Araneus in the Era of XML

Giansalvatore Mecca [1] *Paolo Merialdo* [1,2] *Paolo Atzeni* [2]
mecca@dia.uniroma3.it merialdo@dia.uniroma3.it atzeni@dia.uniroma3.it
[1] D.I.F.A. - Università della Basilicata [2] D.I.A. - Università di Roma Tre

1 Introduction

A large body of research has been recently motivated by the attempt to extend database manipulation techniques to data on the Web (see [13] for a survey). Most of these research efforts – which range from the definition of Web query languages and the related optimizations, to systems for Web site development and management, and to integration techniques – started before XML was introduced, and therefore have strived for a long time to handle the highly heterogeneous nature of HTML pages. In the meanwhile, Web data sources have evolved from small, home-made collections of HTML pages into complex platforms for distributed data access and application development, and XML promises to impose itself as a more appropriate format for this new breed of Web sites. XML brings data on the Web closer to databases, since, differently from HTML, it is based on a clean distinction between the way the data, its logical structure (the DTD), and the chosen presentation (the stylesheet) are specified. By virtue of this, most of the early research proposals for data management on the Web are now being reconsidered in this new perspective (see, for a collection of references, [4]).

In this paper, we discuss the impact of XML on the research work conducted in the last few years by our group in the framework of the ARANEUS project. ARANEUS started as an attempt to investigate the chances of re-applying traditional database concepts and abstractions, such as the ones of data-model and query language, to data on the Web. In this spirit, we have developed several tools and techniques to handle both structured and semistructured data, in the Web style, as follows: (i) a data model called ADM for modeling Web documents and hypertexts [8]; (ii) languages for wrapping [12, 16, 14] and querying [8, 17] Web sites; (iii) tools and techniques for Web site design [9] and implementation [18].

An interesting question is how these tools and applications will work in the era of XML. In the following sections, we will try to answer to this question, by emphasizing how XML fits in this framework, how it is influencing our ideas and our way of thinking about Web data sources, and the design choices needed to adapt our tools – originally conceived for HTML – to the management of XML data. However, a word of caution is needed here: although very popular, XML is still a new proposal, and there are very little (if any) real XML-based applications on the Web. It is therefore quite difficult to reason about the data management problems that will come with XML, since we still haven't experienced them. Because of this, our development will be mainly informal; we will basically try to discuss some of the choices and tradeoffs related to modeling (in Section 2), querying (in Section 3), and developing (in Section 4) XML data sources.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

2 ADM as a Model for XML

The ARANEUS Data Model (ADM) [8] was introduced as a tool to give an intensional description of Web sites, in the spirit of databases. The model allows to describe the organization of data in Web pages by abstracting the logical features with respect to the physical HTML code. The fundamental modeling primitives of the model have a natural counterpart in the ones that are typically offered by object-database systems, the main differences being the absence of hierarchies and inheritance, and the presence of union types. Thus, ADM modeling primitives might somehow be considered as a subset of ODMG and SQL3 data models, enriched with union types.

In ADM each page is seen as a complex object, with an identifier, the URL, and a set of attributes. Pages sharing the same structure are grouped in *page-schemes* or *page-types*. Attributes in page-schemes have a type, which can be either simple, i.e. mono-valued, or multi-valued. Simple types are `TEXT`, `IMAGE`, and `LINK` (plus other popular MIME types). Links may be typed or untyped; in the latter case, the destination of a link need not to belong to a specified page-type; this is very important when modeling Web sites, since it is often the case that homogeneous links reach pages of different structure and functions. Complex attributes are built using the following type constructors: (i) *tuples*; (ii) *union types*, i.e., disjunctions of attributes; (iii) *lists*, i.e., ordered collections of tuples (possibly nested).[1] A site scheme is a collection of page-types, one for each class of pages in the site, connected by links.

In essence, ADM is neither a conventional, heavy-weight object-model, nor a light-weight semistructured model; it can rather be considered as a *middle-weight* data model, in the sense that it allows to capture structure when this is present, but at the same time offers flexibility in modeling heterogeneities and irregularities using union types and untyped links. In our experiences, the model has proven quite successful in the description of large and fairly well-structured HTML sites, both for site querying and for site development purposes.

When comparing the model to XML, we should keep in mind that XML has its own formalism for describing a document structure, i.e., the DTD. However, XML DTDs do mix together both a description of the logical features of the document, with rather "physical" ones (such as, for example, entities, which can be more appropriately considered as storage units or macros rather than structural components); in fact, XML is rather a format than a data model. If we concentrate on the logical structure, it is possible to see that ADM nicely abstracts the modeling primitives present in XML DTDs, i.e., we can easily associate a database type with a DTD, and a database object with an XML document.

Consider for example a hypothetical XML repository about books, and books reviews, organized according to the DTDs in Figure 1. We have a document containing a list of books (an entry point), pointing to one document for each book; for each book, we have authors, title, price, plus an optional link to a document containing a list of reviews; each review may be either from a customer or from a professional; in the latter case, the reviewer name is provided, and an affiliation (for space reasons, links have been specified simply as a macro `%Xlink`, which we assume corresponds to the full definition of a link in the XLink [5] syntax). These DTDs specify the document structure in terms of a number of elements, i.e., attributes for the associated documents; attributes may be optional, and have a type; this can be either simple or complex; simple type elements are essentially strings (`#PCDATA`) or blobs (`ANY`), or XLinks to other XML documents, like in `<!ATTLIST ToBook %Xlink>`; note that, although there are ways to constrain the destination of an XLink, these are in general unconstrained, i.e., untyped. Complex attributes are based on a limited number of primitives, as follows: (i) *structures*, i.e., typed tuples of elements, like in `<!ELEMENT Book (Author+, Title, Price, ToReviews?)>`; (ii) *union types*, i.e., disjunctions of elements, like in `<!ELEMENT Review (CustReview|ProfReview)>`; (iii) *lists*, i.e., ordered collections of elements (possibly nested) like in `<!ELEMENT BookList (Book+)>`.

It can be seen how these fundamental primitives closely correspond to the ones that are offered by ADM. Figure 1 shows a graphical representation of an ADM scheme corresponding to example above. In the scheme, "stacks" are used to represent classes of documents; edges denote links. Figure 1 also contains an explanation of

[1] The model also offers some rather Web-specific constructs, such as forms and maps, which are outside the scope of this paper.

```
<!DOCTYPE BookList [
  <!ELEMENT BookList (Book+)>
  <!ELEMENT Book     (Title)>
    <!ATTLIST  ToBook %Xlink>
  <!ELEMENT Title    (#PCDATA)> ]>

<!DOCTYPE Book [
  <!ELEMENT Book (Author+, Title, Price,
                  ToReviews?)>
  <!ELEMENT Author   (#PCDATA)>
  <!ELEMENT Title    (#PCDATA)>
  <!ELEMENT Price    (#PCDATA)>
    <!ATTLIST Price   currency (#PCDATA)>
  <!ELEMENT ToReviews EMPTY>
    <!ATTLIST ToReviews %Xlink>]>

<!DOCTYPE BookReviews [
  <!ELEMENT Title  (#PCDATA)>
  <!ELEMENT BookReviews (Title, Review+)>
  <!ELEMENT Review (CustReview|ProfReview)>
  <!ELEMENT CustReview  (#PCDATA)>
  <!ELEMENT ProfReview  (RevName, Body,
                         Affiliation)
  <!ELEMENT RevName     (#PCDATA)>
  <!ELEMENT Body        (#PCDATA)>
  <!ELEMENT Affiliation (#PCDATA)>]>
```

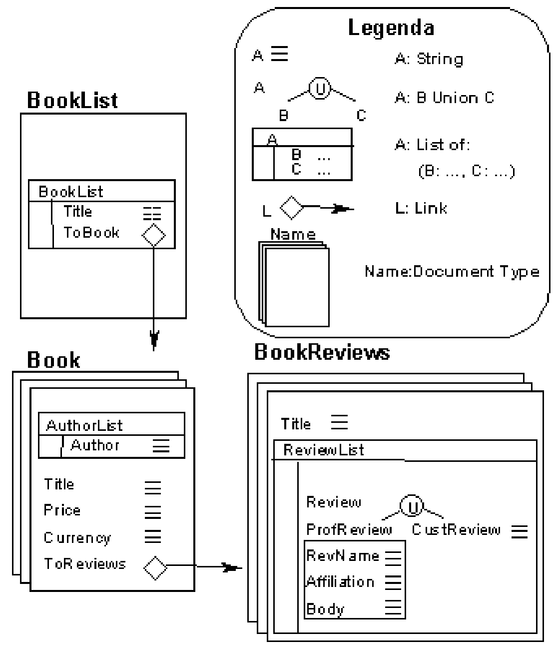

Figure 1: A Sample ADM Scheme

the other graphical primitives.[2]

These ideas show how ADM – and more generally complex-object models with union types and untyped links – provides a natural framework for describing XML repositories and their DTDs, and that we can use the ADM abstraction of DTDs as a description of the repository structure. This holds regardless of the specific kind of application we need to develop on the data, i.e., both in the case in which we need to query an existing data source and want to derive an *a posteriori* high-level description to be used as a basis for the query process, and in the case in which we are designing and developing our own XML data source and prefer to reason at a higher, abstract level, instead of directly working with the XML code. These two classes of applications are discussed in the following Sections.

3 Querying XML Data Sources

In our approach, once a target site has been identified for querying purposes, the ADM description of the site is first derived on the basis of a "reverse engineering" process, in which sets of homogeneous pages in the site are classified into page-schemes. Then, for typical HTML data sources, appropriate wrappers need to be written in order to build, from HTML pages in the site, an internal representation of data as instances of the data model. This process, which in general is rather delicate and demanding, can either be conducted by manually coding the wrappers, or with the help of algorithms and tools for automatic wrapper generation. The ADM description of the site and the corresponding wrappers are then used as a basis for the site querying process.

Consider now the case in which we need to query an XML data source instead of an HTML one. As it is reasonable, we assume that the data source is in general composed of several XML documents, linked using

[2]To see how the model abstracts with respect to the physical features in the DTD, consider for example attribute currency of element price in DTD Book; although in the DTD it was considered rather as a metadata, and therefore stored in a slightly different way with respect to elements (such as the title, for example), in the database abstraction of the DTD we may decide to ignore this difference, and model it as a conventional attribute; the same happens in our example for links; in this way, the model provides a uniform view and query mechanism for data and metadata.

XLinks. As discussed above, ADM may, in principle, model the structure of these sources based on their DTDs. There are however a number of subtleties in this modeling, and its effectiveness is strongly related to the nature of the data, as we discuss in the following paragraphs.

DTDs are Optional Our modeling approach is heavily based on the mapping of DTDs to ADM types. It is well known that DTDs are not mandatory. They are explicitly referenced in *valid* XML documents, and are used by an XML parser to validate the document itself. Other documents, called *well-formed*, do not have an explicitly associated DTD. Due to this freedom, it is still unclear to what extent DTDs will be adopted in XML applications; some consider DTDs useless; on the other side, there is a growing interest by major software vendors in defining standard classes of DTDs to be used, for example, in e-commerce applications to ease data-exchange and integration [2].

However, whatever the usage of DTDs will be, we would like to note that, given the special tagging format of XML documents, a (minimal) DTD for which the document is valid can be derived by a simple syntactic check of the document.[3] We can thus say that any XML document has an associated DTD (either explicit or implicit) describing its content. In the following, we will assume that DTDs are available and use ADM to abstract their logical structure.[4]

DTDs are not Database Types It is true that DTDs are not exactly what we usually consider a type. They have been introduced by the document community as a means for specifying the document structure from the parser viewpoint rather than from the DBMS one. As a consequence, there are several shortcomings associated with their use. For example, elements are ordered in DTDs, and, although this order may in some cases be relevant, it becomes immaterial once the corresponding database type is built (as noted in [19]). Also, elements may in some cases mix free text with subelements, thus complicating the modeling. There are in fact more elegant and elaborated proposals for superimposing schemes on XML data that are now coming from the database community (see, for example, [5]). However, these proposals are still at a preliminary stage, and their success in the XML community is still unclear.

In the meanwhile, DTDs represent a reasonable compromise. DTDs have been used for a long time in text databases as a schema definition formalism for documents (with reference to SGML, of which XML is a derivative; see, for example, [15, 7, 10, 11]). These works mainly concentrate on querying a single SGML document at a time, or several documents from the same DTD, by loading them in a database, and do not consider collections of linked documents coming from different DTDs that are to be queried remotely through the network. Still, such a large body of work shows that it is possible to build quite effective mappings from the DTD structure to a database structure, and adopt the latter as a schema for the data. This approach is also advocated by some of the major database vendors (e.g., Oracle [6]) in their support for XML data.

How (Semi)Structured is XML ? A nice feature of XML is that DTDs also act as natural wrappers for the documents; in fact, they specify a grammar that can be used to parse the documents looking for attribute values. Hence, we do not need to generate wrappers as it was needed for HTML sites, and, once the ADM scheme has been constructed based on the DTDs, it is in principle possible to run queries on the source.

However, there are peculiarities of XML data that depart from the traditional database framework, and impose to re-consider some aspects of the querying process. It can be seen that the approach we have discussed so far is more effective on fairly well-structured sources, and degrades as soon as the data source becomes less structured. It is very likely that, in the same way as it happens for current HTML Web sites, XML repositories available on the Web will range from very structured and regular to highly unstructured (with a whole degree of variants in

[3] This is, in essence, what a browser does when it parses a well-formed document.

[4] Based on these ideas, we have built a tool that automatically associates an ADM scheme with an XML repository, using the existing DTDs and inferring the missing ones.

between). The latter case will most probably happen in all applications in which the XML data source to model and query is remote and autonomous, i.e., a collection of linked documents available on the Web, completely beyond the control of the query system.

In the case or fairly structured repositories, it is possible to use query languages and optimization techniques in the spirit of databases. It is, for example, possible to load the whole repository in an object database according to the ADM types associated with DTDs[5] and query it using SQL. Or – opposite to this "warehousing" approach – in all cases in which freshness of query answers is a requirement, it is possible to extract data on-line from the repository by navigating it through the network. To this end, we have developed a query algebra, called ULIXES [9], for navigating and querying data sources on the Web; queries are written using path-expressions on the ADM scheme of the source: based on the specified path, ULIXES navigates the source starting from some entry-point (whose URL is known to the system) and materializes the query result in a local database. To give one example, suppose that we are interested in finding all books in the repository above having at least one review written by Codd. Based on the scheme in Figure 1, the query can be written in ULIXES as follows[6]:

```
CREATE VIEW CoddBooks (Title, Price)
OVER         http://books.bla.com
AS SELECT Book.Title, Book.Price
   FROM    BookList.ToBook -> Book.ToReviews -> BookReviews
   WHERE   BookReviews.ReviewList.ProfReview.Name LIKE '%Codd%'
```

There are a couple of important points that we want to emphasize. When applicable, this approach has several advantages that are due to the availability of a concise description of the repository structure (in the spirit of a database scheme) and of a query algebra based on that. In fact, this allows to specify more accurate queries, and to apply optimization techniques to make query evaluation more efficient. However, we do not require that the data source is loaded in the database before actually querying it. ULIXES is capable of navigating it remotely and materializing in the database only the portions of interest. Therefore, the XML repository is not a conventional database, and traditional query evaluation techniques need to be reconsidered in this respect [17].

If, on the contrary, the degree of structure in the repository is relatively low, or the scheme tends to evolve quickly, the techniques above will be quite ineffective. In these cases query languages developed for semistructured data, which completely ignore the schema, are to be considered more appropriate (we don't have enough space here to elaborate on the various query languages inspired on semistructured that have been proposed for XML, and refer the reader to [4]).

We therefore envision a scenario in which different XML applications will require the adoption of different data models and query languages, going from very structured to semistructured. A key problem, in this respect, is that of choosing the right approach based on the level of "structuredness" of the target data source. However, a nice feature of XML – which represents a fundamental difference with respect, for example, to HTML – is that it allows to precisely *measure* the level of structuredness of a data source, as follows.

Measuring Regularity and Stability in XML Data There may be two different sources of semistructuredness in XML repositories. The first one is due to low regularity in the structure, the second one to high chances that the structure evolves from time to time. In both cases it is still possible to build an ADM scheme of the repository (possibly inferring the missing DTDs). However, in the case of quite irregular repositories, the number of different DTDs will be large, and therefore the ADM scheme will be large. To see this, consider the book example above: suppose that in a less regular version of the data two (or more) different DTDs are used for books (for instance, `Book` for ordinary books, `OldBook` for old and collectable books, and `NewBook` for books to be published, all with slightly different sets of elements); then, when reconstructing the structure, these would be modeled as

[5] In this respect, some mechanism for modeling union types in conventional databases is needed.

[6] With respect to the version of the language presented in [8], in the last version of the prototype we have changed the syntax to make it more similar to SQL3.

separate page-types, and union types would be introduced for each link towards books. As an extreme, in some cases the number of different types may be in the order of the number of documents. We can in this case measure the *level of regularity* in the repository by the ratio between the size of the scheme (i.e., the number of types and the number of attributes inside types) and the size of the data (i.e., the number of documents).

Evolution of the repository structure (i.e., changes in the existing DTDs or use of new DTDs) of the repository are equally important, since they may require to change the database schema. If the DTDs change, the ADM schema also changes from one repository analysis to the next one. Also in this case we can measure the level of stability in the repository in a given period of time, for example by using the ratio between the size of the schema, and the number of types that have changed, or the speed of change (number of changes over time).

To summarize, we foresee that XML will be used in data-intensive applications of very different nature; in some of these, the data will be very structured and stable, in others quite semistructured or unstable; these different classes will require the adoption of different query languages, which can be based on the schema or not. Once a target XML repository has been selected, the measures of regularity and stability discussed above can be used as a metric and provide a criterion to choose the right approach to query the repository. However, more work and experiments on actual XML data are needed to establish the right tresholds for this choice.

4 Developing XML Data Sources

Beside accessing existing Web data sources, another important class of applications is devoted to developing new ones (see [13] for a survey). This is even more important in the case of XML, since we are witnessing a growing interest in its adoption, but there are very little available data yet. In this section, we discuss how techniques we have developed for HTML sites have been extended to generate XML repositories as well.

Model-Based Development in ARANEUS Differently from market tools – which are mostly rather low-level tools, either oriented to pure-HTML development or to procedural and SQL programming – we have adopted a *model-based approach* to Web site development, in the sense that our approach heavily relies on the adoption of high-level models, both at the conceptual and logical level, for designing and developing sites at all levels of the site design process, namely designing data, hypertext and presentation.

To leverage robust and wide-spread technology, we adopt a relational database as a back-end for the site; as a consequence, data to be published in the site is described at the conceptual level using the well-established Entity-Relationship Model, and at the logical level using relational tables.

The hypertext structure is described using ADM; consider again Figure 1; suppose now we are designing a book site from scratch and want to develop it based on an underlying database – that is, we are now forward-engineering the site, instead of reverse-engineering it as discussed in the previous section; a crucial step would be to specify how the target hypertext is to be organized; it can be seen how ADM is particularly effective in giving a concise description of the site hypertext.

Finally, based on ADM, we also have a precise notion of *style* for describing the graphical layout of data items in a page, and an associated tool, TELEMACO [18] for designing styles. Assuming that we have chosen the scheme in Figure 1 as a scheme for the target site, styles are used to associate a graphical layout with data items in pages. An attribute style is made of two arbitrary pieces of HTML code, between which the attribute values will be enclosed when generating pages. Consider the book title in the example above; a simple style for it might be: `NAME: [<FONT COLOR="red"><B>] [</B></FONT>]`. In this way, when the actual HTML page is generated, the book title – say "*The HTML Sourcebook*" – a piece of HTML code of the form: `<FONT COLOR="red"><B>The HTML Sourcebook</B></FONT>` will be produced.[7]

[7] The styling mechanism provided by TELEMACO is in fact more sophisticated. For example, it offers rapid prototyping facilities, and allows to work with sample HTML pages, from which it generates styles, instead of directly writing style code. We do not elaborate on this here for space reasons.

These models fit into a larger methodological framework [9], in which designers start from a conceptual description of the site domain (an Entity-Relationship scheme) and through a set of precise steps progressively moves to database logical design (this produces the database relational scheme), then hypertext design (this produces the site ADM scheme), and finally presentation design (producing page-styles). This process is assisted by HOMER [18], a CASE tool conceived to simplify this design process, and to automatically produce the code for page generation based on the site design artifacts. One of the main advantages of this model-based approach is that the overall process is in essence independent from the actual tool chosen for page generation; in fact, in its current version, HOMER may either generate code for our own tool for Web site generation, called PENELOPE [8], or Java Server Pages Templates [3]; however, it could easily be extended to generate code for any of the major Web database gateways. Another important advantage of working with high-level models such as ADM and TELEMACO styles is that the site generation phase is independent from the chosen format, HTML or XML. We have in fact re-generated most of the sites originally developed in HTML using XML, as discussed in the following paragraphs.

From HTML Sites to XML Repositories In our approach, each page is seen as an instance of an ADM type, i.e., as a URL-identified nested object. When the page is generated, as a first step the relevant data is extracted from the database and it is nested into the corresponding ADM object; then, as a second step, the actual HTML code is produced, as specified by the corresponding style. Based on the close correspondence between ADM and DTDs, we have easily adapted our tools in order to produce an XML document (possibly with XLinks) from the ADM object — by enclosing each attribute value between the corresponding element tags — and the associated DTD from the ADM page-type. As a result of this step, the system produces a collection of linked XML documents and their DTDs.[8]

XSL in Action As such, the repository could be used for exchange purposes, but it is not browsable. To make it browsable we need to encode the chosen TELEMACO style using one of the stylesheet formalisms that have been proposed for XML. After some experiments, we decided to adopt XSL [5], the "Extensible Stylesheet Language", and most of our experiences were done using Internet Explorer 5, one of the first browsers equipped with XML and XSL processors.[9]

XSL is a full-fledged functional programming language for tree-like structures like XML documents that can be used to restructure them at will. In fact, the straightforward way of associating a browsable layout to an XML document using XSL is to "restructure" it into HTML code. In essence, a stylesheet associated with an XML document is a collection of definitions (called "templates"), that specify how to recurse down the XML tree and what output to produce when a certain element is encountered. This output will go to the browser, which will display it on the screen. To give an example, suppose in documents about books we have an element `<Title>`, like in `<Title>The HTML Sourcebook</Title>`; then we may have a template in the stylesheet that specifies that, whenever element `<Title>` is encountered, a piece of HTML code must be produced by enclosing the corresponding value between certain HTML tags, like, for example, in `<FONT COLOR="red"><B>The HTML Sourcebook</B></FONT>`.

[8]There are several extended features of XML that currently the system does not handle; for example, we do not use XML element attributes, as the distinction between XML elements and their attributes is still unclear to us; we generate IDs and IDREFs, but only to a limited extent; we do not generate extended XLinks, neither XPointers.

[9]There are two main contenders for associating a presentation with XML data and make the documents browsable. One is XSL, currently still at the stage of a proposed W3C recommendation. The other one, already a W3C standard, is CSS [5], for "Cascading Style Sheets"; conceived originally for HTML, it allows to associate with each tag in a document (and therefore with each XML element) a set of layout properties, like, for example, the font size and face or the paragraph alignment. While we write, there is still some discussion about what should be the preferred format for styling XML documents. The two major browsers have chosen different routes: Internet Explorer version 5 has adopted the more flexible XSL, whereas Gecko (code name for Netscape 5) only supports the more stable CSS. Note that CSS, in our experience, may be profitably used in conjunction with XSL, by embedding it into the HTML code generated by XSL to refine the layout of the HTML tags. We have done this in some of our experiments.

It can be seen that, like ADM objects can be straightforwardly mapped to XML documents, similarly TELEMACO styles can be easily implemented as XSL stylesheets. As an outcome, our system can easily generate both plain HTML sites or XML sites with XSL stylesheets. Although in principle completely transparent, this operation has several subtleties. This is due to the fact that the XSL processor is quite strict in enforcing correctness of the HTML code it produces: it is in fact designed to produce XHTML code, the reformulation of HTML 4.0 in XML. As a consequence, some typical errors HTML developers do – like opening and not closing tags like `<P>` or `<TD>` – are not allowed, and produce run-time errors when the page is displayed.[10] As a consequence, a certain effort was needed in order to polish the HTML code in styles, before being able to use XSL.

Some of the XML sites we have generated, including some portions of the SIGMOD On Line Web site (`http://www.acm.org/sigmod`), are available on the ARANEUS Project Web site [1].

References

[1] The Araneus Project Web site. `http://www.dia.uniroma3.it/Araneus`.

[2] The Biztalk Web site. `http://www.biztalk.org`.

[3] Java Server Pages (JSP) home page. `http://www.java.sun.com/products/jsp/`.

[4] The W3C Query Languages Workshop, 1998 `http://www.w3.org/tands/ql/ql98/`.

[5] The W3C Technical Reports W3C Working Draft, March 1998. `http://www.w3.org/TR/`.

[6] Xml support in Oracle8i and beyond, 1999. `http://www.oracle.com/xml/documents/xml_twp`.

[7] S. Abiteboul, S. Cluet, V. Christophides, T. Milo, G. Moerkotte, and J. Siméon. Querying documents in object databases. *Journal of Digital Libraries*, 1(1):5–19, April 1997.

[8] P. Atzeni, G. Mecca, and P. Merialdo. To Weave the Web. In *VLDB'97*.

[9] P. Atzeni, G. Mecca, and P. Merialdo. Design and maintenance of data-intensive Web sites. In *EDBT'98*.

[10] G. E. Blake, M. P. Consens, P. Kilpeläinen, P. Larson, T. Snider, and F. W. Tompa. Text/relational database management systems: Harmonizing SQL and SGML. In *ADB'94, LNCS 819*, Springer-Verlag, June 1994.

[11] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *SIGMOD'94*.

[12] V. Crescenzi and G. Mecca. Grammars have exceptions. *Information Systems*, 23(8):539–565, 1998.

[13] D. Florescu, A. Levy, and A. O. Mendelzon. Database techniques for the world wide web: A survey. *Sigmod Record*, 27(3):59–74, 1998.

[14] S. Grumbach and G. Mecca. In search of the lost schema. In *ICDT'99*.

[15] P. Kilpeläinen, G. Lindén, H. Mannila, and E. Nikunen. A structured document database system. In *Intern. Conf. on Electronic Publishing, Document Manipulation and Typography (EP'90)*, pages 139–151, 1990.

[16] G. Mecca and P. Atzeni. Cut and Paste. *Journal of Computing and System Sciences*, 58, 1999.

[17] G. Mecca, A. Mendelzon, and P. Merialdo. Efficient queries over Web views. In *EDBT'98*.

[18] G. Mecca, P. Merialdo, P. Atzeni, and V. Crescenzi. The (Short) ARANEUS Guide to Web-Site Development. In *WebDB'99*.

[19] D. Suciu. Managing Web data, 1999. Tutorial given at SIGMOD'99.

[10] Also, empty HTML tags – like `<BR>` – need to be specified as `<BR/>`; entities, like ` ` must become `#032` etc.