Semistructured Data and XML

Dan Suciu
AT&T Labs
suciu@research.att.com

1 Introduction

Today much of the existing electronic data lies outside of database management system: it lies in structured documents like HTML and SGML, non-standard data formats, legacy systems, etc. The structure of this "non-relational" data is sometimes irregular, often unknown in advance, and even when it is known, it may change often and without notice (e.g. data on the Web).

Research on *semistructured data* has aimed at extending database management techniques to data with irregular, unknown, or often changing structure. Most of the research has focused on the logical data model and on query languages for semistructured data. Several query languages have been proposed: e.g. Lorel [AQM+97], UnQL [BDHS96], MSL [PAGM96], StruQL [FFLS97a, FFLS97b]. Some research resulted in prototypes using semistructured data for tasks like heterogeneous data integration (Tsimmis [PGMW95]), management of semistructured data (Lore [QRS+95, AQM+97, MAG+97]), Web-site management (Strudel [FFK+98]), data conversion (Yat [CDSS98]). Other research efforts have focused on schema formalisms for semistructured data [NUWC97, BDFS97, GW97, MS99, BM99], schema extraction [NAM97], optimizations [AV97, Suc96, Suc97, FS98, MW97], indexing [MWA+98, MS99]. Two excellent tutorials on semistructured data are available, by Abiteboul [Abi97] and Buneman [Bun97].

This paper argues that the research on semistructured data is receiving a new set of challenges with the advent of XML (Extensible Mark-up Language [Bos97, Con98]). This is a new standard approved by the World Wide Web Consortium that many believe will become the de facto data exchange format for the Web. XML supports the electronic exchange of machine-readable data (while HTML is designed primarily for human-readable documents). XML data shares many features of semistructured data: its structure can be irregular, is not always known ahead of time, and may change frequently and without notice. On the other hand it is easy to convert data from any source into XML which will make it attractive for organizations to "publish" their information sources in XML, and thus make them available to other XML applications on the Web.

For XML applications to reach their full potential however, we need to build the right tools to process data in this new format. Existing Web tools (browsers, search engines) are oriented toward *document operations*. For XML we need *database operations*, like data extraction, data integration, data translation, data storage. The research done so far on semistructured data may offer some solutions to the database problems posed by XML. For example the recently proposed query language for XML, called XML-QL [DFF+98b, DFF+98a] has been designed by transferring techniques from semistructured data to XML. But, as we argue in this paper, XML and its predicted applications require solutions to problems which the research on semistructured data has not yet addressed (e.g. type inference), or has not considered important (e.g. distributed evaluation), or simply hasn't found solutions yet (e.g. storage).

The paper starts with a brief tutorial on semistructured data and an overview of XML (Sec. 2 and 3). Then it compares XML with the semistructured data model, and the XML schemas (DTD's) with some schemas proposed for semistructured data (Sec. 4). In Sec. 6 the paper poses a few challenges for research on semistructured data, derived from predicted XML applications.

2 Semistructured Data

2.1 Data Model

OEM The OEM model (Object Exchange Model) is the de facto model for semistructured data and was originally introduced for the Tsimmis [PGMW95] data integration project. We follow here the presentation in Lore [QRS+95]. OEM is a graph-based, self-describing object instance model. Data is repre-

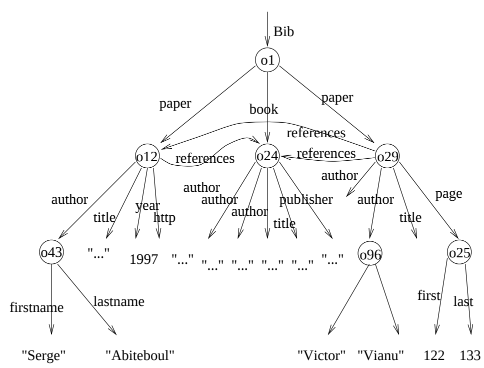

Figure 1: OEM Data

sented by a collection of *objects*. Each object can be atomic or complex. The value of an atomic object is of some base type (integer, string, image, sound, etc). The value of a complex object is a set of (attribute, object) pairs. In summary data in OEM is a *graph* in which the nodes are the objects and the edges are labeled with attribute names, and in which some leaf nodes[1] have an associated atomic value. The graph has a *root*, i.e. a distinguished object with all other objects are accessible from it.

Figure 1 illustrates an example of some OEM bibliography data, and its textual representation[2] is given in Fig. 2. Here `&o12, &o43`, etc. are *oids*: when the underlying graph is a tree, oid's can be omitted from the textual representation completely without any loss of information. For example the value of the object `&o24` can be written as:

```
{author: "Abiteboul",
 author: "Hull",
 author: "Vianu",
 title: "Foundations of Databases",
 publisher: "Addison Wesley"
}
```

Comparison to the Relational and OO Models A relation is a *set* of *records* [Ull89]. Both sets and records can be easily represented in OEM: the set $\{$`5,2,9`$\}$ can be represented as, say, $\{$`elm:5, elm:2, elm:9`$\}$, while a record `[a=5, b="seven", c=3.14]` can be represented as $\{$`a:5, b:"seven", c:3.14`$\}$. Sets are OEM objects where all attributes are the same, and records are objects where all attributes are distinct. Relational data can be easily represented in OEM. On the other hand OEM

[1]Here we use the term *leaf* as in trees, meaning a node without outgoing edges.

[2]The original OEM textual representation has a slightly different syntax.

```
Bib:&o1
{paper: &o12
  {author: &o43
    {firstname: &23 "Serge",
     lastname: &76 "Abiteboul"},
   title: &o63 "Querying semi-structured data",
   year: &o42 1997,
   references: &o24,
   http: &o91
    "http://www-rocq.inria.fr/~abitebou/pub
                  /icdt97.semistructured.ps"},
 book: &o24
  {author: &o23 "Abiteboul",
   author: &o62 "Hull",
   author: &o82 "Vianu",
   title: &o97 "Foundations of Databases",
   publisher: o22 "Addison Wesley"},
 paper: &o29
  {author: &o52 "Abiteboul",
   author: &o96 {firstname: &243 "Victor",
                 lastname: &o206 "Vianu"},
   title: &o93 "Regular path queries with constraints",
   references: &o12,
   references: &o24,
   page: &o25 {first: &o64 122, last: &o92 133}}}
```

Figure 2: Textual representation of OEM data

allows more flexibility in the data, like, for example, missing attributes in records, multiple occurrences of the same attribute, heterogeneous sets, attributes with different types in different objects, etc. One should also note that the analogy between OEM and sets has its limitations. For example, in OEM $\{\mathtt{a}:\{\mathtt{b}:1,\mathtt{c}:2\},\mathtt{a}:\{\mathtt{b}:1,\mathtt{c}:2\}\}$ is not equal to $\{\mathtt{a}:\{\mathtt{b}:1,\mathtt{c}:2,\}\}$. OEM is in this respect closer to an object-oriented model, where objects have an existence of their own.

Object Equality Like in object-oriented languages, there are two operators for testing object equality: *shallow equality* returns true iff the two are the same object, and *deep equality* returns true if the graphs accessible from the two objects are isomorphic. Of course, none of the two equalities would identify $\{\mathtt{a}:\{\mathtt{b}:1,\mathtt{c}:2\},\mathtt{a}:\{\mathtt{b}:1,\mathtt{c}:2\}\}$ with $\{\mathtt{a}:\{\mathtt{b}:1,\mathtt{c}:2,\}\}$.

The UnQL Variation on the Data Model A more radical approach is taken by the data model in UnQL [BDS95, BDHS96], in which $\{\mathtt{a}:\{\mathtt{b}:1,\mathtt{c}:2\},\mathtt{a}:\{\mathtt{b}:1,\mathtt{c}:2\}\}$ and $\{\mathtt{a}:\{\mathtt{b}:1,\mathtt{c}:2,\}\}$ *are* equal. Data is still a labeled graph, but the model explicitly states that two graphs G, G' are equal if they are *bisimilar*.

The notion of bisimulation has bee used in non-well founded set theory [Acz88] and in process algebra [Mil89], and equates two graphs when they "look the same" when traversed from their respective roots. Intuitively, two graphs are bisimilar if they become isomorphic after (1) unfolding them into (possible infinite) trees, and (2) eliminating duplicates at each node. For $\{\mathtt{a}:\{\mathtt{b}:1,\mathtt{c}:2\},\mathtt{a}:\{\mathtt{b}:1,\mathtt{c}:2\}\}$ and $\{\mathtt{a}:\{\mathtt{b}:1,\mathtt{c}:2,\}\}$ only (2) is needed, and they are bisimilar. For a more complex example, the graphs `&o1{a:{a:&o1}}` and `&o2{a:&o2}` are bisimilar, since both unfold into the infinite chain $\xrightarrow{a}\xrightarrow{a}\xrightarrow{a}\ldots$ The formal definition of bisimulation can be found in [BDS95, BDFS97].

2.2 Query Languages

Lorel LORE[QRS+95, MAG+97, AQM+97] (Lightweight Object REpository) is a general purpose data management system for semistructured data, and Lorel is its query language. The following Lorel query returns the titles of papers written since 1995:

```
select X.title
from Bib.paper X
where X.year > 1995
```

Lorel is derived from OQL, an object-oriented query language [Cat96]. Queries bind variables `X, Y, ...` to objects in the `from` clause, navigate using path expressions, test conditions in the `where` clause, and return objects in the `select` clause. There are a few differences from OQL however. The first deals with coercions. The query above will return a title if its year is `"1997"` (a string) or `1997` (an integer).

A second difference is the way missing attributes are handled. OQL generates an error if the query is applied to data missing a `year` or `title` attribute. Lorel simply ignores papers missing any of these attributes.

A third difference is that attributes can be either singletons or sets. For example if `X` has several `year` attributes, then `X` will be selected as long as at least one of them is `> 1995`.

Finally, to query data with partially unknown structure, Lorel introduces *generalized path expressions*. These extended OQL-like path expressions with wild-cards and arbitrary regular expressions. For example to find all papers referenced directly or indirectly by `Ullman` one could write:

```
select X.title
from Bib.paper X, Bib.paper Y
where Y.author.(lastname)? = "Ullman",
      Y.(reference)+ = X
```

Here ( )? denotes an optional path expression, and ( )+ denotes the strict Kleene closure. Other regular expressions are Kleene closure ( )*, concatenation ( ).( ), and alternation ( ) | ( ).

Lorel does not require a `from` clause, and has a rule stating that common paths in the query correspond to the same object (unless specified otherwise). For instance the first query above can be rewritten as:

```
select Bib.paper.title
where Bib.paper.year > 1997
```

UnQL A line of research [BBW92, BLS+94, BNTW95] has advocated that query languages be designed starting from the mathematical foundations of the language's data types. This approach is tempting to follow, especially when one faces a new, unfamiliar data model, like semistructured data: it resulted in a new query construct called *structural recursion* for semistructured data, and lead to the language UnQL (Unstructured Query Language) [BDS95, BDHS96]. Structural recursion is interesting in that it can express both *queries* and *data translations* in a single, simple formalisms. By comparison, a `select-from-where` like in Lorel is limited in the kind of translations it can express. We illustrate structural recursion next.

The first example retrieves all integers in the database:

```
(1) f(v)        = if isInt(v) then {result : v}
                  else {}
(2) f({})       = {}
(3) f({l : t}) = f(t)
(4) f(t1 U t2) = f(t1) U f(t2)
```

This is a definition of a recursive function `f` by pattern matching, with four rules. We illustrate its computation on the database `D` in Fig. 2. Here `D = {Bib : &o1}` hence rule `(3)` applies, and `f(D) = f(&o1)`. Next `&o1 = {paper:&o12, book:&o24, paper:&o29}`, which is a union `&o1 = {paper:&o12} U {book:&o24} U {paper:&o29}`: hence rule (4) applies twice, then rule (3) applies three times, and we get `f(&o12) U f(&o24) U f(&o29)`. Eventually we reach the leaves, for which rule `(1)` applies: the end result is `{result : 122, result : 133, result : 1997}`.

The second example illustrates a translation. It takes some bibliography database and makes a copy of it, converting all integers to strings in `books`, and leaving integers untouched in other publications:

```
g(v)         = v
g({})        = {}
g({l : t}) = if l = book then {book : h(t)}
               else {l : g(t)}
g(t1 U t2) = g(t1) U g(t2)

h(v)         = if isInt(v) then Int2String(v)
               else v
h({})        = {}
h({l : t}) = {l : h(t)}
h(t1 U t2) = h(t1) U h(t2)
```

Here we have two mutually recursive functions: `g` only looks for a `book` label and returns everything unchanged, while `h` converts `Int`s to `String`s and leaves everything else unchanged.

The general form of structural recursion consists in a definition of $m \geq 1$ mutually recursive functions, with certain restrictions. Each function `f` is defined by pattern matching, with four rules (as above) and must return {} on {} and `f(t1) U f(t2)` on `t1 U t2` (rules 2 and 4). In the rule `f({l:t})` it may call recursively `f(t)`, and the other functions on `t`, and use their values to construct the result. We refer the reader to the references for further details.

Structural recursion for semistructured data has a few nice properties: it has a clean semantics on graphs with cycles[3], and is computable in PTIME on any input data. One property is its potential for optimization. Since this is of special interest in practice, we illustrate it next. Assume that we have to compute the query Q(D) = f(g(D)), i.e. apply the translation `g` first, then apply `f` to the result (`f, g, h` are defined above, `D` is some bibliography database). The meaning of this query is that it returns all integers except those in `books`. A naive execution of this query would store the intermediate result `D' = g(D)`; this is inefficient since its size is comparable to that of the original data. However `Q` can be rewritten into a single traversal given by:

```
q(v)         = if isInt(v) then {result : v}
               else {}
q({})        = {}
q({l : t}) = if l = book then {} else q(t)
q(t1 U t2) = q(t1) U q(t2)
```

The new query does not need an intermediate result any more. Such a rewriting is a form of optimization: other well-known optimizations like selection pushing and join reordering can be expresses in similar ways.

[3]An operational way to deal with cycles is to *memoize* recursive calls and thus avoid infinite loops. There is a second, declarative semantics, which coincides with the operational one.

Expressing Translations with Skolem Functions A different approach to express translations of semistructured data is with Skolem Functions. Skolem Functions come from first order logic [End72], have been introduced in object-oriented systems by Maier [Mai86], and have been extensively studied in the context of deductive query languages by Hull and Yoshikawa in [HY90]. For transforming semistructured data, they have been first used by Papakonstantinou et al. [PAGM96] in the Mediator Specification Language MSL, and later in StruQL [FFLS97a, FFLS97b], the query language of the Web site management system Strudel [FFK+98] and YATL, the query language of the YAT data conversion system [CDSS98]. In all three applications *transforming* semistructured data is a central task, and Skolem functions proved a useful technique to express such translations. We illustrate Skolem functions next in the context of Strudel.

Assume we have a bibliography data like in Fig. 1 and would like to construct a web site organized as follows. (a) There is a root page, (b) it has links to persons' pages (c) from each person's page there are entries for each year when that person published, and (d) each year entry has links to the actual publications. This can be achieved with the following StruQL query:

```
where  Bib -> L -> X,  X -> "year" -> Y
       X -> "author".("lastname")? -> Z,
       isString(Z),
create RootPage(), PersonPage(Z),
       YearEntryPage(Z,Y)
link RootPage() -> "Person" -> PersonPage(Z),
     PersonPage(Z)->"YearEntry"->YearEntryPage(Z,Y),
     YearEntryPage(Z,Y) -> L -> X
```

Here `RootPage`, `PersonPage`, `YearEntryPage` are *Skolem Functions* that create new oid's. By definition a Skolem function applied to the same inputs produces the same node oid: for example if `PersonPage("Abiteboul")` is called twice, it will return the same oid. Note that `YearEntryPage(Z,Y)` needs to have two arguments since we need a new node for each author and for each year.

3 XML

XML, "Extensible Mark-up Language" [Con98] is a standard recently approved by the W3C that is widely believed to become a universal format for data exchange on the Web. XML supports electronic exchange of machine-readable data on the web, where as HTML currently supports exchange of human-readable documents.

```
<Bib>
  <paper id="o12" references="o24">
     <author>
        <firstname> Serge </firstname>
        <lastname> Abiteboul </lastname>
     </author>
     <title> Querying ss data </title>
  </paper>
  <book id="o24">
     <author> Abiteboul </author>
     <author> Hull </author>
     <author> Vianu </author>
     <title> Foundations of Data Bases </title>
     <publisher> Addison Wesley </publisher>
  </book>
  <paper id="o29", references = "o12,o24">
     ...
  </paper>
</Bib>
```

Figure 3: An Example of XML Data

XML is a strict fragment of SGML. XML is more powerful than HTML[4] in three major respects [Bos97]:

1. Users can define new tag names at will.

2. Document structures can be nested to any level.

3. Any XML document can contain an optional description of its grammar for use by applications that need to perform structural validation. The grammar portion is called DTD (Document Type Descriptor)

Data in XML is grouped into elements delimited by tags and elements can be nested. Figure 3 illustrates an example of XML data. It contains three publications (two papers and a book). The strings enclosed between < and >, like `paper, author, title, firstname, lastname`, are called *tags*. Each start-tag `<abc>` must have a matching end-tag `</abc>`, and the XML fragment between these matching tags is called an *element*. We will explain `id` and `references` below.

Data in XML is self-describing, and bears striking similarities to semistructured data. The analogy can be summarized in:

[4]However, XML is *not* a superset of HTML, since the latter does not require all tags to have a matching end tag.

Semistructured data model	XML
attribute	tag
object	element
atomic value: string int, float, video, etc	character data (CDATA) strings only

It is important to note however that XML is only a mark-up language, and does not have an associated data model. The analogy between XML (syntax) and the semistructured data model (semantics) has a few rough points, which we discuss next. For a more general discussion on text models and conventional data models we refer the reader to [RTW96].

Order XML documents are by definition ordered while the semistructured data model is unordered. It is easy to make OEM fully ordered, by having a complex object's value be an *ordered* set of (attribute, object) pairs (i.e. a *list* as opposed to a *set*). We could even consider a mixed OEM, in which some objects are ordered while others are not. The hard part is to extend query languages to deal with order. We discuss this in Sec. 6.

Attributes XML elements may contain *attributes*, as in the following example:

```
<paper>
   <title textheight = "12pt">
        This title is large
   </title>
   ...
</paper>
```

Here `textheight` is called an *attribute* (we will call it *XML-attribute* in the sequel to distinguish from the OEM-attributes), and `"12pt"` is its value. Each tag may have several XML-attribute, value pairs; values are always strings. There are two ways to capture XML-attributes in the semistructured data model. The first is to blur the distinction between XML-attributes and tags. The above data becomes:

```
<paper>
   <title>
      <textheight> "12pt" </textheight>
      <value> This title is large </value>
   </title>
   ...
</paper>
```

A possible useful variation is in [Tom89].

The second is to include XML-attributes in the OEM model by associating a set of (XML-attribute, value) pairs to each object.

References Elements can be given explicit oid's, and can refer to other elements' oids using reserved XML-attributes[5]. For example in Fig. 3 `id` is the attribute *defining* an element's oid, and `references` is an attribute *referring* to some other oid. Only three elements in Fig. 3 have explicit oids: `o12, o24, o29`. The definition of XML leaves it up to the application how interpret references, and only requires that all referred oids be defined. One possible interpretation of the XML text in Fig. 3 is, of course, the data in Fig. 1, but this is not the only choice. Under this choice, if `X` is the first `paper` (`X = o12`) then order to get the publisher `"Addison Wesley"` we have to write the path expression `X.references.publisher`, not `X.references.book.publisher`. Another possible interpretation is as a graph with labels on *nodes* rather than edges: then the correct path would be `X.references.book.publisher`.

Links XML documents can have links to elements inside other XML documents. Like HTML anchors, links can cross server boundaries. Ignoring links in the logical data model may result in huge efficiency penalties. A proposal on how to take the link cost into account is made in WebSQL [MMM96]. The authors consider three kinds of links in their data model: intra-document, inter-document but intra-site, and inter-sites. Path expressions in queries can specify the three kinds of links and say, for example, that it is OK to traverse links as long as the target document is on the same server.

4 Schemas and DTD's

Semistructured data is self-describing. That is, the schema is embedded with the data, and no a priori structure is assumed. This gives us flexibility to process (store, query, etc) any data, and we can deal with changes in the data's structure seamlessly. But this extreme view has a few drawbacks:

- Data is inefficient to store, since the schema needs to be replicated with each data item.
- Queries are hard to evaluate efficiently: even a simple regular path expression may require the entire graph to be traversed.
- Queries are hard to formulate: the user has to rely on undocumented information about the data in order to formulate relevant queries.

[5]These attributes are specified in the Document Type Descriptor.

```
<!ELEMENT Bib    (paper|book)*>
<!ELEMENT paper  (author+,title,year?,page?,http?)>
<!ELEMENT book   (author+,title,publisher?,year?)
<!ELEMENT author (\#PCDATA|firstname|lastname)*>
<!ELEMENT firstname  (\#PCDATA)>
<!ELEMENT lastname   (\#PCDATA)>
<!ELEMENT title  (\#PCDATA)>
<!ELEMENT year   (\#PCDATA)>
<!ELEMENT page   (first,last)>
<!ELEMENT http   (\#PCDATA)>
<!ELEMENT publisher (\#PCDATA)>
<!ELEMENT first  (\#PCDATA)>
<!ELEMENT last   (\#PCDATA)>
```

Figure 4: An Example of a DTD

In the applications of semistructured data one observes that the data often has some regular structure which can, and should be exploited to address the problems above. Researchers have looked for ways to describe and exploit regularities in the data's structure, without having to fully specify a rigid schema ahead of time. Proposals fall into two categories: (1) Flexible schema formalisms, which are described ahead of the data, and offer a continuous scale of how precise the data's structure is described: at an extreme, there exists a "void" schema that does not impose any structure at all. Examples are graph schemas [BDFS97], description logics [DGM98], Schema Definition Language [BM99]. (2) Inferred schemas, which are inferred automatically from the data. These are rigid, and can describe the data's structure quite accurately, and are recomputed, or incrementally updated, when the data changes. Examples are data guides [NUWC97, GW97], unary datalog types [NAM97], T-indexes [MS99].

XML comes with its own kind of schemas, called *Document Type Descriptor* (DTD). A DTD specifies, by means of regular grammars, how elements can be nested. Fig. 4 illustrates a DTD for a bibliography data like that in Fig. 3. The first line says that a `Bib` element may contain arbitrary `book` and or `paper` elements, in any order. Next a `paper` element is described by another regular expression. The order in `paper` is imposed: there are ≥ 1 `authors` followed by a `title`, then an optional `year`, etc. `#PCDATA` stands for *parsed character data*, and means a string value or valid tags[6].

As illustrated by the example, in a DTD one can fine-tune the amount of structure imposed via regular

[6]Note that the definition of `author` allows any combination of `firstname`, `lastname` elements or character data. There are restrictions on the use of `#PCDATA`: the regular expressions `#PCDATA | (firstname,lastname)` is not allowed.

expressions: we can say that some elements are optional (e.g. `year?`, `page?`, etc.), allow for multiple occurrences of an attribute (e.g. `author+`), leave order unspecified (in `(paper|book)*`). Order differentiates DTD's from the schemas considered for semistructured data (except for [BM99]). Even forgetting order, DTD's are more powerful than most schemas for semistructured data: they can say, for example, that "every `book` has an even number of `authors`".

5 A Case Study: XML-QL

XML-QL is a query language for XML designed by carrying over techniques from semistructured data to XML [DFF+98b, DFF+98a]. Its goals are to allow XML data to be queried, translated, and integrated. XML-QL incorporates many of the features of other query languages for semistructured data, and extends them to an ordered data model.

Pattern Matching XML-QL uses patterns in the `where` clause to extract data. For example:

```
where <book>
         <publisher><name>Addison-Wesley</></>
         <title> $t</>
         <author> $a</>
       </> in "www.a.b.c/bib.xml"
construct <result>
              <author> $a</>
              <title> $t</>
          </>
```

returns all (author,title) pairs of books published by Addison-Wesley. Note that `</>` is a shorthand for ending tags (`</publisher>` or `</name>` etc.) and variables are preceded by $. Patterns extend to other XML components, like XML-attributes.

Tag Variables and Regular Expressions The following query retrieves all publications by `Abiteboul`:

```
where <$p> <title> $t </title>
           <$e.lastname?> Abiteboul </>
      </> in "www.a.b.c/bib.xml",
      $e in {author, editor}
construct <$p> <title> $t </title> </>
```

Here `$p` is a *tag variable*, and will be bound to tags like `book, paper`, etc. Similarly, `$e` can be bound to either `author` or `editor`. Tags can now be regular path expressions, like `<$e.lastname?>`

Skolem Functions Since two of its intended applications are data translation and data integration, XML-QL has Skolem functions. For example the following query constructs a database of authors by integrating a bibliography source with an information source on patents.

```
where <paper> <author.lastname?> $a </>
              <title> $t </>
      </paper> in "www.a.b.c./bib.xml"
construct <person id=PersonNode($a)>
              <name> $a </>
              <papertitle> $t </>
          </person>

where <patent> <applicant> $x </>
               <description> $d </>
      </patent> in "www.patents.org/filed.xml"
construct <person id=PersonNode($a)>
              <name> $a </>
              <patenttitle> $d </>
          </person>
```

6 Challenges for Semistructured Data

XML creates new challenges for semistructured data, derived both from the mismatches described earlier and from XML's predicted applications. For some of these challenges, inspiration can be found in the techniques developed for processing tagged text, and SGML in particular, for which there exists an extensive literature (see e.g. [Loe94, BBT92, ST94, Mac91]). While some of these techniques are definitely of interest, many are limited given their restricted view of the text as a parse tree, and are ill suited for some of XML's applications (e.g. data integration). We enumerate here a possible list of research challenges, which is, by necessity, biased by the author's own taste and interests.

Order Order was considered a side issue in semistructured data, but it becomes a central problem for XML. Models for text databases do have order, but their query languages are more limited (e.g. lack joins or the ability to construct new data). We have explained earlier how OEM can be extended into an ordered data model. For the query language, this creates two problems:

- How can we inquire the order in the input ?
- How can we order the query's output ?

For the first problem, XML-QL has index variables which can, at a minimum, test the order of sub-objects. For example consider the query "find all papers published by `Abiteboul` and `Vianu` where `Vianu` is the first author":

```
where <Bib.paper> <title> $T </title>
                  <author>[$i] Vianu </>
                  <author>[$j] Abiteboul </>
                  $i < $j
      </>
construct <result> $t </>
```

Here `$i` and `$j` are *index variables* and are bound to numbers (0, 1, 2, ...) corresponding to the positions of the two `author` sub-objects. (They are not necessarily 0 and 1, because, e.g. `title`, or `year` could occur earlier.) The query selects only those titles for which `Vianu` occurs before `Abiteboul`. In the case of a mixed model, if some `paper` node is unordered, we may safely assume that `$i`, `$j` are not bound, i.e. that `paper` will not contribute to the result. Index variables are limited to single edges. It is not clear how to handle order in the case of more complex path expressions.

For the second problem, one can order the query's output with an `ordered-by` clause, like in SQL. Consider the following query in Lorel extended with an `ordered-by` clause:

```
select X.title
from Bib.paper X
where X.author.lastname = "Vianu"
ordered-by X.year
```

This lists paper titles in order of their year of publication. Papers without an `year` are not listed. In case of ties (i.e. same `year`) the order in the input data is preserved. We would like to extend this last rule to queries without `ordered-by`, and say that in that case one just preserves the order in the input. But we need to be more precise here. For example consider the following (unlikely) query:

```
select Y.author.lastname
from Bib.book X,
     X.references Y, Y.references Z
where X.publisher = "Addison-Wesley"
ordered-by Z.year
```

What does it mean for this query to "preserve the order in the input" ? We suggest a possible answer[7], which is meant to illustrate the difficulties rather than be a definite answer. Start by introducing variables in all path expressions in `select, where`, and `ordered-by`:

[7]This is also the semantics adopted by XML-QL.

```
select N
from Bib.book X,
     X.reference Y, Y.reference Z,
     Y.author.lastname N, Z.year U
where X.publisher = "Addison-Wesley"
ordered-by U
```

Fix a total order on the nodes in the input database. This order depends on the data only (not the query) and can be recomputed and stored with the data. It could be the order given by the text representation of the data, or, when no text representation is available, it could be obtained by a depth-first or breadth-first graph traversal. Use this order to sort lexicographically all bindings of the variables `(U, X, Y, Z, N)`: note the order in which the variables are listed (`U` is first, because it appears in `ordered-by`, the others are in the order mentioned by the query). Finally project on the variable `N`.

Finally, we only mention that, in the context of query languages with declarative updates, one also needs to worry about being able to insert new objects at specific places in lists.

XML Data Translation Different organizations may choose different DTD's to structure semantically related data (e.g. two purchase-order catalogs may organize data about shoes differently); in order to be exchanged, data needs to be converted. Inside one organization a DTD may evolve over time, and old data needs to be converted into the new structure. These are only two examples illustrating the need for data translation. Translation of XML data raises two problems:

- Given two DTD's $D1, D2$ and a query Q, check that all XML data instances of type $D1$ will indeed be translated into instances which of type $D2$.
- Given two DTD's how can we automate, or semi-automate the translation process ?

The first problem is a type inference problem, and has not been addressed so far in semistructured data (except in a restricted setting in [BDFS97]). The second problem has been addressed in [ACM97, MZ98].

Type checking As mentioned before type checking is important in data translation. In a broader sense it can be used to assist the user in query formulation. Given the DTD in Fig. 4, a path expression like `Bib.paper.address` doesn't make sense (since the tag `address` is not mentioned in the DTD), and the system may issue a warning to the user, rather than just return an empty result. A more subtle hint can be given for a path expression like `Bib.book.*.lastname`: here the system may warn the user that `*` can only be bound to `author`.

Optimization McHugh and Widom describe a traditional query optimizer for Lorel in [MW97]. Queries are translated into query execution plans, which use a rich set of physical operators, and these in turn are optimized.

Other researchers have considered schema-based query optimizations (a kind of semantic optimization). Assume the query condition `Bib.*.last = 130`. Without schema information, a query processor has to traverse the entire data to search for the attribute `last`. Now assume that the data's structure is described by the DTD in Fig. 4. Only a `Bib.(paper|book).page` path can lead to a `last` attribute. This offers a useful hint to the query processor: do not follow, say, `author` links. Such an optimization is called *query pruning* in [FS98], and can be achieved by a process similar to that of constructing the product automaton of two automata; it has been shown to yield a guaranteed "optimal" query (under certain assumptions). But query pruning has only been described for single path queries and a restricted kind of schemas (graph schemas): DTD's are richer, and they may interact in subtle ways with more complex queries.

Some researchers have addressed query optimization from a theoretical angle. Abiteboul and Vianu [AV97] consider *path equations*. For example, considering a database of assemblies and parts, the equation `part.(subpart)*.description = catalog.partdescription` says that all `descriptions` can be reached directly through the catalog. Then the query `part.(subpart)*.description.section.paragraph` can be replaced with the simpler `catalog.partdescription.section.paragraph`. In general, we have a set of path equations E and would like to decide whether two given path expressions $p1, p2$ are equal. The authors in [AV97] have shown the surprising result that this problem is decidable, and moreover shown that its complexity is tractable under certain restrictions. More complex queries are considered in [FLS98]: the authors show that containment and equivalence *conjunctive* queries with regular path expressions decidable, and, moreover, for a restricted case, the complexity is NP. Both results are still at a theoretical level and have not yet been applied in a practical framework.

Distributed Evaluation XML *links* extend and generalize HTML anchors: an XML document may contain links to objects residing in other XML documents. Links may cross server and organization boundaries, similarly to HTML anchors. This poses serious challenges to query evaluation.

A naive solution is of course to down-load all interlinked XML documents in a centralized site, and evaluated the query locally. This is impractical in most cases. A more sensible approach is to start evaluating the query at the root site, then ship the query to the remote site whenever a link is reached. A problem arises if the remote site is not responding, or if it doesn't accept the same query language. Assuming it does return some result, we proceed with the query evaluation at the root site until we reach another link, and so on. An improvement would be to anticipate all possible query shippings and bundle them into a single TCP connection. This approach was pursued in [Suc96] and in more depth in [Suc97]. That work shows that single path expressions can be evaluated with only two connections between the root site and all the other sites, while more complex queries can still be evaluated with a fixed number of connections, but that number may depend on the query. Those techniques are described in the general framework of semistructured data, and do not consider any schema information. By contrast XML data will most likely be described by DTD's, which can make pretty clear which data fragments reside on which sites. This information must be used in order to evaluate queries efficiently on distributed XML data.

Efficient Storage Current research prototypes using semistructured data store the data as a graph. The schema is thus stored with every data item. This is often an unnecessarily conservative approach. When a DTD is given, XML data can be stored more efficiently. One way is to store it in an object-oriented database, using a one-to-one mapping between the elements in the DTD and the classes in the object-oriented database: each element in the XML data becomes an object. Christophides et al. [CACS94] describe this method in detail, in the context of SGML and the O2 database. For the case of a text-retrieval system, this method may lead to unnecessary fragmentation of the SGML data [KKEX97, MKK96]: here the proposed solution consists in making the mapping between the objects in SGML and the database configurable. Some SGML objects are stored in textual form, thus increasing the granularity of the storage. When that object is fetched, it needs to be parsed.

A more radical approach could be to deploy data mining techniques to infer a good physical representation for a given XML data. Consider for example a data consisting of three persons:

```
<Person> <Name> <FN>John</FN>
                <LN>Smith</LN>
        </Name>
        <Office> A123 </Office>
        <Phone> x1234 </Phone>
</Person>
<Person> <Name> <FN> Joe </FN>
                <LN> Doe </LN>
         </Name>
         <Phone> x4321 </Phone>
         <Phone> x4322 </Phone>
</Person>
<Person> <Name> Baker</Name>
         <Fax> x5555 </Fax>
</Person>
```

For this particular data we may propose the following relational storage:

FN	LN	Office	Phone1	Phone2
John	Smith	A123	x1234	null
Joe	Doe	null	x4321	x4322

Name	Fax
Baker	x5555

7 Conclusions

XML is catching on rapidly. Software companies are choosing it as the meta-grammar for the data of their future Web applications. XML data will be exchanged freely between applications belonging to the same, or to different organizations, in the same way in which we now exchange freely HTML documents. Finding solutions to the technical problems generated by these applications will be, or is already, imperative. This paper has argued that research on semistructured data can have an impact on solving some these problems, and has raised a few new challenges for future research.

Acknowledgments Peter Buneman made me first think about semistructured data. Later, I was privileged to work with a number of people on semistructured data, which shaped my thinking on this topic and, hence, influenced this paper in an indirect way: Serge Abiteboul, Susan Davidson, Mary Fernandez, Daniela Florescu, Alon Levy, Tova Milo. I also thank Michael Rys, Frank Tompa, and Jennifer Widom for their comments.

References

[Abi97] Serge Abiteboul. Querying semi-structured data. In *Proceedings of the International Conference on Database Theory*, pages 1–18, Deplhi, Greece, 1997. Springer-Verlag.

[ACM97] Serge Abiteboul, Sophie Cluet, and Tova Milo. Correspondence and translation for heterogeneous data. In *ICDT*, 1997.

[Acz88] P. Aczel. *Non-Well-Founded Sets*. Number 14 in CSLI Lecture Notes. Stanford Univ Center for the Study, Stanford, 1988.

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

[AV97] Serge Abiteboul and Victor Vianu. Regular path queries with constraints. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 122–133, 1997.

[BBT92] G.E. Blake, Tim Bray, and F. Tompa. Shortening the OED: experience with a gramamr-defined database. *ACM Transactions on Information Systems*, 10(3):213–232, July 1992.

[BBW92] Val Breazu-Tannen, Peter Buneman, and Limsoon Wong. Naturally embedded query languages. In J. Biskup and R. Hull, editors, *LNCS 646: Proceedings of 4th International Conference on Database Theory, Berlin, Germany, October, 1992*, pages 140–154. Springer-Verlag, October 1992.

[BDFS97] Peter Buneman, Susan Davidson, Mary Fernandez, and Dan Suciu. Adding structure to unstructured data. In *Proceedings of the International Conference on Database Theory*, pages 336–350, Deplhi, Greece, 1997. Springer Verlag.

[BDHS96] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 505–516, 1996.

[BDS95] Peter Buneman, Susan Davidson, and Dan Suciu. Programming constructs for unstructured data. In *Proceedings of the Workshop on Database Programming Languages*, Gubbio, Italy, September 1995.

[BLS+94] P. Buneman, L. Libkin, D. Suciu, V. Tannen, and L. Wong. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, March 1994.

[BM99] Catriel Beeri and Tova Milo. Schemas for integration and translation of structured and semi-structured data. In *Proceedings of the International Conference on Database Theory*, 1999. to appear.

[BNTW95] Peter Buneman, Shamim Naqvi, Val Tannen, and Limsoon Wong. Principles of proramming with collection types. *Theoretical Computer Science*, 149:3–48, 1995.

[Bos97] John Bosak. Xml, java, and the future of the web, 1997. Available from `http://sunsite.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm`.

[Bun97] Peter Buneman. Tutorial: Semistructured data. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 117–121, 1997.

[CACS94] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In Richard Snodgrass and Marianne Winslett, editors, *Proceedings of 1994 ACM SIGMOD International Conference on Management of Data*, Minneapolis, Minnesota, May 1994.

[Cat96] R. G. G. Cattell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, San Mateo, California, 1996.

[CDSS98] S. Cluet, C. Delobel, J. Simeon, and K. Smaga. Your mediators need data conversion ! In *Proceedings ACM-SIGMOD International Conference on Management of Data*, pages 177–188, 1998.

[Con98] World Wide Web Consortium. Extensible markup language (xml) 1.0, 1998. `http://www.w3.org/TR/REC-xml`.

[DFF+98a] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Applications of xml-ql, a query language for xml, 1998. In preparation.

[DFF+98b] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Xml-ql: A query language for xml, 1998. `http://www.w3.org/TR/NOTE-xml-ql/`.

[DGM98] D.Calvanese, G.Giacomo, and M.Lenzerini. What can knowledge representation do for semi-structured data ? In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[End72] Herbert B. Enderton. *A Mathematical Introduction To Logic*. Academic Press, San Diego, 1972.

[FFK+98] Mary Fernandez, Daniela Florescu, Jaewoo Kang, Alon Levy, and Dan Suciu. Catching the boat with Strudel: experience with a web-site management system. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, 1998.

[FFLS97a] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language and processor for a web-site management system. In *Proceedings of the Workshop on Management of Semistructured Data*, 1997. Available from `http://www.research.att.com/~suciu/workshop-papers.html`.

[FFLS97b] Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3):4–11, September 1997.

[FLS98] Daniela Florescu, Alon Levy, and Dan Suciu. Query containment for conjunctive queries with regular expressions. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 139–148, 1998.

[FS98] Mary Fernandez and Dan Suciu. Optimizing regular path expressions using graph schemas. In *Proceedings of the International Conference on Data Engineering*, pages 14–23, 1998.

[GW97] Roy Goldman and Jennifer Widom. DataGuides: enabling query formulation and optimization in semistructured databases. In *Proceedings of Very Large Data Bases*, pages 436–445, September 1997.

[HY90] R. Hull and M. Yoshikawa. ILOG: Declarative creation and manipulation of object identifiers. In *Proceedings of 16th International Conference on Very Large Data Bases*, pages 455–468, 1990.

[KKEX97] K.Böhm, K.Aberer, E.Neuhold, and X.Yang. Structured document storage and refined declarative and navigational access mechanisms in HyperStorM. *VLDB Journal*, 6(4):296–311, November 1997.

[Loe94] Arjan Loeffen. Text databases: a survey of text models and systems. *SIGMOD Record*, 23(1), 1994.

[Mac91] I. A. Macleod. A query language for retrieving information from hierarchic text structures. *The Computer Journal*, 34(3), 1991.

[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.

[Mai86] D. Maier. A logic for objects. In *Proceedings of Workshop on Deductive Database and Logic Programming*, Washington, D.C., August 1986.

[Mil89] Robin Milner. *Communication and concurrency*. Prentice Hall, 1989.

[MKK96] M.Volz, K.Aberer, and K.Böhm. Applying a flexible OODBMS-IRS-Coupling to structured document handling. In *Internaltional Conference on Data Engineering*, February 1996.

[MMM96] A. Mendelzon, G. Mihaila, and T. Milo. Querying the world wide web. In *Proceedings of the Fourth Conference on Parallel and Distributed Information Systems*, Miami, Florida, December 1996.

[MS99] Tova Milo and Dan Suciu. Index structures for path expressions. In *Proceedings of the International Conference on Database Theory*, 1999. to appear.

[MW97] Jason McHugh and Jennifer Widom. Query optimization for semistructured data. Technical report, Stanford University, 1997.

[MWA+98] J. McHugh, J. Widom, S. Abiteboul, Q. Luo, and A. Rajaraman. Indexing semistructured data. Technical report, Stanford University, 1998.

[MZ98] Tova Milo and Sagit Zohar. Using schema matching to simplify heterogeneous data translation. In *VLDB*, 1998.

[NAM97] S. Nestorov, S. Abiteboul, and R. Motwani. Inferring structure in semistructured data. In *Proceedings of the Workshop on Management of Semistructured Data*, 1997. Available from `http://www.research.att.com/~suciu/workshop-papers.html`.

[NUWC97] S. Nestorov, J. Ullman, J. Wiener, and S. Chawathe. Representative objects: concise representation of semistructured, hierarchical data. In *International Conference on Data Engineering*, pages 79–90, 1997.

[PAGM96] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proceedings of Very Large Data Bases*, pages 413–424, September 1996.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *IEEE International Conference on Data Engineering*, pages 251–260, March 1995.

[QRS+95] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying semistructure heterogeneous information. In *International Conference on Deductive and Object Oriented Databases*, pages 319–344, 1995.

[RTW96] D. Raymond, F. Tompa, and D. Wood. From dat representation to data models. *Computer Standards & Interfaces*, 18:25–36, 1996.

[ST94] A. Salminen and F. W. Tompa. Pat expressions: An algebra for text search. *Acta Linguistica Hungarica*, 41(1-4):277–306, 1994.

[Suc96] Dan Suciu. Query decomposition and view maintenance for query languages for unstructured data. In *Proceedings of the International Conference on Very Large Data Bases*, pages 227–238, September 1996.

[Suc97] Dan Suciu. Distributed query evaluation on semistructured data, 1997. Available from `http://www.research.att.com/~suciu`.

[Tom89] Frank Tompa. What is (tagged) text ? In *Pro.f of 5th Conf. of U. of Waterloo Centre for the New OED*, pages 81–93, September 1989.

[Ull89] Jeffrey D. Ullman. *Principles of Database and Knowledgebase Systems I*. Computer Science Press, Rockville, MD 20850, 1989.