured data. In *Proc. IEEE Intl. Conf. on Data Engineering*, 1998.

[14] E. F. Codd. A relational model of data for large shared data banks. *Comm. of the ACM*, 13(6):377–387, 1970.

[15] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D Suciu. Xml-ql: A query language for xml. www.w3.org/TR/1998/NOTE-xml-ql-19980819/.

[16] The world wide web consortium (w3c)'s dom (document object model) web page. www.w3.org/DOM/.

[17] H. Garcia-Molina, W.J. Labio, and J. Yang. Expiring data in a warehouse. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 1998.

[18] Tomasz Imieliński and Jr. Lipski, Witold. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[19] T. Lahiri, S. Abiteboul, and J. Widom. Ozone: Integrating structured and semistructured data. www-db.stanford.edu/ tlahiri/ozone.pdf.

[20] A. Levy, A.O. Mendelzon, D. Srivastava, and Y. Sagiv. Answering queries using views. In *Proc. ACM SIGMOD/SIGACT Conf. on Princ. of Database Syst. (PODS)*, 1995.

[21] D. Maier. Database desiderata for an xml query language. www.w3.org/TandS/QL/QL98/pp/maier.html.

[22] 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, 1997.

[23] M.Fernandez, D.Florescu, J.Kang, A.Levy, and D.Suciu. Strudel: A web site management system. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, 1997.

[24] Y. Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object exchange across heterogeneous information sources. In *International Conference on Data Engineering*, 1995.

[25] Y. Papakonstantinou and V. Vassalos. Query rewriting using semistructured views. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, 1999.

[26] M. Soo. Bibliography on temporal databases. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 14–23, 1991.

[27] C. Souza, S. Abiteboul, and C. Delobel. Virtual schemas and bases. In *Proc. EDBT, Cambridge*, 1994.

[28] D. Suciu. An overview of semistructured data. *Database Theory Column (ed V. Vianu), Sigact News*, 29(4):28–38, 1998.

[29] J.D. Ullman. *Principles of Database and Knowledge Base Systems, Volume I.* Computer Science Press, 1988.

[30] J.D. Ullman. *Principles of Database and Knowledge Base Systems, Volume II: The New Technologies.* Computer Science Press, 1989.

[31] Special issue on workflow and extended transaction systems. Data Engineering Bulletin, 16(2), 1993.

[32] The world wide web consortium (w3c)'s xml web page. www.w3.org/XML/.

[33] Query for xml: position papers. www.w3.org/TandS/QL/QL98/pp.html.

[34] H.Z. Yang and P.-Å. Larson. Query transformation for PSJ-queries. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 1987.