

Indexing XML Documents: A Hybrid Approach

Kevin Reiss LIS 329 Fall 02

May 10, 2003

Contents

1	Introduction	2
2	Why Structured Documents?	2
3	Current Issues in Indexing and Searching XML	3
3.1	Navigating XML Documents	5
3.2	Indexing Strategies for XML	6
3.2.1	Database Strategies	6
3.2.2	IR Strategies	7
3.3	Ranking and Searching Strategies	8
3.3.1	XQuery	9
3.3.2	XPRES	9
3.3.3	XIRQL	10
3.3.4	ELIXIR	11
3.3.5	A Tree-Matching Approach	11
3.4	General XML IR Concerns	12
4	A Hybrid Structured Approach	13
4.1	The Full-Text Index	13
4.2	The Database	15
4.3	Proposed System Architecture	17
5	Open Source Software	19
5.1	Modified swish-e	20
5.2	eXist	22
5.3	Others	22
6	Conclusion and Future Work	23
	References	23

1 Introduction

Finding efficient and useful ways to search and index XML documents is a popular research topic in the fields of computer and information science today. There seems to be a clear division between two major areas of research on applications for the searching and indexing of XML documents, research focusing on data-centric applications and research focusing on document-centric applications(3).

Data-centric applications typically involve the processing on large amounts of regularly structured data, these can be e-commerce applications, interchange applications that transfer and translate information from one database schema to another, or what has become known as “web services” which involve packaging some sort of function, object or other entity in an XML wrapper to be delivered to another application over the network.(12). The sorts of XML structures produced by such applications are quite simple and typically do not persist as a permanent data store, but are rather used as a temporary means to store data in a regular, easily parsed format as it is exchanged between applications. In contrast, document-centric applications typically involve the storage of and processing of XML documents that are composed of loosely structured hierarchies that include mixed content, textual data, tabular data, and external binary entities such as figures, and other graphics formats.

This paper will focus on methods and research on the searching and indexing XML documents for document-centric applications such as IR (Information Retrieval). Examples of XML documents that could benefit from effective indexing and search strategies include medical case histories, species descriptions, journal articles, and technical documentation. A number of different XML query languages and indexing approaches have been developed for structured documents encoded in XML. The strengths and limitations of these approaches will be discussed. This discussion will build to a consideration of a proposed IR system for searching and indexing XML documents that can potentially extend the XML query languages and other approaches that will be discussed. Examples using a particular document class appropriate for a document-centric application will be used to give some context to the hypothetical system model that will be proposed¹. To conclude the paper consideration will be given to several current open source software projects that may be useful in creating a working prototype of the proposed system.

2 Why Structured Documents?

Structured documents have particular characteristics that make them fundamentally different than the types of documents that are found in data-centric applications. They are composed of what has been termed *semi-structured data* containing a large variety of different structures of varying data-types, includ-

¹The example DTDs and Document Instances used in this discussion appear courtesy of the BioBrowser Project, see <http://www.biobrowser.org> for further details. For the actual DTDs of the documents in the collection see figure 4.

ing both textual content and more regular data formats such as tables, numeric values, date values, or enumerated values.(5) The searching and indexing of structured documents has become one of the most popular areas of IR research as the World Wide Web continues to grow.(1) The descriptive markup approach that XML was developed from was first considered an advance for its ability to define the logical structure of a document thereby freeing it from the constraints of a particular formatting or processing system(13). While this was the initial impetus behind the descriptive approach, the logical structure of a document defined with a particular descriptive markup vocabulary can also be advantageously used by an information retrieval system to predict the relevance of a document to a query. Queries that take advantage of the document structure are known as structural queries and can take a number of forms, including path expressions and tree matching both of which will be discussed in section 3.(1).

How can an IR system best take advantage of structured documents? The goals of such an effort can be informed by considering the three purposes of indexing in IR as defined by Robert Korfhage(14):

1. To permit easy location of documents by topic
2. To define topic areas and hence relate one document to another
3. To predict relevance of a given document to a given information need

How well do current structured query and indexing models for XML fulfill these requirements? The ideal system seems to be one that will provide efficient and comprehensive indexing of document content and structure, and be able to support the predicted degree of relevance all matching documents have to a particular query. It is in the second consideration, the ability to provide ranking of matching documents, that many proposed XML indexing and searching systems fail to provide adequate support for.

3 Current Issues in Indexing and Searching XML

What is an XML Document? An XML document instance is a plain-text file that uses markup delimiters (tags) to define the logical structure of a document in a hierarchical fashion. The vocabulary of these delimiters can be defined in a schema that any particular document instance ascribing to adhere to the schema can be validated against. The most popular type of schema currently used for defining XML vocabularies is the DTD, (Document Type Definition). Whether or not a document instance is validated against a particular schema, the instance can be thought of as a tree data structure made up of any number of different nodes, see figure 1 for a graphical representation of this. Each node corresponds to a particular part of the logical structure of the document, and must be nested within its parent node. The root element of a document instance is the root node of the tree structure and every node within the tree, unless so constrained

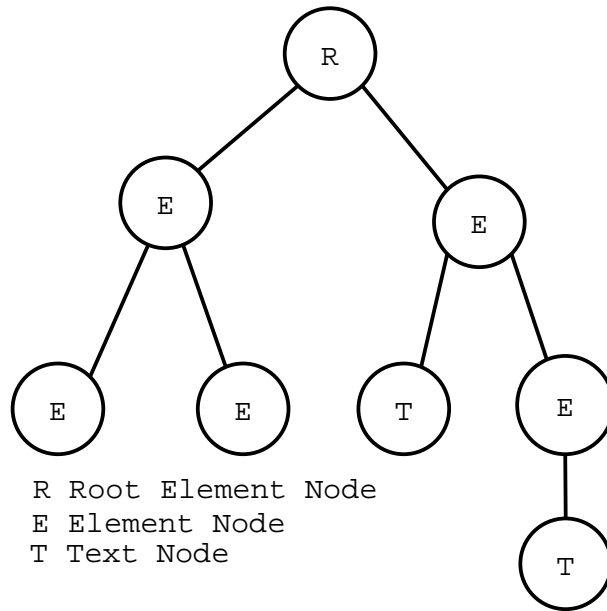


Figure 1: Example of an XML Document Tree

by a schema, can have any number of children and siblings but only one parent node.

When constructing an IR application that will facilitate the indexing and searching of such a data structure there are three main areas of consideration:

1. *Navigation* What are the different methods used to traverse the nodes in the XML tree structure. What are the different ways computer programs use navigation strategies to manipulate the different nodes in an XML tree structure, seen in figure 1. Which, if any of current strategies is most appropriate for IR applications?.
2. *Indexing* Methods of Indexing XML documents. What sorts of strategies are currently being implemented and studied that utilize the document structure in creating index of the document content? There are two major categories of XML indexing approaches, those coming from the database community, see section 3.2.1, and those coming from the IR community, see section 3.2.2.
3. *Ranking and Searching* What are current strategies for the effective ranking and searching of XML documents in IR systems informed by structured indexing? These strategies, see section 3.3, include a number of proposed query languages for XML that claim to support effective search of XML collections with the return of results weighted by relevance to a particular query. The strengths and weaknesses of these approaches for meeting the

goals of IR as described by Korfhage, see section 2, will be considered in the discussion of each approach.

Each of these will be discussed at length in the next section.

3.1 Navigating XML Documents

There are several different standards for navigating and selecting nodes in an XML document instance that one must consider when designing an IR system that facilitates the searching of XML documents containing semi-structured data. Developers and researchers seem to have settled upon 4 main standards for processing XML documents.

- *DOM* (Document Object Model) A W3C standard API for traversing and accessing the nodes in an XML document. An application using the DOM creates an in memory representation of the entire document tree while running. This makes it easily possible to search, manipulate and re-order the nodes of the tree. However it cannot be easily integrated into IR applications that depend on swift retrieval of information since the it does not scale well to the types of large deeply nested documents that many IR applications store.
- *SAX* (The Simple API for XML) A de-facto standard for treating the different nodes of XML document as a series of events. A program implementing SAX defines specialized handling methods that process a particular node whenever it is encountered by an application using SAX. SAX is an efficient way to process regular structured XML data similar to that encountered in data-centric applications, however it's application in document-centric activities such as IR is limited by the fact that it is often quite difficult and complex to do complex manipulations of a tree structure with SAX, making it hard to integrate in such activities as indexing.
- *XPath* A W3C standard that describes a data model for navigating and selected nodes in an XML document using path-based expressions that begin from the document's root. XPath expressions can have a modifying predicate that gives further specificity to the expression such as substring or child number of a particular node. Ex:

```
/FNA/Description[1]
```

This particular expression selects the the Description element that is the first child of an FNA element. XPath expressions, such as in the example, return a boolean value of either 0 or 1, either a particular path exists in a document or it does not. XPath expressions are used extensively to select nodes in XSLT², a transformation and selection language for XML documents that is often used in formatting the results of a query to XML IR system.

²see <http://www.w3c.org/Style/XSL/>

- *XQuery* XQuery a developing W3C effort to provide a query language capable of querying XML documents on the types of exact and range matching important in Database applications and typified by a Database query language such as SQL. However, there currently is nothing in the specification that would help support the ranking of query results, a document or a particular sub-tree selected has a weight of 0 or 1, it either is a match or it isn't a match.

How well does the use of one of these different paradigms support the IR concerns reflected in the opening quotes from Korfhage, see section 2? Notwithstanding the computational performance issues brought about by having to repeatedly search through large and potentially deeply nested structures that one typically finds in document-centric applications, the above descriptions are unable to provide information capable of making supporting relevance judgments on a returned document set from a particular query. All of the four navigation models described can only supply results with a boolean value. Even XQuery, the XML query language, does not provide an much support for IR applications besides allowing for the identification of a particular text strings that appear within text nodes of XML document instances. Fuhr and Großjohann point out, that this is the only 1 of the 19 proposed requirements of XQuery as listed by the W3C (6) as being to particularly to the needs of IR applications.(11) While all of these protocols could be an important part of how a particular IR system indexes, queries, and processes XML documents, to effectively create an a system that will provide both selection and relevance ranking of documents a system needs to use them as part of a comprehensive strategy, rather than serve as the sole means to get at an search content in XML documents.

3.2 Indexing Strategies for XML

3.2.1 Database Strategies

Quite a large number of indexing strategies have been proposed for XML documents. One of the most common is to map an XML structure to a relational database schema. This can be done by mapping the root element of an XML document to the name of a table residing within a RDBMs (Relational Database Management System). The child elements of the document are then mapped to the particular attributes of that the entity modeled by the table structure. However this approach only will work for less complicated documents that one might find in a data-centric application. The content model used for the XML would have to map directly to an entity represented in a relational table, with child elements corresponding to the attribute values that populate the rows of the table. A more sophisticated scheme to map XML documents to a database can be achieved by using an object oriented approach to modeling the structure of an XML document class and use the resulting objects hierarchy to permanently store the data from the documents in a relational or object-oriented database structure for future retrieval. Once a mapping of either type is in both of these approaches allow the content of the documents to be indexed and available for

searching under the guises of the well-established relational model.(2) While the relational model is very robust, it does not support the ranking of query results, making this solution unattractive as the sole indexing choice for an IR application.

In addition to using a traditional RDBMs for the purposes of storage and retrieval of document content, XML documents have also been indexed by systems that can be described as *Native XML Databases*. Some of these systems may ultimately use a RDBMs as the back-end storage device for the XML data, but the common characteristic to native XML databases is that the underlying data model for access, updating, deletion, and indexing is the XML tree structure of the documents being stored(19). There are a number of commercial software products, Ronald Bourret maintains an extensive catalog of XML database products³, which have been developed in recent years. However, Salmiinen and Tompa state that many of these products lacking in their support for the storage and retrieval of complex and rich XML document structures because of their limited support for enforcing the rules of a DTD or Schema that defines a particular document class(17). Without support for explicitly understanding what to expect in document instances being processed by an application the products have a difficult time working with the document structure, limiting their usefulness for IR applications.

3.2.2 IR Strategies

Both the native XML database products that have been developed and using a mapping of XML data to an RDBMs or and Object-Oriented Database system have serious limitations when one considers their prospects for supporting rich IR applications that are able to provide relevance measures to matching documents. Luk, et al conducted an extensive survey of XML indexing strategies from the IR community(15) They divided XML indexing strategies for XML documents in IR applications into the following different categories(15):

- *Flat-File Indexing* Flat-file indexing completely ignores the document structure and index the document in a as a typical full-text IR application, calculating the tf.idf value for each term found in the collection and storing the results in an inverted file structure.
- *Field-based Indexing* Each element in a document is treated like a field and the data in each element's content is mapped to the particular element (field) it occurs in. A field-based index is not aware of a document's parent-child relationships among nodes.
- *Segment-based Indexing* The document is divided up into segments, for example all the chapter segments are selected and the content within these segments is indexed. An index of this construction treats each segment as unique document to be queried and retrieved.

³see <http://www.rpbouret.com/xml/XMLDatabaseProds.htm>

- *Tree-based Indexing* A scheme that indexes the actual tree structure of an XML document including parent-child and sibling relationships among elements. These structures can then be incorporated in queries of the document's data. Navigation of the index is accomplished with XPath like expressions.
- *IR/DB Indexing* Structured approaches take approaches similar to those described in section 3.2.1 where XML content is mapped to another schema like a RDBMs or Object-Oriented System.
- *Path-based Indexing* Two index files are maintained when using Path-based indexing, one that indexes the text of the document and another that indexes the structure of the document. Each unique XPath that occurs in a document is indexed and given an identifier, the text that appears in the XPath is mapped to the structural index via this identifier. Path based indexing approaches can frequently be of use to ranking schemes for documents as path frequency can be calculated.
- *Position-based Indexing* The document is regarded as a two-dimensional object, different element tags are regarded as rectangular regions and the content within them is indexed and mapped to that particular region. This has been regarded as advantageous because both a rendered version of the document instance and the original can be indexed used the same indexing scheme.
- *Multidimensional Indexing* Traditional indexes are based upon indexing one attribute, such as the path expressions or words in a text, this approach indexes documents on multiple values. A variety data structures can be used to create a multidimensional index of a structured document.

All these approaches have particular advantages and disadvantages when considered in the context of structured documents that contain semi-structured data. Some do not capture the richness of such documents by ignoring or inadequately representing the hierarchical document structure defined by XML markup: Flat-file, Field-based and Segment-Based indexing. Others have problems providing useful information to ranking mechanisms and also have difficulty representing complex XML structures: IR/DB Indexing. Others suffer from performance lags when used on large collections of documents: Path-based, Position-based, and Multidimensional indexing.(15)

3.3 Ranking and Searching Strategies

Effective searching of indexed content and having the ability to rank the results from a particular query using a strategy common to one of the established IR models for ranking, including VSM, Probabilistic, Weighted Boolean, are common desires among IR researchers working on XML Query languages. Many authors also seem to feel that it is important to be able to rank not only entire

documents, but also assign rankings to particular subtrees within a structured XML document that could possibly be returned as a match to a given query. The following sections consider 5 proposed or experimental XML Query Languages under current development and consider the prospects of each one to respond to the challenges of providing IR applications with the ability to rank the matching documents to a structured query and also to retrieve particular subtrees of an XML document as a matching document to a query.

3.3.1 XQuery

XQuery is the product of ongoing efforts by the W3C⁴ to describe a query vocabulary for XML documents that provides similar functionalities for searching XML documents that SQL⁵ provides for relational structures stored in RDBMs. The developing XQuery specification has been based from two earlier proposals of SQL-like XML Query Languages, XQL(16) and QUILT(7). XQuery provides ample support for exact and range matching of values in an XML document(6). It also provides strong support for the incorporation of the structure of the document instance within queries to XQuery aware application. However none of these languages addresses the need to provide relevant weighted results to the a query. Additionally they do not support approximate or partial matching. Many researchers point out that these two limitations negatively impact XQuery's ability to make a significant impact in the context of a rich IR system for an XML documents(20)(18)(8)(11).

3.3.2 XPRES

Wolff, Florke and Cremers define an experimental query language, *XPRES* (eXtended Probabilistic Retrieval to Structure, that extends the classic probabilistic BIR (binary independence retrieval) IR model(1), to include structural information in the calculation of similarity measures for matching documents to a particular structural query(20). Several modifications to the probabilistic model are made to incorporate document structure. Instead of using the Inverse Document Frequency (IDF) as the basis for calculations the authors define a concept termed "Inverted Element Frequency"(20) that calculates the frequency of a term's occurrence within a particular element in the document structure. The IEF is used in similarity valuation similiar to the tf.idf measurement in traditional IR models. Weighting of query results in XPRES can also be influenced by defining the "role" of an element, possibly by the user of the sytem. The user can define differing elements to have the same or different roles in the context of the query. Those elements that are defined to be of the same role will be given the same weight in the ranking algorithm for query results, those with a higher role will have a higher weight in figuring rank of the query results.(20)

The index for the *XPRES* consists of two parts, a text index and a structure index. The text index actually contains the IEF for each term along with other

⁴<http://www.w3c.org/XML/Query>

⁵Structured Query Language

lexical information about the term. The structure index contains information about the particular subtrees that occur in a document, an identifier for each of these unique trees is mapped to each term in the text index that appears in the subtree(20). This approach provides an interesting way to achieve ranked query results using the probabilistic IR model, but it does not provide the means to deal with data-types other than strings of text, such as numeric or range values. XPRES has been successfully tested on a large collection of documentations taken from the PubMed database, initial tests demonstrate that the precision of the system is improved by the incorporation of structural information to the index and query structure(20).

3.3.3 XIRQL

Fuhr and Großjohann propose *XIRQL* an XML indexing model and query syntax that will allow for the following(11):

- Weighting
- Relevance-Oriented Search
- Data-Types and Vague Predicates
- Semantic Relativism

This approach extends the original XQL Query Language(16) to support the IR constructs in the previous list. Particular nodes in a document are defined as the index points for the document and the content within these nodes and child nodes of these nodes are stored in an index file containing each unique path that occurs within any defined index node(11). Each unique index node that occurs in a document instance is considered a unique probabilistic event by the ranking algorithm proposed by Fuhr and Großjohann(11). Those nodes(elements) defined as index terms should be indicated as such in the Document Type Definition of the particular document class being indexed by a XIRQL aware application(11). It is worth noting that Fuhr and Großjohann state only valid documents should be used in a XIRQL application, in contrast to the approaches taken by XQuery which is explicitly designed to work with any well-formed XML document(11).

A weight is assigned to each unique path in relation to the frequency it occurs in a particular document, terms that are higher up in the document tree are given a lower weight in Fuhr and Großjohann's scheme because it is assumed that a finer granulated index path will select a more relevant document within the applications weighting scheme(11). A predicate can be added to any search path supplied by the application to further narrow the search. This predicate can search for a particular range of path values, or other user-defined data-types such as proper Names, dates, thesaurus or index terms(11).

Semantic relativism would allow the query to expand the search term that ends a particular path by stemming or filling words similar to the term, however this has not been introduced by the authors into any of the test applications that have used XIRQL(11). The language is an advance on XQuery in that it allows ranking of query results using the frequency of terms within pre-defined index

nodes within the document structure, but it has not yet been put through any rigorous tests so it is not known how robustly an application using this approach will perform.

3.3.4 ELIXIR

Chinenyanga and Kushmerick propose another extension of the original XQL proposal(16), called *ELIXER*, the expressive and efficient language for XML information retrieval(8). This approach utilizes a subset of XQL(16) expressions to search collections of XML documents. For any given XQL expression a document, or sub-tree within a document if the query is specified at that level is either relevant to the expression or not(8). This search yields a set of matching documents with equal weights of 1. If a query in ELIXER form specifies a similarity predicate that instructs the application to determine how similar a resulting set of documents is to the query, the matching documents are run through a WHIRL(9) system that uses IR techniques to create a similarity measure. The weight of each term vector is calculated and similarity is derived by calculating the cosine similarity between the documents among the results set and the query specified by the user(8).

This approach could conceivably support range and exact matching using XQL syntax, and relevance scores can be achieved by the presence of a similarity operator that invokes the WHIRL system that evaluates the similarity of the returned document set every time. However the lack of an index system limits this query language. In diagnostic tests the cost of doing the ELIXER search than calling WHIRL to compute the similarity of the matching documents on the fly was shown to increase linearly as the size of the document collection went up(8). An effective indexing approach would have to be devised to make this approach attractive in a large-scale application.

3.3.5 A Tree-Matching Approach

Schlieder and Meüss describe a different indexing and searching approach that differs from the previous four that have been examined. They suggest an indexing approach that indexes every unique tree pattern of nodes that occurs within an XML document and utilizes a tree-matching algorithm to compute the similarity of documents to a given query combining both a path description of the document structure and particularly search terms(18). The model uses the basic VSM, Vector Space Model(1), as the basis for arriving at similarity values between a specified structural query and those documents containing tree patterns that approximately match the query as specified(18).

Instead of interpreting a query as a list of terms as is done in the classic VSM model, the queries in this approach is considered a labeled tree, that consists of a label containing a path statement that describes the tree structure that the desired term should appear in. The documents in the model of Schlieder and Meuss are not actual separate document instances but rather sub-trees of one large document, since their model assumes the entire XML document collection

to be one tree with each sub-tree representing a particular logical document. The index of such a collection is created by identifying each unique structured term in the collection. The authors define a structured term as a unique labeled tree, matched with the actual textual terms that occur in the tree(18)

An engine implementing this approach must be able to handle the processing of both of the following(18):

1. The *matches* of all query subtrees (which represent occurrences of structural terms) into document collection must be found.
2. The *weights* of all structural terms that have occurrences in both the query and the document collection must be calculated

In their test implementation Schlieder and Meuss create an inverted index that contains references to all nodes(logical documents) within the collection that contain a particular structural term. A tree-matching algorithm than takes the query, specified in the form of a structured term itself, and matches all subtrees within the document collection that contain the structured term. A ranking algorithm than uses the index information to create a standard tf.idf measure for the returned subtrees. This information is than used to calculate VSM measures like cosine similarity and euclidean distance to obtain a ranked result set for the subtrees that have been identified as relevant logical documents within the collection.(18)

The tree-matching approach was reported to be tested on a small document collection with favorable effect on precision and that also recorded an efficient processing time(18). This was especially if the structured terms specified in the query were not semantically similar(18). This strategy proposes a complex and interesting approach that fully incorporates the logical structure as defined by XML into index, query, and ranking structures. However it cannot meet the full demands of searching semi-structured data because it can only deal with approximate matching and does not make provision for range and exact searching of particular data-types because although the structure of the document is fully accounting for in the indexing scheme, the data contained within the document is treated as text, an fact that the authors acknowledge in the work published on their proposed model(18).

3.4 General XML IR Concerns

The previous considerations lead one to the conclusion that there is something lacking in current XML indexing strategies. While there is some work being to done to devise ranking strategies and much work being done to provide database like queries for XML documents the work that has been done attempting to combine the two approaches into an IR system that allows for full-text structural searching and database like range and exact searching is just starting. However an approach that combines the strengths of both of these two different approaches has not yet been identified. To adequately support the retrieval and ranking of semi-structured data of varying data-types a system that imple-

ments both approaches, full-text structural indexing and database like queries, is needed. A system for doing so will be proposed in the next section.

4 A Hybrid Structured Approach

Often times it would be useful to be able to engage all three types of the common matching that is necessary in Information and Data Retrieval systems(14):

- approximate matching
- exact matching
- range matching

An ideal IR system for semi-structured data stored in XML document instances should be able to support all three types of matches. The system should be able to provide exact and range matching on numeric types and also be able to generate appropriately ranked lists of matching documents to a given query. Over the next several sections an example where these abilities could prove useful will be considered. The example takes the form of a species description taken from the Flora of North America⁶, it contains semi-structured data that is particularly well-suited to demonstrate why it useful to support all three types of matching in an IR system. Some of the text within the document instances should be treated as a numeric values, but other portions should be treated as textual data. For this to actually be enforced an XML Schema⁷ should be defined to enforce data-typing on the value of particular element values that need to occur only as certain types, currently only a document type definition is in existence for the example document class.

A system that could potentially incorporate all three modes of matching if it had access to a full-text structural index of textual data and had access to a database that contained all the data in the documents that requires strong typing. Using the example document class mentioned in the previous paragraph the next two sections 4.1 and 4.2 consideration will be given as to why this could be a potential way to implement all three forms of matching within an IR system that is still able to rank matched documents to the user by their relevance to the given query.

4.1 The Full-Text Index

A typical full-text index of an XML document records information about the frequency of occurrence of different paths in the document. XPath expressions typically serve as the models for path selection in such cases. For example consider the document fragment:

⁶see figure 4 for the DTD for the FNA document class

⁷<http://www.w3.org/TR/xmlschema-0/>

```
<!-- species_example.xml -->
<FNA>
<Description>Trees to 23m; trunk to 0.6m diam.;
crown spirelike. Bark gray, thin, smooth, in
age often becoming broken into irregular brownish
scales. Branches diverging from trunk at right angles,
the lower often spreading and drooping; twigs
mostly opposite, greenish brown, pubescence sparse.
</Description>
</FNA>
```

When a path aware, see section 3.2.2, full-text indexing engine looks at this document the words that are recognized as valid index terms will be represented in the index with the appropriate path expression to their location in the document tree:

```
/FNA/Description/crown      species_example.xml 1
/FNA/Description/spirelike  species_example.xml 1
/FNA/Description/0.6m      species_example.xml 1
```

The number of times a particular path occurs in a document is also recorded and a weighting measure like *tf.idf* can be easily applied to the contents of the index to obtain a ranking of documents in relation to a particular query. While this is effective in creating a full-text index that also is aware of the document structure, all of the element content is treated as text and if the following path-aware boolean query is run on the index containing the above data:

```
(FNA/Description['spirelike'] and FNA/Description['trunk'])
and FNA/Description['0.4m']
```

The query will be unsuccessful because the third path expression in the query does not explicitly match the element content. For it to be successful the query would have to be:

```
(FNA/Description['spirelike'] and FNA/Description['trunk'])
and FNA/Description['0.6m']
```

Unfortunately the above problem cannot be solved by including further markup granularity. Consider markup that introduces a height and diameter element for the numeric types:

```
<FNA>
<Description>Trees to <height>23m;</height>
trunk to <diameter>0.6m</diameter> diam.;
crown spirelike. Bark gray, thin, smooth, in
age often becoming broken into irregular brownish
scales. Branches diverging from trunk at right angles,
the lower often spreading and drooping; twigs
mostly opposite, greenish brown, pubescence sparse.
```

```
</Description>
</FNA>
```

Even with further markup granularity a full-text path-aware indexing approach will still fail to find a match on the following boolean query, because the element content is still being treated like text:

```
(FNA/Description['spirelike'] and FNA/Description['trunk'])
and FNA/Description/diameter['0.4m']
```

To actually get a match to this query the ability to perform a range match on the diameter value is needed. The query should be able to read as something akin to this:

```
(FNA/Description['spirelike'] and FNA/Description['trunk'])
where FNA/Description['RL < 0.4 < RU']
L Range Lower Bound
U Range Upper Bound
```

4.2 The Database

A database should contain the values, such as the content of the diameter element in section 4.1, that need to be strongly typed as numeric, range, or enumerated values. The full-text index will index the other elements whose values should be treated as text. A defined similarity function using a modified tf.idf value that includes path frequency will calculate the ranking of matching documents to a query returned from this structure, see figure 2 to see the pieces required and the process that will be used to construct these two aspects of the system. It is important to note that a DTD/schema will be required, this is necessary for indexing efficiency and accuracy as the indexing engine can expect to receive only content models and path expressions corresponding to the particular document class that is being indexed.

A modified DTD for the application can be found in figure 5, in actuality a schema would need to be devised to strictly enforce the data-typing on the appropriate values. The markup for a document in the data-type aware application backed up by both the database and the full-text index will now look like this:

```
<FNA>
<Description location='ft'>Trees to
<height location='dbase' numtype='range'
start='0' end='23'
value='meters'>23</height>
trunk to <diameter location='dbase' numtype='range'
start='0' end='.6' value='meters'>0.6</diameter> diam.;
crown spirelike. Bark gray, thin, smooth, in
age often becoming broken into irregular brownish
scales. Branches diverging from trunk at right angles,
```

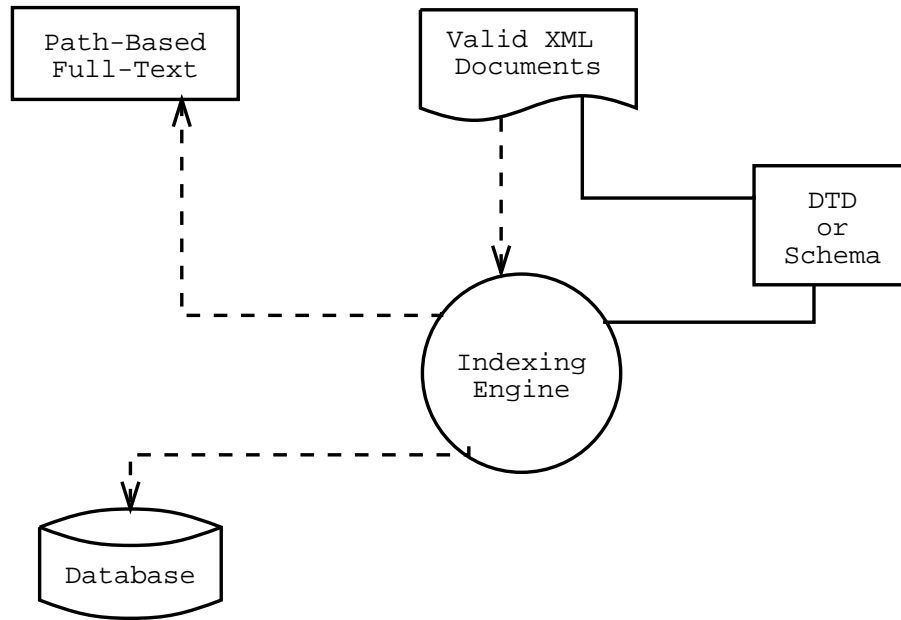


Figure 2: Indexing System

the lower often spreading and drooping; twigs mostly opposite, greenish brown, pubescence sparse.
 </Description>
 </FNA>

- Attribute “location” points the query engine to look in either the full-text index or the database.
- Attribute “numtype” indicates that the value in the element should be evaluated as a range starting at Zero, implied if to lower bound is specified.
- Attribute “value” indicates exactly what the number in the element contents represent.

The system will now be able to provide exact and range matching of numeric types along with full-text searching and the ability to provide effective ranking of the matching documents to the query, see figure 3 for details. See figure 5 for a representation of the modified DTD with the new attribute values that point the indexing engine to the proper data-store, the full-text index or the database, see figure 2 for details.

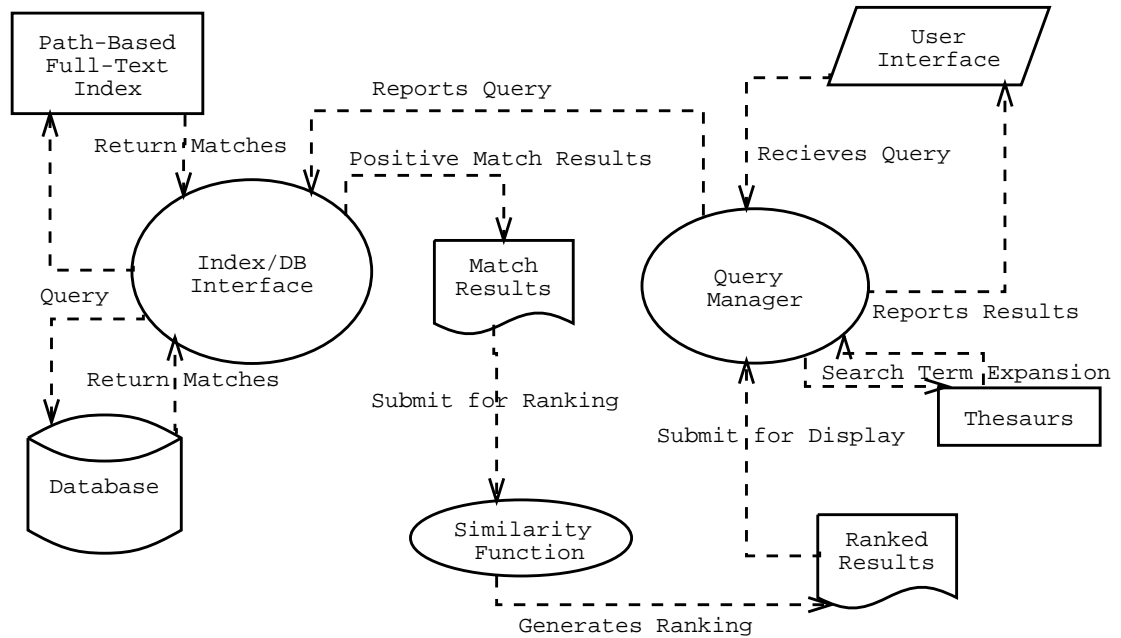


Figure 3: Hybrid IR System

4.3 Proposed System Architecture

The previous sections consider an example query that can't be met by a full-text indexing approach alone, then describe an ideal indexing system, see figure 2 that could meet the demands of a query that requires both a lookup of a data-typed range value and a full-text search with results ranked by relevance. This section will discuss a theoretical implementation of such an approach.

The system must have two items available:

1. a collection of valid document instances, each with an automatically assigned unique identifier
2. a schema capable of enforcing data-typing on element content where it is required

Figure 3 diagrams the flow of information between the different parts of the system. There are two data-stores that will be necessary:

1. *Full-Text Index* Consists of a collection of every unique path that occurs in a document instance. Every particular element node can be indexed or only those nodes selected as index nodes will have their textual content indexed by the indexing engine, see figure 2. Only valid document instances will be accepted by the indexing engine, see figure 2. An identifying schema must be specified in order to tell the indexing engine what content

begins and ends a path within the document. The frequency of each unique path will be stored along with the unique document identifier. In the interest of simplicity and disk-space frequencies will be counted at the traditional document-level of retrieval units, as opposed to the ability to select a particular portion of a document instance as a matched document, something that a number of researchers talked about, see section 3.3. The indexing engine should also carry out the commonplace IR activity of stemming, storing paths containing the root words of terms found in the document to expand the matching possibilities available.

2. *Database* The database will contain the those elements that it is necessary to enforce data-typing, such as explicit numeric or explicit range values. It also could contain element content that must be chosen from an enumerated list of options. The unique document ID will serve as the key-field for the database table with each element that is marked for the database contained in a particular attribute value mapped to the element name.

In addition to the indexing engine, see figure 2, which will have to applied every time an update is made to the document collection, there will be four processes that will have to be continuously running while the IR system, see figure 3 is in operation. This four parts may actually be implemented on the same level in an actual system implementation, but their roles are distinct and how each will be accomplished will merit strong consideration when actually implementing the system. These four parts are:

1. *User Interface* The user interface should provide the user with a clear indication of how to formulate a structural query to take best advantage of the indexing scheme and the similarity measures that can be generated by the system. In particular it might give users the option to place a higher weight on specific structural constructs. For example, a document in the example document class that contains a particular path in which a search term resides in the name element, see figure 5, may be given a higher weight than if the search term resides in a different path. The interface should provide the user the option to weight terms in such a manner. The interface should also provide the user of an indication of what fields may be searched using numerical or other enumerated values stored in the database. The interface should be extensible written to accept any give schema or DTD and configure itself with minimal intervention by the application developer. The interface should also have a logical and intuitive display mechanism for ranked query results.
2. *Query Manager* The query manager validates queries from the user and transforms them into a form that the full-text/DB interface can translate into a query of the actual data-stores. The Query Manager should translate queries written by the user into the proper path notation that is ultimately decided upon for the application and used by the full-text/DB interface to search the data-stores. These queries should be able to be

able to support boolean expressions, thesaurus expansion of terms, and stemming. The query manager should also package results from a query into suitable form for acceptance by the interface. The query manager should also hold responsibility for validating that the similarity function was properly run on the match results set generated from the query processing by the DB/Full-text interface.

3. *Similarity Function* The similarity function should use the values stored in each index entry for a unique query term for a document, see article about structured query terms insert REFERENCE here. This function should compute the rankings for documents that have been returned as matches to the user's query according to relevance of the document to the specified query. This function could use the tf.idf values stored for each unique path in the index to arrive at a measure of cosine similarity or euclidean distance as a means to obtain a ranking. The ranked results should be then packaged to the query manager that has the responsibility of getting them ready to be sent to the user interface.
4. *DB/Full-Text Interface* This is the actual interface to the two data-stores. This interface does the work of actually retrieving the data from these structures. When it receives a valid query statement from the Query Manager it should be able to decide which data-store to look at first. The database should be always be consulted first, since it should be the most efficient means to remove documents from the result set since the match must be exact or within a certain range. Once these documents have been removed the more computationally expensive task of approximate matching will be undertaken using the full-text index. This interface should return the resulting documents as an appropriate data structure that the similarity function can use to provide a ranked results list.

This system description provides a plan for constructing an XML indexing system that can support a rich IR application that handles exact, range and approximate matching.

5 Open Source Software

A number of different Open Source Projects are currently experimented with implementations of some of the XML query languages discussed in section 3. These projects take one of the three approaches discussed earlier:

- Construct a full-text index file of the document by counting path and index term frequency within those paths
- Construct a native XML data-store queried by an XML query vocabulary
- Construct a relational database based upon by the XML schema

```

<!-- DTD for FNA    Hong Cui    2001 -->
<!-- generated to match classified FNA record segments -->

<!ELEMENT FNA      (NomenclaturalInfo*, Description*, Distribution*, Discussion*,
Images?, ImagesMap?, Copyright?, Other?, Variability?, References0?, References1?)>
<!ELEMENT NomenclaturalInfo (#PCDATA)>
<!ELEMENT Description      (#PCDATA)>
<!ELEMENT Distribution      (#PCDATA)>
<!ELEMENT Discussion        (#PCDATA)>
<!ELEMENT Images            (#PCDATA)>
<!ELEMENT ImagesMap         (#PCDATA)>
<!ELEMENT Copyright         (#PCDATA)>
<!ELEMENT Other             (#PCDATA)>
<!ELEMENT Variability       (#PCDATA)>
<!ELEMENT References0       (#PCDATA)>
<!ELEMENT References1      (#PCDATA)>

```

Figure 4: The Original Document Type Definition

An open-source project that had support for computing relevance was difficult to find. The following products to be discussed, with the exception of *Swish-e* were located through the frequently updated list of XML Database Products that Ronald Bourret maintains(4). With modification each one of these programs could serve as a piece of the system described in section 4.

5.1 Modified swish-e

*swish-e*⁸ could serve as the basis for the indexing engine, and provide a basis for developing an application query syntax. This program has been modified to index the full-text and each unique path that occurs in a document collection that has been validated by the same DTD by the BioBrowser project⁹. Swish-e can create an index entry for each unique path in the collection with a frequency count for each document instance. The index entries will consist of the unique path identifier and a count of it's frequency in each document, so this information can be exploiting for developing similarity measures. The modified swish index could be index to recognize those nodes whose contents should be stored in a database and an interface could be provided in the indexing engine proposed in figure 2 to handle the loading of these nodes into the database.

⁸see <http://swish-e.org/>

⁹see <http://www.biobrowser.org>

```

<!-- Modified DTD for FNA 2002 -->
<!-- generated to match classified FNA record segments -->

<!-- Insert attribute entity structure -->
<!ENTITY % coreattrs
"location      (ft|dbase)      #REQUIRED
 numtype       (range|exact) #REQUIRED
 start         CDATA          #IMPLIED
 end           CDATA          #IMPLIED
 value         CDATA          #IMPLIED''
>

<!-- ATTLIST definitions for all elements
      excluded for brevity -->
<!ELEMENT FNA      (NomenclaturalInfo*, Description*, Distribution*, Discussion*,
Images?, ImagesMap?, Copyright?, Other?, Variability?, References0?, References1?)>
<!ELEMENT NomenclaturalInfo (#PCDATA)>
<!ELEMENT Description      (#PCDATA|height|diameter)>
<!ATTLIST Description      %coreattrs;>
<!ELEMENT height          (#PCDATA)>
<!ATTLIST height          %coreattrs;>
<!ELEMENT diameter       (#PCDATA)>
<!ATTLIST diameter       %coreattrs;>
<!ELEMENT Distribution    (#PCDATA)>
<!ELEMENT Discussion     (#PCDATA)>
<!ELEMENT Images         (#PCDATA)>
<!ELEMENT ImagesMap     (#PCDATA)>
<!ELEMENT Copyright      (#PCDATA)>
<!ELEMENT Other          (#PCDATA)>
<!ELEMENT Variability    (#PCDATA)>
<!ELEMENT References0    (#PCDATA)>
<!ELEMENT References1    (#PCDATA)>

```

Figure 5: Modified Document Type Definition

5.2 eXist

*eXist*¹⁰ is a Java implementation of a Native XML database, see section 3.2.1 for further details on native XML databases. This package could possibly serve as a model to build upon in developing the query manager and full-text/DB interface portions of the proposed system. eXist implements a database server that accepts queries in the form of statements conforming to following three commonly utilized means of transporting information between applications:

- XML-RPC
- XML:DBI
- SOAP

The database server that exists in eXist accepts queries written in path-based syntax. These queries are actually carried out by a Broker Interface that queries the index file of the native XML data store maintained by the application. This broker also has an interface to a relational data store that is optionally in the current eXist package. Currently this option is made available to those who wish to back their data up by storing it in a database.

eXist could be extended to use the modified swish-e package to handle indexing and incorporate a similarity function like that discussed in section 3 to generate rankings among matches to a given query, since eXist does not support any sort of ranking or similarity measures in its indexing and searching system. It could also incorporate validation by schema into the processing process, since schema validation is important in the proposed system of section 4. eXist does provide XPath based query support and offers an interface into a relational and native XML database structure in the form of an index maintained on the documents stored in the system making it an intriguing package to consider for extension and modification to incorporate the system described in the previous section(4).

5.3 Others

Kweelt¹¹ is an open-source implementation of the XML query language Quilt(7), but it doesn't actually implement a relational or native XML structure of its own. An interface is provided to tie Kweelt to a particular structure or index using DOM, SAX, XML:DBI or another user defined language. However, Kweelt doesn't have a lot of active development going on at the present time. Kweelt's last version was released back in August of 2000, so it is not a good candidate for consideration when compared to eXist, which is being actively developed at the present time.

GNU Qexo¹² is a partial implementation of the proposed XQuery Language(10) by the W3C. Like Kweelt and eXist it is implemented in Java. It is implemented

¹⁰see <http://exist.sourceforge.net/features.xml.html> for current details

¹¹see <http://kweelt.sourceforge.net/>

¹²see <http://www.gnu.org/software/qexo/>

as part of the GNU Kawa project¹³, which is a java-based scheme framework that developers have adapted to XML tasks. While under active development, XQuery provides no ranking constructs for documents returned by a specific query and the Qexo project does not seem to be at a stage where it could provide enough viable support in the way of interfaces to RDBMs or Native XML databases to be a strong candidate for choice in helping to a develop a prototype of the proposed system.

6 Conclusion and Future Work

There is much work being done on the problem of how best to provide support for structural queries that facilitate the retrieval of information stored in XML documents by the IR and Database communities. However much of this work centers on data-centric applications, that represent simple XML document instances that generally do not contain complex structures in the form of semi-structured data as described by Peter Buneman(5). Those researchers who have focused on document-centric applications like IR systems, have done some substantial research on techniques to create indexing systems and similarity measures that support relevance rankings of matched documents to a given structural query. However, these projects are in their infancy and primarily focus on creating IR systems whose goal is only approximately matching documents to a query, but don't consider the problem of creating a system that will support the exact and range matching of numeric and other strongly typed document content.

Considering the wide variety of semi-structured data available in structured XML documents a need exists for a system that can provide database-like precision when searching strongly typed content within an XML document. A system that supports this type of searching should also still be able to accomplish the goal of providing the user with the ability to search the full content of documents and provide a list of matching documents to the query ranked by relevance. The system proposed in section 4 is a plan for such a system. It provides the functionality of both a full-text index containing all of the unique paths found within a particular document class and a database of data-typed and enumerated values. Such a system could potentially be implemented by modifying the open source software package discussed in section 5.

References

- [1] BAEZA-YATES, R., AND RIBEIRO-NETO, B. *Modern Information Retrieval*. ACM Press and Addison Wesley, 1999.
- [2] BOURRET, R. Mapping dtDs to databases. Web Document, May 2001. <http://www.xml.com/pub/a/2001/05/09/dtdtodbs.html>.

¹³see <http://www.gnu.org/software/kawa/>

- [3] BOURRET, R. Xml and databases. Web Document, November 2002. <http://www.rpbourret.com/xml/XMLAndDatabases.htm>.
- [4] BOURRET, R. Xml database products. Web Document, 2002. <http://www.rpbourret.com/xml/XMLDatabaseProds.htm>.
- [5] BUNEMAN, P. Semistructured data. In *Proceedings of the ACM SIGPODS (1997)*, Association of Computing Machinery, ACM Press, pp. 117–121.
- [6] CHAMBERLIN, D., FRANKHAUSER, P., MARCHIORI, M., AND ROBIE, J. Xquery: A query language for xml. Web Document, Feb 2001. <http://www.w3.org/TR/xmlquery-req>.
- [7] CHAMBERLIN, D., ROVIE, J., AND FLORESCU, D. Quilt: An xml query language for heterogenous data sources. In *Proceedings 3rd International Workshop on World Wide Web and Databases (2000)*, LNCS, pp. 1–25.
- [8] CHINENYANGA, T. T., AND KUSHMERICK, N. Expressive retrieval from xml documents. In *Proceedings of the ACM SIGIR (September 2001)*, The Association of Computing Machinery, ACM Press, pp. 163–171.
- [9] COHEN, W. Whirl: A word-based information representation language. Tech. rep., AT&T Labs, 180 Park Avenue Florhan Park, NJ 07932, November 1998.
- [10] FERNANDEZ, M., MALHATRA, A., MARSH, J., NAGY, M., AND WALSH, N. *XQuery 1.0 and XPath 2.0 Data Model*, working draft ed. World Wide Web Consortium, November 2002. <http://www.w3.org/TR/query-datamodel/>.
- [11] FUHR, N., AND GROSSBJOHANN, K. Xirql: A query language for information retrieval in xml documents. In *Proceedings of the ACM SIGIR (November 2001)*, Association of Computing Machinery, ACM Press, pp. 172–180.
- [12] GLASS, G. Applying web services to applications. Web Document, November 2000. <http://www-106.ibm.com/developerworks/library/ws-peer1.html?dwzone=components>.
- [13] GOLDFARB, C. A generalized approach to document markup. In *Proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation (1981)*, Association of Computing Machinery, ACM Press, pp. 68–73.
- [14] KORFHAGE, R. R. *Information Storage and Retrieval*, 1 ed. Wiley Press, 1997.
- [15] LUK, R. W., LEONG, H., DILLON, T. S., CHAN, A. T., CROFT, B., AND ALLAN, J. A survey in indexing and searching xml documents. *Journal of the American Society for Information Science and Technology* 53, 6 (2002), 414–437.

- [16] ROBIE, J., LAPP, J., SCHACH, D., HYMAN, M., AND MARSH, J. Xml query language (xql). Web Document, 1998. <http://www.w3.org/TandS/QL/QL98/pp/xql.html>.
- [17] SALMINEN, A., AND TOMPA, F. W. Requirments for xml document database systems. In *Proceedings of the ACM Workshop on Doucment Engineering* (November 2001), Association of Computing Machinery, ACM Press, pp. 85–94.
- [18] SCHLIEDER, T., AND MEÜSS, H. Querying and ranking xml documents. *Journal of the American Society for Information Science and Technology* 53, 6 (2002), 414–437.
- [19] STAKEN, K. Introduction to native xml databases. Web Document, October 2001. <http://www.xml.com/pub/a/2001/10/31/nativexml.db.html>.
- [20] WOLFF, J., FLORKE, H., AND CREMERS, A. Searching and browsing collections of structural information. In *Proceedings of IEEE Advances in Digital Libraries* (2000), IEEE, pp. 141–150.