IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com
ITB8329

Chapter IX

Web Retrieval of XML Documents: Practice and Challenges

Barbara Catania
University of Genoa, Italy

Elena Ferrari
University of Insubria, Italy

ABSTRACT

Web is characterized by a huge amount of very heterogeneous data sources, that differ both in media support and format representation. In this scenario, there is the need of an integrating approach for querying heterogeneous Web documents. To this purpose, XML can play an important role since it is becoming a standard for data representation and exchange over the Web. Due to its flexibility, XML is currently being used as an interface language over the Web, by which (part of) document sources are represented and exported. Under this assumption, the problem of querying heterogeneous sources can be reduced to the problem of querying XML data sources.

In this chapter, we first survey the most relevant query languages for XML data proposed both by the scientific community and by standardization committees, e.g., W3C, mainly focusing on their expressive power. Then, we investigate how typical Information Retrieval concepts, such as ranking, similarity-based search, and profile-based search, can be applied to XML query languages. Commercial products based on the considered approaches are then briefly surveyed. Finally, we conclude the chapter by providing an overview of the most promising research trends in the fields.

This chapter appears in the book, *Web-Enabled Systems Integration: Practice and Challenges* edited by Ajantha Dahanayake and Waltraud Gerhardt.

INTRODUCTION

One of the most important characteristics of the Web is that it makes a huge amount of information available to users scattered all over the world. The benefit of having this information available is, however, related to the availability of suitable techniques and tools for querying Web contents. In this respect, one of the main issues is related to the heterogeneity of Web data sources, that differ both in media support and format representation. Web data sources can store either traditional structured data, or unstructured or semi-structured information. In this scenario, there is thus a strong need for an integrating approach for querying heterogeneous Web documents.

To this purpose, XML (eXtensible Markup Language) (WWW-XML) can play a key role. XML is currently the most relevant standardization effort in the area of document representation through markup languages and is rapidly becoming a standard for data representation and exchange over the Web. Like HTML, XML is a markup language. However, it supports a richer set of features, such as user-defined tags, that allow one to represent within a single document both data and descriptive information about data, maintaining at the same time presentation aspects decoupled from data representation. Another important characteristic of XML is that it can define application specific document types through the use of Document Type Definitions (DTDs) or XML Schemas (WWW-XML Schema). Due to its flexibility, XML is currently being used as an interface language over the Web, by which (part of) document sources are represented and exported. Under this assumption, the problem of querying heterogeneous sources can be reduced to the problem of querying XML data sources.

Several query languages for XML data have been proposed so far (Bonifati & Ceri, 2000; Fernandez et al.). Most of the proposed languages come from the database community and thus are greatly inspired by standard database query languages (i.e., SQL and OQL). As such, these languages have been mainly designed to express *exact queries*, that is, queries in which the characteristics of the result are fully specified. Additionally, the result of a query contains the documents that fully satisfy the query, whereas all the other documents, included those that partially satisfy the query, are discarded.

However, people searching the Web usually have different requirements with respect to traditional DBMS users, and this requires a re-visiting of the philosophy underlying DBMS query languages. Usually, one of the main problems in the Web environment is that of finding the "right information" among a large amount of useless information, or the information that best matches the user needs. Thus, we must be able to process partial answers quickly and be able to relax a query when its results are too small. Additionally,

we must be able to provide support for incomplete and approximate answers. On the other hand, since we are considering millions of documents, a particular attention has to be devoted to optimization techniques able to make query processing efficient.

An alternative approach to face the problem of querying XML documents, that has received less attention to date, is to view XML documents as a collection of text documents with additional tags and relations between these tags. Information Retrieval (IR) techniques have traditionally been applied to search large sets of textual data and provide some of the characteristics that a query language for Web users must possess, such as the specification and management of approximate queries, the ranking of the results with respect to the degree of matching with the query and the use of profile information during query processing.

Thus, integrating IR and XML search techniques will enable more sophisticated search on the structure as well as on the content of XML documents. We believe that, due to the nature of XML documents, what is needed is a mix of database and IR characteristics. This chapter is thus mainly devoted to investigate how typical IR concepts, such as ranking, approximate queries, similarity-based and profile-based search, can be applied and integrated with XML query languages.

In the remainder of the chapter, we first briefly summarize the main characteristics of XML, for those readers which are not familiar with the language. Then, we survey the most relevant query languages for XML data proposed so far by both the scientific community and standardization committees, e.g,. W3C. In describing the query language proposals, we mainly focus on expressive power and impact in commercial products. The second part of the chapter is devoted to illustrate the main proposals related to the IR approach for querying XML documents. In this part of the chapter, we analyse both research and commercial proposals. The analysis will then help us in identifying future trends in the field and in understanding how typical IR concepts, such as ranking, similarity-based search and profile-based search, can be applied to XML documents and integrated with existing XML query languages.

A BRIEF INTRODUCTION TO XML

XML (WWW-XML) is a text-based markup language similar to SGML, which has been defined for clearly separating structure and representation issues in Web documents. This approach has the main advantage that an XML document, differently from an HTML document, can be written once and

visualized in several ways. Text is enclosed in *start tags* and *end tags* for markup, and the *tag name* provides information on the kind of *content* enclosed, thus it represents an element we would like to represent inside our document. The content of an element can be composed of other elements (tags can be nested) or it may correspond to plain text. Each tag can also be associated with some attributes, further describing the meaning of the corresponding element.

All XML documents have to be *well-formed*, that is, the nesting of elements must be correct. Moreover, an XML document can be associated with a *Document Type Definition (DTD)* or an *XML Schema*, defining the syntax of a set of XML documents with similar structure. An XML document is *valid* if it conforms to the corresponding DTD/Schema. An example of an XML document, modelling a sales order, is given in Figure 1. In particular, for each sales order, the document records the identifier (modelled through the *SONumber* element) and the date of the order (i.e., *OrderDate* element). Additionally, the document keeps track of the information on the customer (*Customer* element). For each customer, we record his/her identifier, name and address. Thus, the *Customer* element is an example of nested element,

Figure 1: An example of a data-centric XML document

```
<SalesOrder>
 <SONumber> “12345” </ SONumber >
  <Customer CustNumber = “543” >
       <CustName>ABC Industries</CustName>
        <Street>123 Main St.</Street>
       <City>Chicago</City>
        <State>IL</State>
        <PostCode>60609</PostCode>
 </Customer>
 <OrderDate>11012001</OrderDate>
 <Item ItemNumber = “1”>
     <Description> PC Monitor </Description>
      <Price>500</Price>
     <Quantity>10</Quantity>
 </Item>
 <Item ItemNumber = “2”>
     <Description>  RAM 256 Mb, one-year guarantee</Description>
      <Price>25</Price>
     <Quantity>5</Quantity>
 </Item>
</SalesOrder>
```

since it consists of five sub-elements. Finally, the XML document in Figure 1 maintains information on the items to which the order refers to. For each item, we maintain information on its identifier, price and quantity, as well as a brief description. Figure 2 shows a DTD for the document in Figure 1.

Alternatively it is possible to give a graph representation of an XML document, where nodes represent elements and attributes, and edges represent relationships between them. More precisely, there is an edge from a node n_1 to a node n_2 if either n_2 is a direct sub-element of n_1, or n_2 is an attribute of element n_1.

As pointed out in Bourret, XML documents can be seen under two different points of view:

- As *transfer vehicles*: in this case, XML documents are seen as containers for data to be transferred over the Web. These documents are called *data-centric*, since their meaning depends only on the structured data represented inside them. Typically, data-centric documents are characterized by a quite regular structure and a homogeneous content. Applications that may benefit by such an approach are typical business-to-business applications, such as buyer-supplier trading automation, inventory database access and sharing, integration of commercial transactions and workflow.
- As *interesting objects*: in this case, XML documents are seen as new data objects, to be stored and managed inside the system. These documents are called *text- centric*, since their meaning depends on the document as a whole. In this case, the structure is more irregular and data are heterogeneous. Examples of document-centric XML documents are books, emails and any XHTML document (a combination of XML and HTML, able to model both structure and presentation layout (WWW-XHTML)). Typical applications that may benefit by such an approach are

Figure 2: A DTD for the document in Figure 1

```
<!ELEMENT SalesOrder (SONumber, Customer, OrderDate, Item+)>
<!ELEMENT SONumber (#PCDATA)>
<!ELEMENT Customer (CustName, Street, City, State, PostCode)>
<!ELEMENT CustName (#PCDATA)>
<!ELEMENT Street  (#PCDATA)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT State (#PCDATA)>
<!ELEMENT PostCode (#PCDATA)>
<!ATTLIST Customer  CustNumber  ID #REQUIRED>
<!ELEMENT OrderDate (#PCDATA)>
<!ELEMENT Item(Description, Price, Quantity>
<!ATTLIST Item ItemNumber  ID #REQUIRED>
```

personalized publishing and portals, customized presentations, and content and document management applications.

An example of data-centric document is the one reported in Figure 1. The structure of the a data contained in the document is quite regular and the content of the various tags is homogeneous. On the other hand, Figure 3 presents an example of a text-centric document, taken from Bourret, describing a specific product.

Data-centric and text-centric documents require different query capabilities. Indeed, due to the presence of a more rigid structure, data-centric documents can be queried by using languages very similar to database query languages. On the other hand, text- centric documents, due to their less structured features, require languages built on concepts developed in the area of Information Retrieval (IR).

Figure 3: An example of text-centric XML document

```
<Product>
        <Name>Turkey Wrench</Name>
         <Developer>Full Fabrication Labs, Inc.</Developer>
         <Summary>Like a monkey wrench, but not as big.</Summary>
        <Description>
            <Para>
                The turkey wrench, which comes in both right- and
                left-handed versions (skyhook optional), is made of the finest
                stainless steel. The Readi-grip rubberized handle quickly adapts
                to your hands, even in the greasiest situations. Adjustment is
                possible through a variety of custom dials.
            </Para>
         <Para>You can:</Para>
         <List>
              <Item><Link URL="Order.html">Order your own turkey wrench</
                  Link></Item>
              <Item><Link URL="Wrenches.htm">Read more about wrenches</
                  Link></Item>
              <Item><Link URL="catalog.zip">Download the catalog</Link></
                  Item>
         </List>
         <Para>
              The turkey wrench costs just $19.99 and, if you
              order now, comes with a hand-crafted shrimp
              hammer as a bonus gift.
          </Para>
        </Description>
</Product>
```

Table 1: Comparative analysis of five XML query languages

	XML-QL	XQL	Quilt	XQuery	Lorel
Basic query structure	Where-In-Construct	Path and filtering expr.	FLWR/XPath expression	FLWR/XPath expression	Select-From-Where
Restructuring	Partial	No	Partial	Partial	Yes
Join operators	Yes	Partial	Partial	Partial	Yes
Tag variables & path expr.	Yes	Path expr. only	Path expr. only	Path expr. only	Yes

In the following, we present and classify several approaches for querying both data-centric and text-centric documents.

XML QUERY LANGUAGES

Due to the widespread use of XML for Web data and database content representation, several query languages for XML data have been proposed so far which differ both in the expressive power and in the way they express XML queries. Some of the proposed languages come from the database community and thus are greatly inspired by database query languages (i.e., SQL and OQL). Among them we recall Lorel (Goldman et al., 1997), originally defined for querying semistructured data; XML-QL (Deutsch et al., 1999); XML-GL (Bonifati and Ceri, 2000); YATL (Cluet et al., 1998); and Quilt (Robbie et al., 2000). Other languages are more closely inspired by XML and come from the document processing community. The most important of these languages is XQL (Robbie et al., 1998), which can be regarded as an extension of the XSL pattern syntax (WWW-XSLT). Since these languages are greatly influenced by database query languages, they are particularly suited for querying data-centric documents.

No standard for an XML query language has been so far released, but the W3C XML Query Working Group is working at a standard XML query language, called XQuery (WWW-XQuery). XQuery is mainly derived from Quilt although it borrows some ideas also from other XML query languages, such as XQL, Lorel, YATL and XML-QL, and it is greatly influenced by SQL and OQL.

Comparative analysis of XML query languages can be done along several dimensions (Bonifati and Ceri, 2000; Fernandez et al.; Maier, 1998). In the following, in presenting the various XML query languages, we consider the following dimensions:

1. **Basic structure of the query**. This dimension evaluates how queries are expressed in the language (for example, whether a SELECT-FROM-WHERE paradigm is used or the queries have an alternative structure).

Figure 4: An example of: (a) XML-QL; (b) a Lorel query on the document in Figure 1

```
WHERE <SaleOrder>                          SELECT xml(SaleOrder: {
 <Item>                                    (SELECT xml(Item:{ItemNumber n,Description d})
  <ItemNumber>$n</ItemNumber>                   FROM SaleOrder.Item  i, i.ItemNumber n,
   <Description>$d</Description >                      i. Description d
  <Quantity>$q</Quantity >                      WHERE i.Quantity >6) })
  </Item>
</SaleOrder> IN "www.dcfm/orders.html"
  $q>6
CONSTRUCT <Item>
                <ItemNumber>$n</ItemNumber>
                 <Description>$d</Description >
                <\Item>
```

(a) **(b)**

2. **Restructuring of query results.** This dimension is related to the support for constructors that allow a complete restructuring of the query with respect to the structure of the original documents on which the query is applied.
3. **Support for join-like operators.** This dimension evaluates whether the query language supports operators to combine data from different documents.
4. **Use of tag variables or path expressions.** This dimension is related to the support for queries on arbitrarily nested data and on data whose structure is not completely known.

In the following, we focus our attention on XML-QL, XQL, XQuery, Quilt and Lorel, and we compare them according to the dimensions previously introduced. In describing the languages we make use of the XML document in Figure 1. A summary of the comparative analysis is given in Table 1.

Basic Structure of the Query

Queries in XML-QL follows a **WHERE** *pattern* **IN** *source* **CONSTRUCT** *result* syntax, where *pattern* defines an XML document template which is matched against the input XML document identified by *source* (e.g., a URI), whereas *result* defines the structure of the result. An example of XML-QL query retrieving the number and the description of all the items whose quantity is greater than 6 is given in Figure 2(a).

Lorel queries follow the **SELECT-FROM-WHERE** syntax, where all the clauses may in turn contain queries. For example, Figure 2(b) reports the Lorel query corresponding to the XML-QL query in Figure 2(a).

Figure 5: An example of Quilt query on the document in Figure 1

```
<Result>
  (FOR $i IN document ("www.dcfm/orders.html")//Item,
     WHERE $i/Quantity > 6
      RETURN $i/ItemNumber, $i/Description
   )
</Result>
```

Quilt and XQuery queries are very similar and both the languages allow a great flexibility in query specification. Like OQL, these languages are functional languages in which each query is represented as an expression. XQuery and Quilt support a variety of expressions, which can be nested and combined together thus allowing a great degree of flexibility. For instance, one of the form of these expressions is an XPath expression (WWW-XPath). Alternatively, a query may consist of **FOR/LET**, **WHERE**, and **RETURN** clauses. The FOR/LET clause contains a set of XPath expressions identifying the set of nodes on which the query applies. The WHERE clause applies a filter to the nodes selected by the XPath expressions, whereas the RETURN clause defines the structure of the query result. This kind of expressions are referred to as *FLWR* (FOR/LET, WHERE, and RETURN) expressions. For instance, Figure 5 reports the Quilt version of the query presented in Figure 4.

An XQL query consists of a *path expression* and an optional *filter expression*. The path expression is used to select specific nodes within an XML document, by giving their path in the graph representation of the document, whereas the filter expression selects only nodes meeting specified criteria. Filter criteria usually refer to the nodes which are accessed during the traversal of the path specified by the path expression, although XQL provides also operators to refer to nodes which are not necessarily along the specified path. Examples of XQL queries over the document in Figure 1 are:

- *SaleOrder/Item[Quantity>6] {ItemNumber | Description}*: returns the number and the description of all the items with a quantity greater than 6.
- *SaleOrder[OrderDate = "11052001"]*: returns all the orders made on November 5th 2001.

Restructuring of Query Results

Sometimes it is useful that the result of a query has a different structure with respect to the original XML documents on which the query is performed. For this reason, most XML query languages provide restructuring operators to be applied to a query result.

In Lorel, new XML elements can be built by using the *xml()* function, whereas in XML-QL restructuring is specified in the CONSTRUCT clause. Quilt and XQuery provide the RETURN clause for restructuring query results. By contrast, XQL does not support a restructuring operator. Additionally, Lorel provides a GROUP-BY construct for aggregating elements. By contrast, an explicit group-by clause is missing from the current description of XML-QL, XQL, Quilt and XQuery.

Join Operators

Join queries are a very relevant class of queries, since they allow one to combine information from multiple XML documents. XML-QL, Quilt, XQuery and Lorel fully support join operations within the same document as well as among different documents. By contrast, XQL only supports join of data belonging to the same document, whereas it does not support join operations across different documents.

Tag Variables and Path Expressions

Tag variables and path expressions are a relevant feature of any XML query language in that they allow the specification of queries on documents whose exact structure is not known. Tag variables allow the selection of document portions in which some element tags match regular expressions (built by using wildcards such as '*' or '%').

XML-QL, Quilt, XQuery and Lorel support tag variables. For instance, consider an XML source containing bibliographic entries. The XML-QL query in Figure 6 selects books in which "Ford" is either an author or an editor. By contrast, XQL, Quilt and XQuery do not currently support tag variables.

As far as path expressions are concerned, Lorel and XML-QL support regular path expressions. Regular path expressions are those built by using the alternation ('|'), concatenation ('.') and Kleen-star ('*') operators. XQL,

Figure 6: Use of tag variables in XML-QL

```
WHERE <$p>
        <Title> $t </Title>
       <$x> Ford </ >
      </> IN www.dscfm/bib.xml
      $x IN {author,editor}
CONSTRUCT <$p>
            <Title> $t </Title>
            <$x> Ford </ >
           </>
```

Quilt and XQuery do not support regular path expressions, but they support the definition of relative and absolute locations, using the '/' and '//' operators.

INFORMATION RETRIEVAL APPROACHES TO XML DOCUMENT RETRIEVAL

Besides the efforts originating from a database context, the IR community has started to investigate how IR techniques can be applied to XML documents. Indeed, each XML document can be seen as a text document including tags. Thus, at least two approaches can be devised for applying IR techniques to quering XML documents. In the simplest approach, we can simply discard tag information, thus obtaining a simple text document, to which typical IR techniques can be applied. This approach is quite simple; however it has several disadvantages since, by removing tags, we lose several important information that could have been used to improve the query process, thus leading to lower retrieval performance. An alternative method consists in considering both textual and tag information. In this case, the search can benefit from the knowledge represented by the tag structure. When dealing with both tag and textual information, traditional IR techniques have to be extended in order to cope with both this information.

The scientific community has therefore started to investigate how XML query languages can be extended with IR features. Several approaches have been defined, such as XXR (Theobald and Weikum, 2000), XIRQL (Fuhr and Grobjohann, 2000), ELIXIR (Chinenyanga and Kushmerick, 2001), ApproXQL (Schlieder and Naumann, 2000) and XIRS (Vutukuru et al.), however no standard proposals exist yet. The existing approaches can be compared according to the following dimensions:

- **Similarity-based conditions**. This dimension concerns how XML query languages have been extended from a syntactic point of view to support the representation of similarity-based queries.
- **Ranking**. Since it is very difficult to compute exhaustive answers for XML queries, it is important to define some mechanism for ranking the obtained results. This dimension thus concerns the used ranking algorithms.
- **Personalization**. Providing precise structural information for querying XML documents is a very difficult task; for this reason the automatic or partially automatic completion of the queries submitted by the user with conditions derived, for example, from user profiles could be quite useful to supply to the user better results, simplifying the user query construction

Table 2: Classification of the IR approaches

XL	XIRQL	ApproXQL	XIRS	ELIXIR
ML-QL	XQL	XQL	XQL	XML-QL
tween an element an attribute and a nstant	Between an element or an attribute and a constant	No new operators, new approach to the evaluation	Applied to (keyword, concept) pairs	Between intermediate XML-QL results
ore assigned to ~ nditions, erpreted as obabilities and mbined under the dependence sumption	Probability-based ranking	Based on transformation costs	Based on the tag-distance between the concept and the keyword	Based on WHIRL
	-	The user can restrict the number of applied transformations	Personalization of (keyword,concept queries by using user profiles	-
	-	Text index + structural index	Text index + structural index	-

process. This dimension is therefore related to the personalization algorithms used in query specification and execution.

- **Indexing**. Typical indexes used for full-text search, such as inverted indexes, have to be revisited in order to index not only text but also tags. This dimension thus concerns the types of indexes used to support full-text search against XML documents.

In the following, we investigate the previous issues by considering some recently proposed IR approaches to XML retrieval. In particular, we mainly focus our attention on XXR, XIRQL, ELIXIR, ApproXQL and XIRS, and we compare them according to the dimensions pointed out above. The result of this comparison is presented in Table 2.

In the table, for each considered proposal, we also specify the XML query language on which it is based. Empty positions mean that the authors are not aware of how the corresponding issue is treated in the considered proposal.

Similarity-Based Conditions

In order to support IR features, XML query languages have been extended to support new constructs, to specify retrieval by similarity, so that a query returns a ranked list of answers, sorted by descending relevance.

XXL has been defined by extending XML-QL with similarity conditions on elements and their attributes. In particular, the similarity operator '~' is introduced which measures the similarity between a value of a node of the XML document and a constant or a variable element given by a query. The '~'

operator can also be used as a unary operator applied to an element as a shortcut for the condition: *element.NAME ~ constant*, where *element.NAME* represents the name of the considered element. For example, the condition *Para.# ~ "wrench"* allows one to retrieve XML documents containing the word "*wrench,*" or a similar word, inside an element named "*Para.*" This condition, when evaluated against the document presented in Figure 3, returns a score equal to 1, since the word "*wrench*" is exactly contained inside a "*Para*" element.

XIRQL is an XML query language with IR features obtained by extending XQL. In particular, whereas the result of an XQL query is a set of nodes, the result of a XIRQL query is a weighted set of nodes. In order to use term weight to define a relevance ranking approach, the operator *cw* (similar to operator '~' of XXL) is introduced which determines whether a given element or text associated with an element contains a given word. The operator returns a value between zero and one, representing a similarity score. Moreover, it is also possible to associate weights with conditions. For example, referring to the document presented in Figure 3: *//Product[.//* cw "steel" and .//* cw "catalog"]* is an example of an XQL query extended with operator *cw*. The query retrieves all the products having a nested element containing the word "*steel*" and a nested element containing the word "*catalog.*" Weights can be assigned to the two independent conditions, leading for example to the expression*: //Product[0.6 * .//* cw "steel" + 0.4 * .//* cw "catalog"]*. In this case, a higher rate is assigned to the condition concerning "*steel*" (0.6). Such weights are then used to rate the retrieved documents.

ApproXQL is a pattern-matching language corresponding to a subset of XQL. To this language, a semantics based on cost-based transformations is applied. In particular, a document is retrieved by a query if it is possible to transform the (tree representation of the) document into the (tree representation of the) query by applying some atomic transformations. Three main transformations are provided: *renaming*, changing the search context of a query part; *insertion*, introducing in the query an inner element, thus restricting the query to a more specific context; *deletion*, removing a query element, thus obtaining a more general query context. Note that this approach represents an example of application of a similarity condition to both element content and document structure.

ELIXIR is another language with IR characteristics for querying XML documents, based on XML-QL query. The main extension with respect to XML-QL is the introduction of a textual similarity operator, which applies to intermediate data generated through XML-QL queries. In order to execute these extended queries, the queries are rewritten into a series of XML-QL

queries, generating intermediate results. These results are then represented in WHIRL (Cohen, 2000), in order to evaluate similarity predicates on this intermediate data. We recall that WHIRL (*W*ord-based *H*eterogeneous *I*nformation *R*epresentation *L*anguage), is an information system combining logic-based and text-based representation methods. More precisely, it corresponds to a subset of non-recursive Datalog, extended by introducing an atomic type for textual entities, and an atomic operation for computing textual similarity.

The output obtained by the evaluation of the WHIRL query is then translated in XML format and returned to the user. Note that this approach is quite different from that supported by XXL and XIRQL. Indeed, in those cases, the similarity operator is just applied between a data value and a constant, but no similarity joins can be specified across two data values. On the other hand, this can be specified by ELIXIR.

Ranking

The basic idea of ranking is that if no exact matching documents are found, results similar to the one requested by the query should be ranked according to their similarity degree. The problem of similarity between keyword queries and text documents has been intensively investigated in the literature. However, traditional models cannot be directly applied to XML documents for two main reasons: they ignore the structure of XML documents and they are based on the frequency by which words appear in the overall documents (*term distribution*). This last property is not useful for data-centric documents term distribution, since in this case documents have a very rigid structure and we would like to use this structure in the queries. On the other hand, term distribution can be useful for text-centric documents, but this concept is too poor without considering the associated structure. Starting from the previous discussion, several approaches have been proposed, adapting traditional ranking approaches for text documents to XML documents.

In XXL, the score assigned to similarity conditions is assumed to be normalized between zero and one, and indicates the relevance of the node appearing in the condition for the given constant. In order to compute similarity score, a text search mechanism is used, based on a thesaurus and other means. The similarity score for similarity-based conditions is interpreted as a relevance probability. Conditions are then provided to compute the relevance of an overall query starting from the relevance of similarity conditions.

A different approach is used in XIRQL. Starting from the consideration that an XML document can be represented as a tree, differently from an unstructured document that is interpreted as a sequence of words, the first

problem is determining to which objects weights should be assigned. XIRQL considers as *index objects* only specific nodes (and the corresponding sub-trees) appearing in the tree representation of an XML document. Such nodes can be specified inside an extended DTD for the considered XML documents. From the weighting point of view, index objects should be disjoint, such that each term occurrence is considered only once. However, non-disjoint index objects are also allowed. In this case only nodes not belonging to the sub-tree rooted by the inner sub-tree are considered in the outer sub-tree.

Based on the notion of index object, a similarity score is assigned to atomic conditions, using the cw operator, and to Boolean queries, using a probabilistic approach, assuming that and and or operations are applied to stochastically independent events. This is possible by assuming that the index node determines the significance of a term in the context given by that node and by assuming that different occurrences of the same term inside an index object represent the same event. Different events are moreover assumed to be independent, and occurrences of the same term in different index nodes are also independent from each other. Following this idea, retrieval results correspond to Boolean combinations of probabilistic events.

A similar approach, tailored to text-centric documents, has been proposed in Schlieder and Meuss, 2000. Also in this case, the basic idea of the proposal is that of replacing the concept of term distribution with the concept of *structural term* distribution. Similarly to the XIRQL approach, a structural term is a sub-tree of the XML document. Based on the concept of structural term, the traditional Vector Space model (Baeza-Yates and Ribeiro-Neto, 1999) has been extended to deal with this new concept. The Vector Space model is based on the comparison of the query term vector with the document term vector. Each term has a certain weight which reflects its descriptiveness with respect to the query or document. This model can be easily extended to deal with structural terms: standard definitions of term frequencies and inverse document frequencies are simply redefined by considering structural term distribution. The main difference between a Vector Space model based on structural terms and one based on terms is that in the first case terms are dependent (since some structural terms can contain some other structural terms). In the context of XML documents, this is a good property since it allows one to improve precision without lowering the recall. For example, every document containing the query structural term *Price[25]* also contains the structural term *25*. However, documents containing *Price[25]* are preferred since their score is a function of both terms. Thus, it is correct to assign to these documents a higher weight.

Even if this approach considers sub-trees as index terms, it differs from

XIRQL from two points of views. First of all, in this approach sub-trees are not necessarily disjoint; moreover, ranking is performed using the Vector Space model and not a probabilistic approach as in XIRQL.

Finally, in ApproXQL, cost-based transformations are used to assign a similarity score to the results. In particular, the score assigned to a document is obtained by adding the cost of each single transformation step. It is important to point out that this approach is suitable for data-centric documents, where the structure is typically static. On the other hand, it is not suitable for text-centric documents, due to their variable structure.

Personalization

With personalization we mean the ability of a system to complete query specification by using information concerning user profiles. This leverages the user from the very difficult step of constructing the correct query and burdens the system with the process of completing the partial query submitted by the user. We believe that personalization is a very important issue in the context of querying XML documents. Unfortunately, as far as we know, only a few works have been done concerning query personalization.

A weak form of personalization has been proposed in XIRS (Vutukuru et al.) to support simple (keyword, concept) conditions. In this system, a user profile is maintained by monitoring the browsing behaviour of the user. Accessed XML documents are grouped in sessions, each corresponding to documents having similar top frequency words. A user profile is generated for each session, consisting of the top frequency words in the session. A keyword-concept matrix is also maintained. Each entry *(i,j)* of the matrix specifies the relative importance of concept *j* for a keyword *i*. The initial entries of the matrix are the number of times a keyword occurs in a particular concept (i.e., an element) over the entire set of XML documents to be searched. Each entry is updated by considering user profiles as follows. Each session is assigned a weight, representing the importance of the session. Higher weights are assigned to more recent sessions. An entry *(i,j)* is updated if both *i* and *j* appear in the top frequency words of a session. In this case, the weight of the session is added to the entry *(i,j)*. At query time, the user just supplies a keyword. The keyword-concept matrix is then used to determine the top k concepts for that keyword, thus generating k queries of the form (keyword, concept) that are then executed.

User preferences are also considered in ApproXQL to limit the number of transformations, applied by the system in executing similarity-based conditions. The user can indeed specify some restrictions on rename, insertion and deletion transformations and the system will not consider these transformations when

computing similarity scores. Note that this process is completely manual and therefore it does not correspond to the concept of personalization we have introduced before.

Another example of personalization is provided by the Niagara system (Naughton et al., 2001). The basic assumption underlying the design of Niagara is that the user cannot know in advance where XML documents are located. Thus, the query is automatically personalized in order to determine the documents against which the query has to be executed. More precisely, a query execution in Niagara is composed of two main steps. First, the user specifies a query in a modified version of the XML-QL language. From this query, a query expressed in the *S*earch *E*ngine *Q*uery *L*anguage (SEQL) is extracted, which supports Boolean combinations of predicates that specify containment relationships between XML elements and their contents. Such SEQL query is then passed to the Niagara search engine, that evaluates the query and returns to the XML-QL engine a set of URLs corresponding to the XML files that should be used in evaluating the XML-QL query.

Indexing

In order to execute IR queries, specific indexes must be constructed over XML documents. If all the tag information is removed, traditional IR indexes can be used, but of course leading to lower retrieval performance. Another approach consists in extracting some important structural and contextual information from XML documents, for example specifying which tags should be considered in the search. A more complete approach consists of indexing the tags as if they were index terms. In this case, the problem is how such an index can be coupled with a structure indexing textual information contained inside an XML document, thus supporting both queries on the document content and structure. For example, XIRS uses two inverted indexes, one for indexing tags and one for indexing content. The first index associates to each tag the list of all the documents containing it; the second index associates with each keyword the list of all the documents containing such a keyword. Each document points to its tree-based representation, at the lowest level tag within which the keyword occurs.

Most of the full-text indexing approaches for XML documents are based on the usage of the inverted index data structure (Baeza-Yates & Ribeiro-Neto, 1999). The proposed techniques mainly differ for two parameters: the method used to specify positions inside documents and the indexed information. Concerning indexed information, inverted indexes for XML documents may consider either element/attribute names or element/attribute content or both. Often, both types of information are indexed (Aguilera et al., 2000;

Florescu et al., 2000; Kanemoto et al., 1998; Vutukuru et al.; Yamamoto et al., 1999). Concerning the method used to specify positions, as pointed out in Sacks-Davis et al. (1997), index approaches can be classified in *position-based* indexes and *path-based* indexes. In path-based indexes, occurrence positions are represented by element paths, from the root of the document to the considered information. For example, the path-based position of word *wrench* inside the document presented in Figure 3 is Product[1]/Para[2]/List[1]/Item[1], meaning that the word is contained in the first item of the list associated with the second paragraph describing the first product. Thus, in the previous path expression, numbers represent the ordering number of the considered tag occurrence in the sibling lists, from left to right.

A disadvantage of path-based indexing is that it can support only a limited number of full-text queries. For example proximity queries, i.e., queries asking for some group of words at a certain distance from each other, cannot be represented by using this approach. On the other hand, these queries can be executed by using position-based indexes that guarantee a higher flexibility. Position-based index refers to the XML content (words, element names, attribute names and attribute values) by means of positions. Different types of positions have been proposed in the literature, such as the number of bytes between the root of the document and the considered information (Aguilera et al., 2000; Kanemoto et al., 1998; Yamamoto et al., 1999) or the pair of numbers assigned by a preorder/postorder visit of the tree (Aguilera et al., 2000).

Finally, in Yamamoto et al. (1999), a new type of inverted index has been proposed to support queries concerning namespaces. Queries referring to elements or attributes belonging to a specific namespace, such as "*Find all XML documents which refer to the Dublin Core schema and the content of its* creator *element is Alessandro Manzoni.*" differ from traditional XML queries from the fact that a specific path to the element or attribute is not provided since such a path can be different from one document to another. Such conditions cannot therefore be specified by using typical structural conditions and traditional indexes cannot be used to support them.,

In Yamamoto et al. (1999), a specific index for supporting namespace conditions is presented. The idea of the index is quite easy. Starting from the consideration that a namespace-based query may just specify an element name defined in a given namespace, instead of indexing words, elements or paths, the index, called *reverse path index,* indexes reverse paths, which are paths from nodes to the root in the XML tree structure. With which node, the URI of the corresponding namespace is associated. Thus, a reverse index is an inverted file of reverse paths, i.e., a set of tuples of the form < reverse path, list of

documents>. In order to be able to answer full-text queries with respect to namespaces, the reverse index proposed in (Yamamoto et al., 1999) has been extended. The new index is a set of tuples of the form <word, reverse path, list of documents>, where each pair (word, reverse path) is associated with all the document identifiers, in which the word is found at the end of the considered path.

Even if position-based indexes support a large number of full-text query types, they do not guarantee good performance with respect to update operations. Indeed, since positions represent an absolute address, updates usually cause the re-computation of the data coordinates. In high dynamic environments, such performance decreasing cannot be accepted. Several approaches have therefore been proposed in order to reduce the update costs. For example, in Kha et al. (2001), the concept of *relative region coordinate (RRC)* is provided, whose idea is that of expressing the coordinate of an XML element in relation to its parent element coordinates. Thus, using RRC, it is only possible to know where the element starts and end inside the region of its parent elements. As a consequence, only a portion of the index file has to be modified in case of update.

COMMERCIAL PRODUCTS

Due to the widespread use of XML, the market has pushed the development of commercial XML data servers and IR systems. In the following, we present a brief overview of the most significant developments in each category.

XML Data Servers

We can divide commercial DBMSs providing support to XML into two broad categories (Bourret): *Native XML Databases* and *XML-Enabled Databases*. In the first category fall all the products that store and manage XML documents in their native form, whereas XML-Enabled Database are databases (usually relational) that provide interfaces for mapping data between XML and their internal data model, and vice-versa.

An XML-Enabled DBMS must thus support two classes of services: *data extraction* services and *data formatting* services. Data extraction services make the DBMS able to receive XML documents from the Web and to extract from them structured data to be stored in the DBMS, according to the DBMS internal data model. Data extraction requires a specific DTD (or XML Schema) associated with the XML documents. On the other hand, data formatting services make the DBMS able to take the result of a query, expressed in the DBMS query language, and encode the resulting data in an XML documents

that can then be transferred over the network.

Two main differences exist between native XML databases and XML-enabled databases. First, native XML databases can uniformly store both data-centric and document-centric XML documents, whereas in XML-enabled databases, the mapping of an XML document onto the database schema is often possible only if the document has an associated DTD or XML Schema. Additionally, the only interface to the data stored into native XML databases is XML itself and its related technologies (such as XPath, DOM, XQL, etc.), whereas in an XML-enabled database, different technologies can be used.

In the following, we illustrate in more detail the features of some XML-enabled and XML-native database systems.

XML-Enabled Databases

From the side of XML-Enabled Databases, almost all the most widely used commercial relational DBMSs have been extended to manage XML documents. In the following, we briefly illustrate the XML support provided by Oracle 8i and 9i (Oracle), IBM DB2 (IBM) and Microsoft SQL Server (Microsoft).

Oracle 8i, 9i. Oracle 8i and 9i have extended their architecture, based on an object-relational model, with a specific XML layer, supporting a set of tools for managing XML documents. Data extraction and data formatting for data-centric documents are supported by XML-SQL Utilities, available as command line front end, Java API and PL/SQL API. It is important to recall that data extraction requires a specific DTD for XML documents and, on the other hand, data formatting services generate XML documents with a fixed structure. The user can, however, relaxe this approach by using XSL to transform generic XML documents before extraction and after construction. Document-centric XML documents are stored in Oracle 8i by using an unstructured representation in CLOB or BFILE fields. In the second case, the XML document is stored and managed outside the database, but metadata for such documents are stored in the DBMS for fast indexing and retrieval. Such documents can then be queried by using a tool called interMedia Context in Oracle 8i and Oracle Text in Oracle 9i, which provides flexible document indexing, with respect to both element/attribute content and names, and enables SQL statements to query the content of XML documents. XML documents can be queried by using XPath, in version 9i.

IBM DB2. In IBM DB2, XML is supported by providing an XML Extender that extends DB2 functionalities to the management of XML documents. Data-centric XML documents can be stored in an *XML collection*, representing a set of relational tables that contain data which have been

extracted from XML documents. The access and storage methods supported by the extender allow one to compose an XML document from existing data, to decompose an XML document and to use XML as an interchange method. Differently from Oracle 8i, the mapping between XML documents and XML collections is not fixed but can be specified through a *Data Access Definition* (DAD) document, that defines the mapping between DTD elements and relational tables and columns by using an XSLT and XPath syntax. Document-centric XML documents are managed through *XML columns*. DB2 supports three different data types for XML columns: XMLCLOB, XMLVARCHAR and XMLFile, storing an XML document as a CLOB, a VARCHAR and a file on the local file system, respectively. DAD are used to specify which XML elements or attributes should be indexed. User-Defined Functions (UDFs) are provided to support insert, delete and update operations against XML columns. Queries can be performed by using the IBM Text Extender or by using specific UDFs.

Microsoft SQL Server 2000. Microsoft SQL Server 2000 provides several tools for managing XML documents. In particular, data can be extracted from XML documents and stored in relational tables by using the OPENXML function. On other hand, XML formatting of a query result is provided by extending the SELECT-FROM-WHERE statement with the FOR XML clause. To this purpose, three different format types can be specified. SQL Server also supports an interesting XML view-based approach, by which it is possible to construct the so-called XML-Data Reduced (XDR) schemas. Such schemas, constructed by using an XML-like syntax, generate views of the database in XML format and can be queried with XPath, either through HTTP or by inserting XPath queries in specific XML templates (a similar approach can be used for SQL queries).

Native XML Databases

Several are the native XML databases today commercially available (Bourret). Among them we recall Excelon, developed by Excelon Corporation (Excelon), and Tamino[1] (Software AG) by Software AG. Other Native XML Databases are dbXML, an open source native XML Database, and Virtuoso, an XML data server which includes support for widely used Internet, Web and Data Access standards such as XPath, XSL-T, SOAP, WebDAV, SMTP, ODBC, JDBC and OLE-DB (Bourret). In the following we briefly present the main characteristics of Excelon and Tamino.

Excelon. Excelon is an XML data server for storing (using DOM), querying (through XQL) and indexing XML documents. Server-side applications can be easily implemented through Java server extensions stored in the Excelon database.

The Excelon XML platform consists of two main classes of components: *design* and *run-time* components. Design components provide tools for assisting application developers in the design, implementation, and deployment of applications running on the Excelon XML Platform. Excelon design components include:

- *Stylus Studio*, which provides a suite of tools for application development, including the *XML editor,* for XML document manipulation; the *DTD and XML Schema editor*, for DTD and XML Schema definitions; the *Java source editor,* for simple edits of Java source file; the *XML-to-XML mapper,* a visual tool for defining mappings between different XML schemas; and the *Debugger*, a visual XSLT and Java debugger.
- *Map Designer*, which provides support for mapping non-XML schemas into XML and vice-versa.

The run-time components provide tools for managing and delivering XML data. These components include:

- *Dynamic XML Engine* (DXE), a native XML repository for managing XML data. It provides a DOM interface to access XML data; an XPath processor to process queries on XML data; an XSLT processor, to transform XML data into HTML, WML or other delivery formats; and the support for triggers written in Java. Additionally, DXE provides three different kinds of indexes: value indexes, for string and numeric searches; text indexes, for word-based searches; and structural indexing, for structured-based searches.
- *XConnects Integration Engine,* a tool for transforming data from legacy sources into XML. Supported data sources are relational databases; the HL7 and DIALOG format; mainframe systems, such as CISC, Paradox and Clipper; ERP systems, such as SAP; and groupware systems, such as Lotus Notes.
- *DXE Manager*, a visual administration tool to manage all the components of the Excelon server.

Tamino. Tamino is an XML information server for storing, publishing and exchanging electronic documents, specifically conceived for e-business applications. Users can design their specific server-side applications by means of server extensions implemented in C, C++ or other COM/DCOM-enabled languages. The architecture of Tamino consists of the following components:

- *X-Port*, an HTTP-based Web server interface by which Tamino objects can be directly accessed trough their URLs.
- *X-Machine,* an XML engine which allows XML documents to be stored and retrieved in their original form. X-Machine includes user-defined server extensions and support for standard document transformations (i.e., XSL and CCS). Data can be queried through XQL.
- *SQL Engine*, a Web-enabled SQL engine to query relational data.
- *X-Node*, which provides support for the integration of data coming from multiple heterogeneous sources, such as data stored in a database, or file systems, or data provided by messaging systems.
- *Tamino Manager*, a graphical tool for performing administrative operations.

Information Retrieval Systems

Several XML search engines have been proposed and are currently present on the market (Luk et al., 2000). Among them, we recall: XML Query Engine (Katz), which is a JavaBean component for full-text indexing and searching of XML documents; XYZFind (Egnor and Lord, 2000), which builds a searchable database of all data from all XML documents, indexing values, numbers, structural names and content; Inktomi Search Software (formerly Ultraseek) (Inktomi), which supports searching XML files as full text, setting up multiple sets of fields for field-specific searching; SIM (The Structured Information Manager) (Structured), which uses XML and full-text indexing to provide a powerful combination of database searching and text retrieval; GoXML Search Engine (GoXML), which does XML-specific search by providing the concept of context, representing the markup tag for the searched text; XRS (Shin et al., 1998), which provides a BUS architecture to index and search XML documents by supporting an inverted list of all the terms appearing in the content of the elements and values of the attributes, referred by means of XPath expressions.

It is important to note that most products, in order to be sufficiently efficient and simple to use, have a limited expressive power. Another important aspect of XML search engines is that often they support *reactively structured queries*. With reactively structured queries we mean queries which are first only partially specified by the user and are annotated with structure in a further refinement process. Reactively structured queries are opposed to *proactively structured queries*, which are fully structured queries first submitted by the user. Proactively structured queries are typically used in database systems. However, it is very difficult to create complete structured queries since the user should have a deep knowledge of the base which is searched. For this reason,

reactively structured queries seem to be a fundamental aspect of XML Search Engines.

As an example of the functionalities supported by XML search engines, in the following we survey the main characteristics of XYZFind and XML Query Engine.

XYZFind. XYZFind is a native XML database, supporting IR features. The most interesting feature of XYZFind is that it classifies documents with respect to their schema and uses this schema to refine user queries, by applying a complete reactive query construction approach. More precisely, the user first submits an unstructured full-text query. The documents satisfying the query are then shown to the user by means of their corresponding schemas. The user has then to choose one of these schema in order to complete the search. At this step, a search form is presented to the user from which it can precisely specify his/her request. The approach applied by XYZFind can therefore be classified as a hybrid approach, since in a first step a category-based search, typical of IR systems, is performed and it is followed by a typical structured database search. It is important to recall that in XYZFind, the search is restricted to specific XML paths, and queries must be fully conjunctive (Egnor and Lord, 2000).

XML Query Engine. XML Query Engine is a search engine tool for XML. It is implemented as a JavaBean component that supports XML document search for element, attribute and full-text content. The index, once built, can be queried using an extension of XQL, providing a full-text capability. XML Query Engine is indeed an embeddable component that can be called inside its own applications. An interesting aspect is that query results can be delivered in three different formats. Two of these formats return XML documents. However, besides returning XML documents in standard format, a specialized format is also supported for representing "navigational metadata," describing the nodes the document contains in terms of their location within their originating documents. These metadata can then be used to re-navigate back into the original documents for further post-processing. The third result-set format is based on a Comma-Separated-Values (CSV) format, for particularly fast and compact result delivery of navigational metadata. Even if the engine has some interesting functionalities, it has not been designed to serve a high-volume, multi-user environment and has memory-dependent limitations that reduce its applicability.

RESEARCH CHALLENGES

Even if in the last years a large amount of work has been done for defining languages and architectures for retrieving XML documents, we believe that much work is still to be done. This is true mainly in the context of IR proposals for querying XML documents. Indeed, even if several search engines for XML documents have been developed, some important aspects have been only weekly investigated.

An important issue that has not received enough attention concerns the personalization of, possibly, similarity-based, XML queries. We claim that this topic is quite relevant since, by a guided personalization of XML queries, the user is relieved from the burden of writing complex expressions, at the same time making the system able to return more precise answers with respect to the user needs.

From our point of view, the personalization problem can be factorised in two main problems. The first concerns which user-based information should be used in the personalization, i.e., which kind of *user profile* could be useful to this purpose. We believe that a user profile should contain both information concerning the user needs and information concerning security aspects. Information concerning the user needs can be either automatically collected by the query engine or explicitly entered by the user when it subscribes to a Web IR service. For instance, the query engine can keep track during query processing of the previous queries made by the same user, and this information can be used to personalize the query result. This information can impact both the set of documents returned to the user and/or the relative ranking of the documents belonging to the query result. Additionally, the user can explicitly state her/his preferences. For instance, a user subscribing to an information dissemination service can state that he/she is interested in receiving all the documents related to Picasso, except those regarding the blue period of the painter. In this case, the system should notify the user when some relevant documents are acquired by the source and discard non-relevant information.

Another important issue related to user profiles is their use for providing some sort of access control on Web documents. Indeed, it is not only important to filter the documents according to the user preferences, but it is also important to filter documents according to the specified security policies. For instance, some Web content should be made available only to users who are older than 18 years, or some other Web content should be made available only to users accessing the Web from specific countries. It is thus important that the profile contains personal information (e.g., age, nationality, and so on) that can then be used to filter document release to users, according to the specified security policies.

The process of profile generation can be partially automatic, in the sense that the system may generate some profiles by analysing user accesses, but of course, to be more effective, the profile should be completed by the user or by the system administrator. Information needed for security purposes must be certified by some sort of Certification Authority (CA). This can be the site to which a user subscribes or a third-party CA.

As a second step, profiles should be used to rewrite XML queries, obtaining queries better reflecting the user needs. Additionally, query results must be filtered according to the security information stored in the profile to ensure that the user receives all and only those documents he/she is authorized to access according to the specified security policies.

A further important issue is to investigate how these flexible and expressive user profiles can fit with the work currently carried on by the Composite Capability/Preference Profiles (CC/PP) W3C Working Group, which is working on a method for using RDF to create a general, yet extensible framework for describing agent preferences and device capabilities (WWW-CC/PP). Although the focus of CC/PP is to define profiles for user agents, specifically Web browsers, and thus it is mainly devoted to the specification of hardware and software preferences, it is interesting to investigate whether and how CC/PP can be extended to deal also with user preferences and security-related information.

Additionally, the development of suitable mechanisms and models for specifying and processing user profiles during query evaluation cannot leave out of consideration the most relevant standardization efforts in the area of XML query languages. We recall that the XML Working Group of the W3C is working on a standard query language for XML documents, i.e., XQuery. It is thus important that all the proposals be designed to be compliant with this emerging standard. In particular, the extension of XQuery to support a full set of IR capabilities is certainly a fundamental issue, which is starting to be investigated by the scientific community.

Another important research issue concerns the relationships existing between Semantic Web and XML retrieval. As pointed out in Berners-Lee et al. (2001), the Semantic Web is the idea of having data on the Web defined and linked in a way that they can be used by machines not just for display purposes, but for automation, integration and reuse of data across various applications. It seems that the Semantic Web will be based on RDF (WWW-RDF) as the basic model for representing data and links among data. In such an environment, an important aspect is how data is queried, in a general and effective way. We believe that research on the Semantic Web will provide important contri-

butions to the retrieval of RDF objects and, since RDF is implemented in XML, also to the intelligent retrieval of XML documents.

CONCLUDING REMARKS

In this chapter, we have investigated problems, solutions and open issues related to the retrieval of XML documents. Since XML is currently being used as an interface language over the Web, by which (part of) document sources are represented and exported, XML retrieval is a fundamental problem in querying Web heterogeneous sources. Approaches to XML retrieval have been first classified in database and IR proposals. These approaches have then been analysed with respect to several parameters. Besides theoretical proposals, we have also surveyed some commercial products supporting XML retrieval. The chapter has been concluded by a discussion of important future research directions.

REFERENCES

Aguilera V. (2000). Querying XML documents in Xyleme. in *Proceedings of the ACM SIGIR 2000 Workshop on XML and Information Retrieval*. Athens, Greece.

Baeza-Yates, R. & Ribeiro-Neto, B. (Eds.). (1999). *Modern Information Retrieval*. New York: Wesley-ACM Press.

Berners-Lee, T., Hendler, J. & Lassila, O. (2001). The semantic Web. *Scientific America*, *5*(1).

Bonifati, A. & Ceri, S. (2000). Comparative analysis of five XML query languages. *SIGMOD Record*, *29*(1), 68-79.

Bourret R. XML Database Products. (2001). Available at: http://www.rpbourret.com/xml/XMLDatabaseProds.htm.

Chinenyanga, T. & Kushmerick, N. (2001). Expressive retrieval from XML documents. *Proceedings of the ACM 24th International Conference on Research and Development in Information Retrieval (SIGIR'01)*, New Orleans, USA.

Cluet, S., Delobel, C., Simeon, J. & Smaga, K. (1998). Your mediators need data conversion! *Proceedings of the ACM-SIGMOD International. Conference on Management of Data*, 177-188.

Cohen, W. W. (2000). WHIRL: A word-based information representation language. *Artificial Intelligence*, *118*(1/2); 163-196.

Deutsch, A., Fernandez, M., Florescu, D., Levy, A. and Suciu, D. (1999). A query language for XML. *Proceedings of the International Word Wide Web Conference*. Available at: http://www.research.att.com/~mff/files/final.html.

Egnor, D. and Lord, R. (2000). XYZFind: Searching in context with XML. *Proceedings of the ACM SIGIR 2000 Workshop on XML and Information Retrieval*, Athens, Greece.

Excelon Corporation. (2001). Extensible Information Server. *White Paper*. Available at: http://www.exceloncorp.com/platform/datasheets/extinfoserver_new.pdf.

Fernandez, M., Simeon, J. and Wadler, P. (Eds.). (1999). *XML Query Languages: Experiences and Exemplars*. Available at: http://www-db.research.bell-labs.com/user/simeon/xquery.html.

Florescu, D., Kossmann, D. and Manolescum, I. (2000). Integrating keyword search into XML query processing. *Proceedings of the 9th International World Wide Web Conference*, May, Amsterdam, The Netherlands.

Fuhr, N. and Grobjohann, K. (2000). XIRQL: An extension of XQL for information retrieval. *Proceedings of the ACM 2000 SIGIR Workshop on XML and Information Retrieval*, Athens, Greece.

Generalized Markup Language (SGML). (1986). In ISO 8879.

Goldman, R., McHugh, J. and Widom, J. (1997). From semistructured data to XML: Migrating the lorel data model and query language. *Proceedings of Very Large Databases Conference (VLDB)*, 436-445.

GoXML. (2001). Available at: http://www.goxml.com.

Kha, D. D., Yoshikawa, M. and Uemura, S. (2001). An XML indexing structure with relative region coordinate. *Proceedings of the 17th IEEE International Conference on Data Engineering (ICDE2001)*, 313-320, April 2-6, Heidelberg, Germany.

Kanemoto, H., Kato, H., Kinutani, H. and Yoshikawa, M. (1998). An efficiently updatable index scheme for structured documents. *Proceedings of 9th International Workshop on Database and Expert Systems Applications (DEXA '98)*, 991-996, IEEE Computer Society.

Katz, H. (2002). *XML* Query Engine. Available at http://www.fatdog.com.

IBM Corporation. (2001). *IBM DB2 Universal Database–Administration Guide–Version 7, 2001*. Available at: http://www.software.ibm.com/software/data/db2.

Inktomi Search Software. (2001). Available at: http://www.inktomi.com/products/search/.

Luk, R., Chan, A., Dillon, T. and Leong, H. V. (2000). A survey of search engines for XML documents. *Proceedings of the ACM SIGIR 2000 Workshop on XML and Information Retrieval*, Athens, Greece.

Maier, D. (1998). Database desiderata for an XML query language. *Proceedings of the Query Languages Workshop*, Cambridge, Massachusetts. Available at: http://www.w3.org/TandS/QL/QL98/pb/maier.html.

Microsoft SQL Server. (2001). Available at: http://www.microsoft.com/sql/default.asp.

Naughton, J. (2001). The Niagara Internet query system. *IEEE Data Engineering Bulletin*, *24*(2), 27-33.

Oracle Corporation. (YEAR). Available at: http://technet.oracle.com/products/.

Robbie, J., Lapp, J. & Scach, D. (1998). XML Query Language (XQL). *Proceedings of the Query Languages Workshop*, Cambridge, Massachusetts. Available at: http://www.w3.org/TandS/QL/QL98/pb/xql.html.

Robbie, J., Chamberlin, D. & Florescu, D. (2000). Quilt: An XML query language. Selected papers from *Third International Workshop on the World Wide Web and Databases,* LNCS, Dallas, Texas, USA.

Sacks-Davis, R., Dao, T., Thom, J. A. & Zoble, J. (1997). Indexing documents for queries on structure, content, and attributes. *Proceedings of the International Symposium on Digital Media Information Base (DMIB'97)*, 236-245.

Shin, D. W., Jang, H. C. & Jin, H. L. (1998). Bus: An effective indexing and retrieval scheme in structured documents. *Proceedings of the ACM Conference on Digital Libraries*, 235-243.

Schlieder, T. & Naumann, F. (2000). Approximate tree embedding for querying XML data. *Proceedings of the ACM SIGIR 2000 Workshop on XML and Information Retrieval*, Athens, Greece.

Shlieder, T. and Meuss, M. (2000). Result ranking for structured queries against XML documents. *Proceedings of the DELOS Workshop on Information Seeking, Searching and Querying in Digital Libraries*, Zurich, Switzerland.

Software AG. (2000). Tamino: The Information Sever for Electronic Business. *White Paper*. Available at: http://www.sofwareag.com/tamino/download/tamino.pdf.

Structured Information Manager. (2001). http://www.simdb.com.

Theobald, A. & Weikum, G. (2000). Adding relevance to XML. *Proceedings of the Third International Workshop on the Web and Databases, in conjunction with ACM SIGMOD'2000*, May.

Vutukuru, V., Khare, A. and Pasupuleti, K. (2000). *XIRS: XML Information Retrieval System*. http://www.cs.utexas.edu/users/vamsikv/xirs/xirs.html.

World Wide Web Consortium. (1998). Extensible Markup Language (XML) 1.0. Available at: http://www.w3.org/TR/1998/REC-xml-19980210/.

World Wide Web Consortium. (1999). *Resource Description Framework (RDF) Model and Syntax Specification*. Available at: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

World Wide Web Consortium. (2001). *XML Schema Part 0: Primer*. Available at: http://www.w3.org/TR/xmlschema-0/.

World Wide Web Consortium. (2000). *XHTML Basic*. Available at: http://www.w3.org/TR/xhtml-basic/.

World Wide Web Consortium. (2001). XQuery 1.0: An XML Query Language, *W3C Working Draft*, 2001. Available at: http://www.w3.org/TR/xquery.

World Wide Web Consortium. (2001). *XSLT: Extensible Stylesheet Language (XSL), 1.0.*

World Wide Web Consortium. (2001) *W3C Recommendation*, October. Available at: http://www.w3.org/TR/2001/REC-xsl-20011015/.

World Wide Web Consortium. (1999). XML Path Language (XPath), *W3C Recommendation*, November. Available at: http://www.w3.org/TR/xpath.

World Wide Web Consortium. (1999). *Composite Capability/Preference Profiles (CC/PP): A User Side Framework for Content Negotiation*. Available at: http://www.w3.org/TR/NOTE-CCPP/.

Yamamoto, Y., Yoshikawa, M. and Umeura, S. (1999). On indices for XML documents with namespaces. *Proceedings of Markup Technologies Conference 1999*, 127-135, December 7-9, GCA, Philadelphia, USA.