XML Information Retrieval and Information Extraction

Norbert Fuhr*

University of Dortmund, Germany

Abstract. We present a new query language for information retrieval in XML documents and discuss its combination with information extraction methods. XIRQL is an XML query language which implements IR-related features such as weighting and ranking, relevance-oriented search, datatypes with vague predicates, and structural relativism. For information extracted from texts, XIRQL can rank records based on uncertainty weights, and single conditions may be evaluated using vague predicates for fact retrieval. When IE is used for automatic XML markup of plain texts, XIRQL is able to consider uncertainty weights resulting from this process, and the markup leads to increased precision of text searches.

1 Introduction

In many applications, large volumes of full-text documents are available. However, often the systems managing these large volumes of data offer only poor retrieval capabilities. Thus, the knowledge contained in the documents can hardly be exploited.

Currently, there is a trend towards XML as standard document format. For content based searches in full-text documents, XML offers two major advantages:

1. Since XML represents the logical structure of a document in explicit form, the retrieval system can return appropriate logical units as answers to content-based queries.
2. Based on the markup of specific elements, high-precision searches can be performed that look for content occurring in specific elements (e.g. distinguishing between the sender and the addressee of a letter, finding the definition of a concept in a mathematics textbook).

Unfortunately, most XML query languages proposed so far do not provide mechanisms for performing content-based searches in XML documents. In the following section, we present the new query language XIRQL (XML IR query language) which implements several IR concepts for XML retrieval.

Even under the assumption that XML will become the standard document format, there is still the legacy problem of large volumes of plain text. In order to apply XML retrieval mechanisms to these texts, automatic markup

* Email: norbert.fuhr@acm.org

methods can be applied. However, due to the uncertainty of this process, subsequent retrieval of the documents prepared this way also should take into account this uncertainty. Since vagueness and uncertainty are central concepts of XIRQL, this can be easily accomplished.

As an alternative method to automatic markup, information extraction aims at the extraction of facts and knowledge from texts, in order to represent this information in a standard record format. Again, no perfect solutions for this process are possible, so the resulting uncertainty should be considered when accessing these records.

In Section 4, we discuss the relationship between automatic markup and information extraction and we show how XIRQL can be applied to the results of these processes. Finally, we give an outlook on further work in this area.

2 XML Retrieval

In XML documents, text is enclosed in *start tags* and *end tags* for markup, and the *tag name* provides information on the kind of *content* enclosed. As an exception to this rule, #PCDATA elements (plain text) have no tags. Elements can be nested, like e.g. `<author> <first> John </first> <last> Smith </last> </author>`.

Elements can also be assigned attributes, which are given in the start tag, e.g. `<date format="ISO">2000-05-01</date>`; here the *attribute name* is `format`, and the *attribute value* is `ISO`.

An example XML document follows , which also illustrates the tree structure resulting from the nesting of elements. Figure 1 shows the corresponding document tree (the dashed boxes are explained in section 3.2).

```
<book class="H.3.3">
  <author>John Smith</author>  <title>XML Retrieval</title>
  <chapter>  <heading>Introduction</heading>
    This text explains all about XML and IR.
  </chapter>
  <chapter>  <heading>XML Query Language XQL</heading>
    <section>  <heading>Examples</heading>
    </section>
    <section>  <heading>Syntax</heading>
        Now we describe the XQL syntax.
    </section>
  </chapter>
</book>
```

All XML documents have to be *well-formed*, that is, the nesting of elements must be correct (`<a><b></a></b>` is forbidden). In addition, a *document type definition (DTD)* may be given, which specifies the syntax of a set of XML documents. An XML document is *valid* if it conforms to the corresponding DTD.

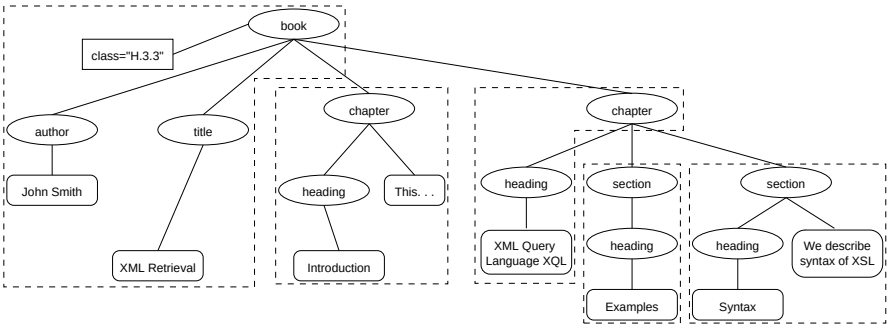

Fig. 1. Example XML document tree

XIRQL is based on the XPath standard [Clark & DeRose 99], which was derived from the XML query language XQL [Robie et al. 98]; XPath also forms the path expression part of query language XQuery [Fernández et al. 01] proposed by the W3C. Here we give a brief description of those elements of XPath which are used in XIRQL.

XPath retrieves elements (i.e. subtrees) of the XML document fulfilling the specified condition. The query `heading` retrieves the four different heading elements from our example document. Attributes are specified with a preceding '`@`' (e.g. `@class`). Context can be considered by means of the child operator '`/`' between two element names, so e.g. `section/heading` retrieves only headings occurring as children of sections, whereas '`//`' denotes descendants (e.g. `book//heading`). Wildcards can be used for element names, as in `chapter/*/heading`. A '`/`' at the beginning of a query refers to the root node of documents (e.g. `/book/title`). The filter operator filters the set of nodes to its left. For example, `//chapter[heading]` retrieves all chapters which have a heading. (In contrast, `//chapter/heading` retrieves only the heading elements of these chapters.) Explicit reference to the context node is possible by means of the dot (`.`): `//chapter[.//heading]` searches for a chapter containing a heading element as descendant. Brackets are also used for subscripts indicating the position of children within an element, with separate counters for each element type; for example `//chapter/section[2]` refers to the second section in a chapter (which is the third child of the second chapter in our example document).

In order to pose restrictions on the content of elements and the value of attributes, comparisons can be formulated. For example, `/book[author = "John Smith"]` refers to the value of the element author, whereas `/book[@class = "H.3.3"]` compares an attribute value with the specified string. Besides strings, XPath also supports numbers and dates as data types, along with additional comparison operators like `gt` and `lt` (for $>$ and $<$).

Subqueries can be combined by means of Boolean operators `and` and `or` or be negated by means of `not`.

For considering the sequence of elements, the operators `before` and `after` can be used, as in `//chapter[ section/heading = "Examples" before section/heading = "Syntax"]`.

These features of XPath allow for flexible formulation of conditions wrt. to structure and content of XML documents. The result is always a set of elements from the original document(s).

3 XIRQL Concepts

3.1 Requirements

From an IR point of view, XPath lacks the following features in order to use it for content-based retrieval of XML documents:

Weighting: IR research has shown that document term weighting as well as query term weighting are necessary tools for effective retrieval in textual documents. These two types of weights should be considered during retrieval, thus resulting in a ranked list of elements.

Relevance-oriented search: The query language should also support traditional IR queries, whereby only the requested content is specified, but not the type of elements to be retrieved. In this case, the IR system should be able to retrieve the most relevant elements.

Data types and vague predicates: Since XML allows for a fine grained markup of elements, there should be the possibility to use special search predicates for different types of elements. For example, for an element containing person names, a similarity search for proper names should be offered. Thus, there should be the possibility to have elements of different data types, where each data type comes with a set of specific search predicates; these predicates also may be vague in the sense that they also may return weights between 0 and 1.

Structural relativism: XPath is closely tied to the XML syntax, but syntactically different XML constructs may express the same information. Thus, appropriate generalizations should be included in the query language.

In the following, we describe how these features have been integrated in XIRQL.

3.2 Weighting

Classical IR models have treated documents as atomic units, whereas XML suggests a tree-like view of documents. In order to develop weighting formulas for structured documents, we generalize the classical weighting formulas. The basic idea is to apply the classical weighting formulas to "atomic" units in XML documents. In addition, we need a combination rule for the case when larger units are retrieved.

We start from the observation that text is contained in the leaf nodes of the XML tree only. So these leaves would be an obvious choice as atomic units. However, this structure may be too fine-grained (e.g. markup of each item in an enumeration list, or markup of a single word in order to emphasize it). A more appropriate solution is based on the concept of *index objects* from the FERMI multimedia model [Chiaramella et al. 96] Given a hierarchic document structure, only nodes of specific types form the roots of index objects. In the case of XML, this means that we have to specify the names of the elements that are to be treated as index nodes. This definition can be part of the XML schema (see below).

From the weighting point of view, index objects should be disjoint, such that each term occurrence is considered only once. On the other hand, we should allow for the retrieval of results of different granularity: For very specific queries, a single paragraph may contain the right answer, whereas more general questions could be answered best by returning a whole chapter of a book. Thus, nesting of index objects should be possible. In order to combine these two views, we first start with the most specific index nodes. For the higher-level index objects comprising other index objects, only the text that is not contained within the other index objects is indexed. As an example, assume that we have defined `section`, `chapter` and `book` elements as index nodes in our example document; the corresponding disjoint text units are marked as dashed boxes in Figure 1.

So we have a method for computing term weights, and we can do a relevance based search. Now we have to solve the problem of combining weights and structural conditions. For the following examples, let us assume that there is a comparison predicate `cw` (contains word) which tests for word occurrence in an element. Now consider the query

```
//section[heading cw "syntax"]
```

and assume that this word does not only occur in the heading, but also multiple times within the same index node (i.e. section). Here we first have to decide about the interpretation of such a query: Is it a content-related condition, or does the user search for the occurrence of a specific string? In the latter case, in would be reasonable to view the filter part as a Boolean condition, for which only binary weights are possible. We offer this possibility by providing data types with a variety of predicates, where some of them are Boolean and others are vague (see below).

For the content-related interpretation, we think that the context should never be ignored in term weighting, even when structural conditions are specified; these conditions should only work as additional filters. So we take the term weight from the index node. Thus the index node determines the significance of a term in the context given by the node. For computing the weight of a term in an index node, we apply standard weighting schemes like e.g. tf·idf (treating index nodes like atomic documents).

With the term weights defined this way, we have also solved the problem of independence/identity of probabilistic events: Each term in each index node represents a unique probabilistic event, and all occurrences of a term within the same node refer to the same event (e.g. both occurrences of the word "syntax" in the last section of our example document represent the same event). Assuming unique node IDs, events can be identified by event keys that are pairs [node ID, term]. For retrieval, we assume that different events are independent. That is, different terms are independent of each other. Moreover, occurrences of the same term in different index nodes are also independent of each other. Following this idea, retrieval results correspond to Boolean combinations of probabilistic events which we call event expressions. For example, a search for sections dealing with the syntax of XPath could be specified as `//section[.//* cw "XQL" and .//* cw "syntax"]`
Here, our example document would yield the conjunction $[5, \text{XQL}] \wedge [5, \text{syntax}]$. In contrast, a query searching for this content in complete documents would have to consider the occurrence of the term "XQL" in two different index nodes, thus leading to the Boolean expression
$([3, \text{XQL}] \vee [5, \text{XQL}]) \wedge [5, \text{syntax}]$.

For dealing with these Boolean expressions, we adopt the idea of event keys and event expressions described in [Fuhr & Rölleke 97], where we also show how correct probabilities can be computed for arbitrary event expressions.

In [Fuhr & Großjohann 01], we describe how this approach can be easily extended in order to allow for query term weighting. Assume that the query for sections about XQL syntax would be reformulated as
`//section[0.6 · .//* cw "XQL" + 0.4 · .//* cw "syntax"]`.
For each of the conditions combined by the weighted sum operator, we introduce an additional event with a probability as specified in the query (the sum of these probabilities must not exceed 1). Let us assume that we identify these events as pairs of an ID referring to the weighted sum expression, and the corresponding term. Furthermore, the operator '$\cdot$' is mapped onto the logical conjunction, and '+' onto disjunction. For the last section of our example document, this would result in the event expression $[q1, \text{XQL}] \wedge [5, \text{XQL}] \vee [q1, \text{syntax}] \wedge [5, \text{syntax}]$. In addition, we assume that different query conditions belonging to the same weighted sum expression are disjoint events and thus the final probability is computed as the scalar product of query and document term weights:
$P([q1, \text{XQL}]) \cdot P([5, \text{XQL}]) + P([q1, \text{syntax}]) \cdot P([5, \text{syntax}])$.

3.3 Relevance-oriented Search

Above, we have described a method for combining weights and structural conditions. In contrast, relevance-based search omits any structural conditions; instead, we must be able to retrieve index objects at all levels. The index weights of the most specific index nodes are given directly. For the

retrieval of the higher-level objects, we have to combine the weights of the different text units contained. For example, assume the following document structure, where we list the weighted terms instead of the original text:

```
<chapter> 0.3 XQL
  <section> 0.5 example </section>
  <section> 0.8 XQL 0.7 syntax </section>
</chapter>
```

A straightforward possibility would be the OR-combination of the different weights for a single term. However, searching for the term 'XQL' in this example would retrieve the whole chapter in the top rank, whereas the second section would be given a lower weight. It can be easily shown that this strategy always assigns the highest weight to the most general element. This result contradicts the structured document retrieval principle mentioned before. Thus, we adopt the concept of augmentation from [Fuhr et al. 98]. For this purpose, index term weights are downweighted (multiplied by an augmentation weight) when they are propagated upwards to the next index object. In our example, using an augmentation weight of 0.6, the retrieval weight of the chapter wrt. the query 'XQL' would be $0.3 + 0.6 \cdot 0.8 - 0.3 \cdot 0.6 \cdot 0.8 = 0.596$, thus ranking the section ahead of the chapter.

For similar reasons as above, we use event keys and expressions in order to implement a consistent weighting process (e.g. equivalent query expressions should result in the same weights for any given document). In [Fuhr et al. 98], augmentation weights (i.e. probabilistic events) are introduced by means of probabilistic rules. In our case, we can attach them to the root elements of index nodes. Denoting these events as index node number, the last retrieval example would result in the event expression $[1, \mathrm{XQL}] \vee [3] \wedge [3, \mathrm{XQL}]$.

3.4 Data Types and Vague Predicates

Given the possibility of fine-grained markup in XML documents, we would like to exploit this information in order to perform more specific searches. For the content of certain elements, structural conditions are not sufficient, since the standard text search methods are inappropriate. For example, in an arts encyclopedia, it would be possible to mark artist's names, locations or dates. Given this markup, one could imagine a query like "Give me information about an artist whose name is similar to Ulbrich and who worked around 1900 near Frankfort, Germany", which should also retrieve an article mentioning Ernst Olbrich's work in Darmstadt, Germany, in 1899. Thus, we need *vague predicates* for different kinds of data types (e.g. person names, locations, dates). Besides similarity (vague equality), additional datatype-specific comparison operators should be provided (e.g. 'near', $<$, $>$, or 'broader', 'narrower' and 'related' for terms from a classification or thesaurus). In order to deal with vagueness, these predicates should return a weight as a result of the comparison between the query value and the value found in the document.

The XML standard itself only distinguishes between three datatypes, namely text, integer and date. The XML Schema recommendation [Fallside 01] extends these types towards atomic types and constructors (tuple, set) that are typical for database systems. For this purpose, various type-checking mechanisms are provided, which operate at the syntactic level.

However, for IR applications, this notion of data types is of limited use. This is due to the fact that most of the data types relevant for IR can hardly be specified at the syntactic level (consider for instance names of a geographic locations, or English vs. French text). In the context of XIRQL, data types are characterized by their sets of vague predicates (such as phonetic similarity of names, English vs. French stemming). Thus, for supporting IR in XML documents, there should be a core set of appropriate datatypes and there should be a mechanism for adding application-specific datatypes.

As a framework for dealing with these problems, we adopt the concept of datatypes in IR from [Fuhr 99], where a datatype T is a pair consisting of a domain $|T|$ and a set of (vague comparison) predicates $P_T = \{c_1, \ldots, c_n\}$. Like in other type systems, IR data types should also be organized in a type hierarchy (e.g. Text – Western_Language – English), where the subtype restricts the domain and/or provides additional predicates (e.g. n-gram matching for general text, plus adjacency and truncation for western languages, plus stemming and noun phrase search for English). Through this mechanism, additional data types can be defined easily by refining the appropriate data type (e.g. introduce French as refinement of Western_Language)[1], by restricting the domain $|T|$ or by extending the set of vague predicates P_T.

In order to exploit these data types in retrieval, the data types of the XML elements have to be defined. For this purpose, we employ XML Schema, but use mainly the application info (which is treated like comments by the XML schema processor) for enumerating the vague predicates of a data type.

3.5 Structural Relativism

Since typical queries in IR are vague, the query language should also support vagueness in different forms. Besides relevance-based search as described above, relativism wrt. elements and attributes seems to be an important feature. The XPath distinction between attributes and elements may not be relevant for many users. In XIRQL, `author` searches an element, `@author` retrieves an attribute and `~author` is used for abstracting from this distinction.

Another possible form of relativism is induced by the introduction of data types. For example, we may want to search for persons in documents, without specifying their role (e.g. author, editor, referenced author, subject of

[1] Please note that we make no additional assumptions about the internal structure of the text data type (and its subtypes), like representing text as set or list of words.

a biography) in these documents. Thus, we provide a mechanism for searching for certain data types, regardless of their position in the XML document tree. For example, `#persname` searches for all elements and attributes of the data type persname.

Further abstraction from the concrete XML syntax is possible by introducing datatypes. For example, a date value can be represented in various forms in an XML document, as illustrated by the following example:

```
<date year="2001" month="12" day="11"/>
<date>2001-12-11</date>
<date><year>2001</year>
      <month>12</month>
      <day>11</day></date>
```

With the 'date' datatype, users just specify the date in a standard format in their query and don't need to know how dates happen to be represented in the current document class.

4 XML Retrieval and Information Extraction

Information extraction (IE) deals with the problem of extracting facts and knowledge from texts ([Crespo et al. 02]). Typically, for a certain type of information need, a template is defined, which contains a number of slots (attributes, fields). Then the IE system processes text documents in order to extract the requested information; as output, instances of the predefined templates (records) are created, where the slots are filled with appropriate values from the text. Like in IR, this task is burdened with the intrinsic uncertainty and vagueness of natural language and its processing (e.g. in the examples in Figures 2–3, Raphael is recognized as a Baroque artist). Since the output of an IE system cannot be perfect, many systems assign uncertainty values to the instantiated templates. In addition, weights may be also attached to single fields of a record.

Instead of creating instantiated templates, the IE system also can be used for automatic markup (AM) of texts. In this case, values filling slots are marked up as XML elements with the corresponding element name. In addition, the whole text belonging to an instantiated template is marked up as an element; since there may be additional text not belonging to any of the slots, the template element has mixed content.

Table 1 compares IE with AM. IE presents facts out of context, but allows for surveys over a number of instantiated templates (e.g. in a table) and also enables post-processing of the extracted facts for text mining. In contrast, AM leaves the facts within the context, so the human reader can easily detect recognition errors and also take into account additional information from the text related to the template (e.g. the date '17th century' is modified in the text to 'beginning of 17th century'). In addition, AM enables retrieval for

<Art> <Style> <Title>Baroque Art</Title> <Description> An art-historical term used both as an adjective and a noun to denote, principally, the style that originated in <Orig_Place>Rome</Orig_Place> at the beginning of the <Orig_Date>17th century</Orig_Date> superseding <Related_Styles>Mannerism</Related_Styles>. <Organisations>The Council of Trent <date>(1545-63)</date> </Organisations> had strongly advocated pictorial clarity and narrative relevance in religious art and to a degree Italian artists such as <Artist>Santi di Tito <date>(1536-1603)</date></Artist> had responded with a more simplified style which has been called <Related_Styles>'Anti-Mannerism'</Related_Styles>. Yet it was not until the <date>17th century</date>, with the grounds well of renewed confidence and spiritual militancy in the <Organisations>Counter-Reformation Catholic Church</Organisations> that a radical new style, the <Related_Styles>Baroque</Related_Styles>, developed. Rome was the most important centre of patronage at this period and the return to compositional clarity was facilitated by a renewed interest in the antique and the <Related_Styles>High Renaissance</Related_Styles> in the work of <Artist>Annibale Carracci</Artist> and his Bolognese followers, <Artist>Domenichino</Artist>, <Artist>Guido</Artist> <Artist>Reni</Artist> and <Artist>Guercino</Artist>. Their work is characterized by a monumentality, balance and harmony deriving directly from <Artist>Raphael<Artist>. </Description> <Source> </Style> </Art>

Fig. 2. Example text from an arts encyclopedia, with automatic markup

```
[Template: Art_Style]
Origination_Place: Rome
Origination_Date: 17th century
Organisations: Council of Trent, Counter-Reformation Catholic Church
Artists: Santi di Tito, Annibale Carracci, Domenichino, Guido
Reni, Guercino, Raphael
Related_Styles: Mannerism, Anti-Mannerism, High Renaissance
```

Fig. 3. Extracted information from example

aspects not covered by the template, in combination with conditions referring to specific slots (for example, we could search for artists mentioned in connection with the phrase 'religious art').

Information extraction	Automatic markup
facts out of context	facts in context
table-oriented view	document-oriented view
regular structure	irregular (text) structure
enables text mining	enables querying facts and (con)text

Table 1. Comparison of information extraction with automatic markup

For IE as well as for AM, the XML retrieval methods described above can be applied. Since uncertainty is a central concept of XIRQL, the uncertainty weights produced by the IE system can be considered during subsequent retrieval.

- In the IE case, instantiated records can be represented as XML documents. Here XIRQL allows for ranking of records based on uncertainty weights produced by the IE system. In case the IE system has assigned weights to single elements, XIRQL would consider only those weights belonging to elements referred to in the query. Another potential benefit results from the concept of vague predicates in XIRQL, where records with values similar to those specified in the query also can be retrieved.
- AM does not only pinpoint the role of certain pieces of text, it also allows for the application of appropriate data types. Both features can be exploited in order to increase the quality of text retrieval: whereas the former serves as precision device, the latter may be used for increasing recall. Again, uncertainty weights produced by the AM system can be considered during retrieval. Query conditions may refer to marked up parts of the text as well as to the remaining parts.

The current version of XIRQL allows for the retrieval of complete elements of XML documents only. In contrast, XQuery supports the restructuring of results and also provides some aggregation operators. Both of these features are useful for text mining. However, a straightforward extension of XIRQL by these operators is rather difficult, due to the uncertainty weights that XIRQL assigns to the elements of the result.

As a simple example, assume that we have searched for documents dealing with XML retrieval, and — among others — we have found two papers by the author Smith, one with probability 0.6 and the other with probability of 0.7. Now we would like to know the number of papers each author has written on this topic, i.e. count the number of documents per author. For Smith, 3 answers are possible: With probability $0.12 = (1 - 0.6) \cdot (1 - 0.7)$ of his papers is on this subject, the probability of two papers is $0.42 = 0.6 \cdot 0.7$, and with probability 0.46, there is exactly one paper by Smith. So we would end up with a probability distribution instead of a single value. Alternatively, we could compute the expected value (which is 1.3), but this is still a floating point number and not an integer as in the deterministic case. So, both solutions yield results that do not conform to the type of answers in the deterministic case. Using expectations may be more attractive, but further processing of these values may lead to inconsistent results (e.g. the product of two expectations is not identical to the expectation of the product of the corresponding variables). So there is no theoretically satisfying solution to this problem. Instead a more pragmatic approach could be used; for example, one could use the top k answers from XIRQL and treat them as deterministic answers for the following processing steps, for which e.g. XQuery could be employed.

5 Conclusions

In this paper, we have described the requirements of IR in XML documents, and we have presented the query language XIRQL which fulfills these needs.

For texts that are not available in XML format, information extraction and automatic markup methods can be applied. Whereas the former presents the information out of context and allows for further processing, the latter retains the context, thus enabling queries referring to both marked up facts and (con)text. In both cases, XIRQL is able to consider the uncertainty resulting from the preprocessing step. For text mining, further processing of the XIRQL results is necessary. Here appropriate methods for dealing with the uncertainty weights of the result elements still have to be developed.

References

[Chiaramella et al. 96] **Chiaramella, Y.; Mulhem, P.; Fourel, F.** (1996). *A Model for Multimedia Information Retrieval*. Technical report, FERMI ESPRIT BRA 8134, University of Glasgow. `http://www.dcs.gla.ac.uk/fermi/tech\_reports/reports/fermi96-4.ps.gz`.

[Clark & DeRose 99] **Clark, J.; DeRose, S.** (1999). *XML Path Language (XPath) Version 1.0*. `http://www.w3.org/TR/xpath`.

[Crespo et al. 02] **Crespo, A.; Jannink, J.; Neuhold, E.; Rys, R.; Studer, R.** (2002). *A Survey Of Semi-Automatic Extraction And Transformation*. `http://www-db.stanford.edu/~crespo/publications/extract.ps`.

[Fallside 01] **Fallside, D.** (2001). *XML Schema Part 0: Primer*. `http://www.w3.org/TR/xmlschema-0/`.

[Fernández et al. 01] **Fernández, M.; Marsh, J.; Nagy, M.** (2001). *XQuery 1.0 and XPath 2.0 Data Model*. `http://www.w3.org/TR/query-datamodel/`.

[Fuhr & Großjohann 01] **Fuhr, N.; Großjohann, K.** (2001). XIRQL: A Query Language for Information Retrieval in XML Documents. In: Croft, W.; Harper, D.; Kraft, D.; Zobel, J. (eds.): *Proceedings of the 24th Annual International Conference on Research and development in Information Retrieval*, pages 172–180. ACM, New York.

[Fuhr & Rölleke 97] **Fuhr, N.; Rölleke, T.** (1997). A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems. *ACM Transactions on Information Systems 14(1)*, pages 32–66.

[Fuhr 99] **Fuhr, N.** (1999). Towards Data Abstraction in Networked Information Retrieval Systems. *Information Processing and Management 35(2)*, pages 101–119.

[Fuhr et al. 98] **Fuhr, N.; Gövert, N.; Rölleke, T.** (1998). DOLORES: A System for Logic-Based Retrieval of Multimedia Objects. In: Croft et al. (ed.): *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 257–265. ACM, New York.

[Robie et al. 98] **Robie, J.; Lapp, J.; Schach, D.** (1998). XML Query Language (XQL). In: Marchiori, M. (ed.): *QL'98 — The Query Languages Workshop*. W3C. `http://www.w3.org/TandS/QL/QL98/pp/xql.html`.