First International Conference on Web Information Systems Engineering (WISE'00),
Proceedings, pp 62-69, June 19-21, 2000, Hong Kong, China, IEEE CS Press

XML Schema Directory: A Data Structure for XML Data Processing*

Evangelos Kotsakis
VTT Information Technology, P.O. Box. 1201, VTT ESPOO, Finland,
kotsakis@acm.org

Klemens Böhm
Institute of Information Systems, Swiss Federal Institute of Technology, Zurich, Switzerland
boehm@inf.ethz.ch

Abstract

The problem addressed in this paper is the execution of XML queries over a large collection of XML documents. This paper concentrates on how to develop the necessary infrastructure to effectively manipulate XML data and it proposes a data structure, named XML Schema Directory (XSD), as an access means to XML repositories. The aim of XSD is to accelerate query processing by quickly finding the relevant set of XML documents for a given query. This is obtained by considering only a small number of relative XML schemata and consequently a limiting number of XML documents, rather than the entire corpus of XML documents. XML schema similarity is introduced as a way to determine the relevance among XML documents, which belong to the same knowledge category. The proposed algorithms for maintaining the XSD structure do not require reorganisation and they may be efficiently used in practice. An alternative advantage of XSD structure is that it may also be used as a method for facilitating browsing.

1 Introduction.

In the last few years, there has been an increasing interest in managing semi-structured data [6, 1, 7, 18]. Techniques for extracting structured information from semi-structured data have been for long the main theme of research endeavors [3, 2, 17, 4]. The recent emergence of the *eXtensible Markup Language* (XML) [9] as a standard for data representation and exchange on the World Wide Web has attracted the interest of many researchers who observed a resemblance between semi-structured models and XML. Literally, XML data sources (documents) may be viewed as entities whose structure is not fixed and regular and both the data described and the structure itself are blurred. It is expected that much of the data encoded in XML will be semi-structured and they will be irregular or incomplete and their structure will change rapidly and unpredictably.

As more and more information is either stored, exchanged in XML, or presented as XML through various interfaces, the ability to intelligently query XML data sources becomes increasingly important. Toward this objective, several XML query languages have been proposed [8]. The realization of a query system is mainly accomplished through a query engine, which accepts XML queries, executes them on a predefined set of XML sources and finally it returns the result set. Querying intelligently XML sources requires a query language that supports many and different forms of processing. For instance, it is desirable to have a single XML query language that allows view definitions, query execution, item identification through Xpointer and XSL pattern evaluation. Although, supporting different domains is important, the languages that provide all the kinds of sophistication are expected not to be trivial since the relationships found in an XML document can be fairly difficult.

The main role of an XML query language is to allow the formulation of queries and determine the result set of the XML elements that should be returned. Although, all of the proposed query languages address the problem of query formulation, they assume that the input to such a query is a set of known documents or nodes within multiple documents. In other words, to execute an XML query, the query engine should be supplied with (1) the query string and (2) the URL data sources on which the query will perform.

In large corpus, which may consist of tens of thousands of XML documents with diverse schemata, the input documents may be not known in advance. In this case executing the XML query over the whole corpus may slow down the query answering process. It is then obvious that if the input documents are not known in advance, an indexing structure is required to alleviate the answering process by restricting the search space to a few documents which actually contains the desired data. Then these documents may be used as the input to the XML query.

Limiting the search space of a query means finding the most relevant XML documents to the query. This implies dividing the initial search space of the query into two disjoint sets; one containing those documents which are relevant to the query and the other containing those

* This research was done while the first author was visiting the Federal Institute of Technology ETH-Zurich from Jun. '99 to Feb. '00 and it was partially supported by the Swiss Federal Foundation of Technology and by the fellowship program of the European Research Consortium for Informatics and Mathematics (ERCIM).

documents, which are not. Therefore, the purpose of an XML indexing structure would be to partition the initial set of documents into relevant and not relevant and to supply the XML query engine with those XML documents that are relevant to the query.

This paper proposes a data structure, named *XML Schema Directory* (XSD), whose purpose is to quickly find the relevant set of XML documents to a given query. Moreover, XSD may be used to support content-based search on large collections of XML documents. The idea is to aggregate similar XML schemata into a merger schema. The merger schema represents a general class that contains all those XML documents of the original schemata. As long as a schema becomes part of the class, all the XML documents of the schema will be considered instances of the merger schema.

The rest of this paper is organized as follows: section 2 presents related work, section 3 discusses XML schemata and XML query evaluation, it also presents how a merger schema can be obtained. Similarity between XML schemata is discussed in section 4. Section 5 presents the *XML Schema Directory* (XSD) structure along with the algorithms for maintaining this structure. Section 6 presents how path queries may be evaluated by using the XSD structure and section 7 discusses some concluding remarks.

2 Related work

A collection of XML documents can be seen as a collection of objects. Finding relevant objects in a given set has been for many years the main research effort of information retrieval community [10, 5, 22, 21, 13, 11, 14].

In a traditional text information retrieval system, each document is segmented into significant terms (words) and a structure (known as inverted index) is generated that indicates what term occurred in what document as well as term frequency, term weight and possibly position data. A user query, in such a system, consists of a set of terms and it may be literally viewed as a document. The information retrieval system retrieves those documents that are considered close to the query. Certain similarity or dissimilarity (distance) measures have been invented to estimate proximity between a query and a document [20]. Although, numerous techniques exist to identify relevant documents [11], their effectiveness in terms of *precision* and *recall* is not adequate and consequently they are not appropriate for use in an XML repository. The proposed XSD structure guarantees 100% accuracy. This means that no non-relevant XML document will be selected (100% precision) and all of the relevant documents will be selected (100% recall).

Collections of semi-structured sources have been proposed as the basis for improving query handling and indexing in [4]. However, this approach is directed to classifying semi-structured sources by using a given set of classes, which are represented by *structural expressions*. That is, a predefined set of classes should be supplied before classifying the semi-structured objects. In practice, it is difficult and sometimes unlikely to have the definition of such generic classes that capture abstract types whose realization could be found in semi-structured sources and therefore it is difficult to specify in advance class structural expressions. Constructing collections of semi-structured objects is also presented in [17] through approximate typing. Approximate typing assumes that each single object is of a unique type and then elimination of types is accomplished by checking for equivalence among the initial types. In [4, 17] the resulting organizations are *approximate* types that are aimed to describe semi-structured objects.

XSD approach is aimed to be used for organizing semi-structured schemata (not semi-structured objects) in a hierarchical way and it may be viewed as a meta-schema organization. XSD approach is based on clustering XML schemata rather than on classifying semi-structured sources such as XML documents and therefore it does not require the preexistence of generic classes. It generates meta-classes (merger schemata) in a dynamic way from the basic semi-structured schemata. Semi-structured sources of a basic schema become members of the meta-classes. XSD serves two purposes: (1) it is an indexing structure that XML queries may use to find the most relevant XML documents in a large XML repository, (2) it is an organization for XML schemata which allows similar schemata to be placed in the same cluster. A cluster (or merger schema) may be viewed as a generic schema that encompasses many basic specialized schemata. XSD is a meta-schema organization, which organizes several XML schemata into a tree-like structure. It is different from that in [2, 15, 17, 4] at the level of organizing information; it is a construction that classifies XML schemata. However, the methods in [2, 15, 17, 4] may be used to extract basic schemata and then XSD may be applied to cluster these schemata. Therefore, under this perspective, XSD may act as a complement to the proposed techniques aiming to optimize queries that are targeted on large XML collections.

3 XML schema and query evaluation

An XML schema identifies the structure of the XML documents. XML 1.0 [9] supplies a mechanism, the Document Type Definition (DTD) for defining XML schemata. A DTD specifies what elements may occur and how the elements may nest in an XML document that conforms to the DTD. It serves two purposes: (1) it describes the characteristics of XML elements and (2) it declares constraints on the use of mark-up. When a DTD is not supplied the XML documents are self-describing and they may be viewed as semi-structured sources. That is, the XML document is combined from data whose

structure is not regular and its schema is contained within the data.

An XML document may be interpreted *literally* or *semantically* [12]. In the semantic mode, the XML document is represented as a graph that includes semantic relationships between XML elements. This is supported by the current XML version by assigning special meaning to some attributes. Attributes like ID and IDREF can be used to define relationships between XML elements. In the literal mode, an XML document is represented as a tree. There are no attributes with semantic interpretations. In this paper, XML documents are viewed *literally* and all kinds of attributes are visible as textual strings. It is also assumed well-formed XML, which place no restrictions on tags, attribute names or nesting patterns.

When DTDs are not available a dynamic schema such as a DataGuide [15] may be used to obtain a basic schema out of an XML document. DataGuides are dynamic schemata generated from semi-structured data sources describing every unique label path of the source once. In general, an XML basic schema is a tree-like structure, whose nodes are the labels (element names) found in an XML document and an edge from label *a* to *b* represents a parent/child relationship between *a* and *b*. A label is identified from the label path, which starts from the root and terminates at the label. Each label path appears in the tree at most once.

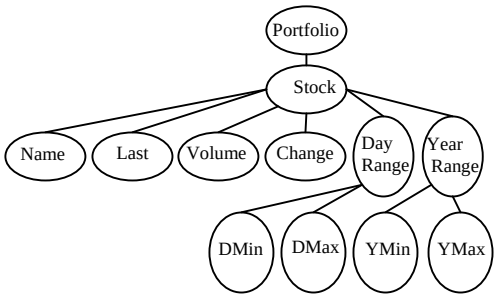
DTD

```
<!DOCTYPE Portfolio [
<!ELEMENT Portfolio (Stock*)>
<!ELEMENT
Stock(Name,Last,Volume,Change,
Day_Range,Year_Range)>
<!ATTLIST Stock
      Market CDATA #IMPLIED
      Ticker CDATA #IMPLIED>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Last (#PCDATA)>
<!ELEMENT Volume (#PCDATA)>
<!ELEMENT Change (#PCDATA)>
<!ELEMENT Day_Range
(DMin,DMax)>
<!ELEMENT DMin (#PCDATA)>
<!ELEMENT DMax (#PCDATA)>
<!ELEMENT Year_Range
(YMin,YMax)>
<!ELEMENT YMin (#PCDATA)>
<!ELEMENT YMax (#PCDATA)>
]>
```

XML document

```
<Portfolio>
  <Stock Market=" " Ticker="^DJI">
    <Name> DJ INDU AVERAGE    </Name>
    <Last>    10430.88    </Last>
    <Volume>      N/A    </Volume>
    <Change>     -2.00%    </Change>
    <Day_Range>  <DMin>    10356.96     </DMin>
                 <DMax>    10638.64     </DMax>
    </Day_Range>
    <Year_Range>  <YMin>    9099.04      </YMin>
                  <YMax>    11750.28     </YMax>
    </Year_Range>
  </Stock>
  <Stock Market="OSA" Ticker="^N225">
    <Name>     NIKKEI 225 INDEX    </Name>
    <Last>     19791.40    </Last>
    <Volume>     N/A    </Volume>
    <Change>     +0.98%    </Change>
    <Day_Range>     <DMin>     19518.08     </DMin>
                    <DMax>     19803.69     </DMax>
    </Day_Range>
    <Year_Range>    <YMin>     18068.10     </YMin>
                    <YMax>     20046.14     </YMax>
    </Year_Range>
  </Stock>
</Portfolio>
```

(a)



(b)

Figure 1: Example of XML Data (a) DTD and XML document (b) basic XML Schema

Figure 1(b) shows the basic XML schema obtained from either the DTD or the XML document in Figure 1(a). The basic XML schema may be seen as the type of an XML document. If the DTD is available, it is straightforward to derive the basic XML schema from the DTD. In case the DTD is not available the basic XML schema may be constructed from the XML document in a similar way as a DataGuide is formed. Using an XML schema, we are able to check whether a given label or label path exists. This is very important to query evaluation. An XML query is a path expression that can reach to arbitrary depths in the XML data. The result set of a path query on an XML document is the set of those elements that match the query. If for instance, a path expression is an ordered set of XML elements of the form $e_1/e_2/e_3$, it returns all elements e_3 for which there exist element e_1 containing e_2 and e_2 containing e_3. An XML query is relative to a document if it can match the document schema. If a path query matches one or more paths in the XML schema, then it is considered that it matches the schema. When an XML query matches a schema, all the XML documents derived from this schema are relative to the query.

3.1 Example 1: schema matching

Let us consider the following simple XML query expressed in XQL [19], which requests the stock names of those stocks with a day minimum grater than 10000.

```
/Portfolio/Stock/Name[/Portfolio/Stock//DMin > 10000]
```

Executing the above query on the XML document in Figure 1(a) with an XQL compliant query engine, it yields the following result:

```
<Name>  DJ INDU AVERAGE   </Name>
<Name>  NIKKEI 225 INDEX    </Name>
```

The XQL query above may graphically depicted as shown in Figure 2. Dashed edges show that the child node may be arbitrary found in any level bellow the parent node.

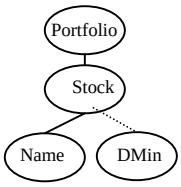


Figure 2: XML path query.

By matching a given query against each given schema, we can find out which of those schemata are relevant to the query and consequently consider all XML documents derived from these schemata to be the input to the query. This is a straightforward way to find relevant XML documents to a given query in a large XML repository, which encompasses numerous schemata. However, matching exhaustively the query against every single schema in the repository may slow down the process of

finding the input to the query. The next section introduces the concept of merging XML schemata into more generic schemata in order to solve the problem of exhaustive matching.

3.2 Merger schema

A merger schema is a generic XML schema, which combines two or more XML schemata. The introduction of the concept of merger schema aims at limiting the initial search space by merging basic XML schemata into more general ones, which may then be used as matching targets against XML queries. The merging process is accomplished as follows:

Let *A*, *B*, be two basic schemata, and *M* be the resulting merger schema. Let *root(A)* and *root(B)* be the root elements of schemata A and B respectively. Let E_A, E_B and E_M be the multi-sets containing the elements of the schemas *A* , *B* and *M* respectively. They are multi-sets rather than sets since elements with the same label may occur more than once in the schema. Let $C_M(e_m)$, $C_A(e_A)$ and $C_B(e_B)$ be the sets containing the child nodes of the element $e_m \in E_M$, $e_A \in E_A$, and $e_B \in E_B$ *respectively*. The merger schema is built recursively by using the following algorithm:

1. Set *M* equal to a temporary root element *TE* and then set *A* to be the child of *TE*
2. Call the recursive routine *Merge* in step 3 by *Merge(TE,root(B))*
3. *Merge*(e_m,e_b){
   //e_m, e_b are elements in schemata *M* and *B* respectively
   if e_b is not in $C_M(e_m)$ then add e_b in $C_M(e_m)$
   $e_m = e_b$ // e_m assigns the copy of e_b in *M*
   for each *x* in $C_B(e_b)$ do *Merge*(e_m, *x*) }

TE is just an assisting element to realise the merging. The resulting merger schema is the child of *TE* if both basic schemata *A* and *B* have the same root label. In case *A* and *B* have different root labels, *TE* is the parent label of both *root(A)* and *root(B)* labels. In the discussion that follows, we assume merging of basic schemata, which have common root label. The merger schema contains finally the union of elements of the basic schemata. To ease the process of schema separation, which is discussed in a following section, a reference number is introduced for each schema element, which indicates how many times the element has been referenced in the merger schema. For instance, if an element *E* is common to two basic schemata, then the merger schema will have a reference number for this element equal to two. The following example shows the merging of two basic schemata into a generic one.

3.3 Example 2: constructing a merger schema

Let us consider the XML schema in Figure 3, which is similar to that in Figure 1(b), although it has a different structure. The schema in Figure 3 may have been generated from the following XML document:

```
<Portfolio>
    <Stock Market="SAO" Ticker="^BVSP">
        <Name>BRSP BOVESPA IND</Name>
        <Last>18008.73</Last>
        <Volume>N/A</Volume>
        <Change>-0.57%</Change>
        <DMin>18008.73</DMin>
        <DMax>18297.38</DMax>
        <YMin>15349.78</YMin>
        <YMax>18885.84</YMax>
    </Stock>
</Portfolio>
```

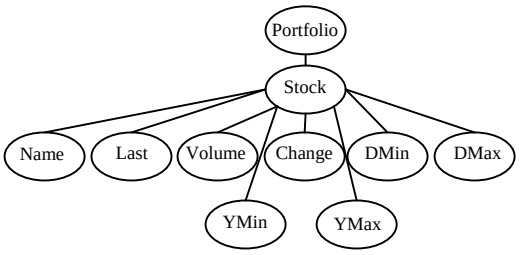


Figure 3: XML schema similar to that in Figure 1(b)

The merger schema will be the one shown in Figure 4. The merger schema keeps the structure of both basic schemata. It could be seen as the union of the two basic schemata in Figure 3, and Figure 1(b).

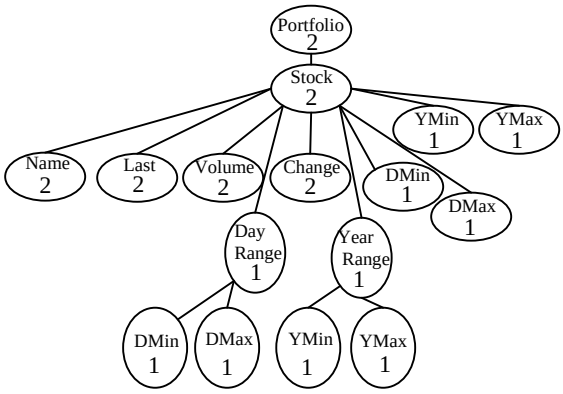


Figure 4: Merger XML schema of the basic schemata in Figure 1(b) and Figure 3

The reference number of each element of the merger schema is also shown on the nodes of the tree in Figure 4. Those nodes that have reference number equal to 1 occur only in one of the corresponding basic schemata. The advantage of the merger schema is that, it is sensitive to queries that are relevant to both basic schemata. Considering again the query in example 1, we see that the query matches both schemata and consequently it is relevant to all the documents derived from these schemata. The merger schema captures this and thus there is no need to go through exhausting matching to find out the relevant XML document to the query. What is actually needed in this case is to match the query only against the merger schema. In that way, we may substantially decrease the

search effort for relevant XML documents. Although using mergers to join XML schemata is promising, it may not yield the expected gain if we do not consider carefully what to merge. The basic schemata to be merged should be as *similar* as possible. Merging similar XML schemata offers an advantage since the resulting categories act as meta-classes that organise the underlying XML documents in separate knowledge categories, which may be used to efficiently evaluate XML queries over large collections of XML sources. Section 4 defines a measure for similarity between XML schemata and discusses how it can be estimated.

3.4 Schema separation

The Merge operation is used to join two relative XML schemata. The schema separation procedure does exactly the opposite. It separates from a merger schema a basic schema. Separation operation is the complement of Merge operation. That is, the sequence *C= Merge (A, B), Separate(C, B*) has no effect at all and it results in two basic schemata *A* and *B*. Let *A* be a merger schema and *B* be a basic schema and let *B* be contained in *A*. The operation *Separate (A, B)* is allowed only when *A* contains *B*, which means that *B* is a sub-tree of *A*. The *Separate (A, B)* operation is accomplished as follows:

1. For all those nodes in B that occur in A decrease the reference number by one.
2. If the reference number is zero, then remove the node from A

Referring to the example 2, Let *A* be the merger schema in Figure 4 and *B* be the schema in Figure 3, then *Separate (A, B)* will result in the schema shown in Figure 1(b).

4 Similarity between XML schemata

Similarity is defined in terms of proximity, which is based on the distance between XML schemata. The larger the distance between two XML schemata, the more dissimilar the XML schemata should be. An XML schema is actually a tree and therefore the distance between schemata should be based on tree differences. A mathematical model for obtaining distance measures between XML schemata is therefore discussed.

The distance between two XML schemata is based on the edit operations that should be performed to one of them in order to obtain the other. An edit operation on an XML schema may be an **insertion**, a **deletion** or a **substitution** of one node by another. Insertion is the complement of deletion.

Insertion of a node x into an XML schema as a child of node y may be accomplished so that the resulting XML schema contains x as a child of y with no children or takes as children some of the children of y. Let $y_1,\ldots, y_k$ be children of y, then for some $0\le i\le j\le k$, the children of y in the resulting tree (after the insertion of x) will be $y_1,\ldots y_i, x, y_j,\ldots, y_k$. If j=i+1, x has no children otherwise x has children $y_{i+1},\ldots y_j$,

Deletion of a node x from the XML schema is accomplished so that the father y of x takes all the children of x. Let $y_1,\ldots, y_k$ be the children of y and $x=y_i$, and let $x_1,\ldots, x_j$ be children of x, then the children of y in the resulting tree (after the deletion of x) will be $y_1,\ldots, y_{i-1}$ $x_1,\ldots, x_j, y_{i+1},\ldots, y_k$.

Substitution of a node x by a node y is accomplished so that the children of y in the resulting XML schema are the children of x in the original XML schema.

Any of the above elementary edit operations may be represented as a general substitution of the form $\alpha\to\beta$, which means *α is replaced by β*. In the case of the substitution operation above, α and β represent two distinct nodes. The deletion of a node α may be represented as $\alpha\to\varnothing$ (i.e. $\beta=\varnothing$), where $\varnothing$ is the null node. The insertion of the node β may be represented as $\varnothing\to\beta$ (i.e. $\alpha=\varnothing$).

An editing operation $\alpha\to\beta$ is associated with a cost $\mathbf{w}(\alpha\to\beta)$. This cost can be different for different nodes. For example editing a node, which is closer to the root might have higher cost than editing a leaf node or vise versa. In case that there is no distinction between nodes, a universal editing weight for all nodes may be used.

The distance between two schemata T_1 and T_2 is measured in terms of the number of editing operations required to change T_1 into T_2 taking into account the cost of each editing operation. The cost **w** to be a distance metric should satisfy the following metric axioms [16, 23].

i) $\mathbf{w}(\alpha\to\beta)\ge 0$ and $\mathbf{w}(\alpha\to\alpha)=0$
ii) $\mathbf{w}(\alpha\to\beta)=\mathbf{w}(\beta\to\alpha)$
iii) $\mathbf{w}(\alpha\to\gamma)\le\mathbf{w}(\alpha\to\beta)+\mathbf{w}(\beta\to\gamma)$

Let S_i be a sequence of editing operations $s_{i1}, s_{i2},\ldots,s_{ik_i}$ that changes the XML schema T_1 into T_2. The cost of performing all the operations in the editing sequence is then given by

$$W(S_i)=\sum_{j=1}^{k_i} \mathrm{w}(s_{ij})$$

The distance between the schemata T_1 and T_2 is then formally defined as

$$d(T_1,T_2)=\min\{W(S_i)\mid S_i \text{ is an editing operation sequence changing } T_1 \text{ into } T_2\}$$

Algorithms that estimate the above-defined distance between trees is discussed in [23], which also presents a dynamic algorithm that solves the minimum distance problem in sequential time.

5 XML Schema Directory (XSD)

XSD is a hierarchical (tree-like) clustering structure whose aggregation technique to cluster XML schemata is based on schema merging. The objects used for clustering are basic XML schemata. The motivation for constructing such a clustering structure is that XML schemata relevant to a path query tend to be more similar to each other than irrelevant XML schemata and they may be clustered together. XSD structure is mainly used to accelerate query processing by considering only a small number of relative XML schemata and consequently a limiting number of XML documents, rather than the entire corpus of XML documents. An alternative advantage of XSD structure is that it may also be suggested as a method for facilitating browsing.

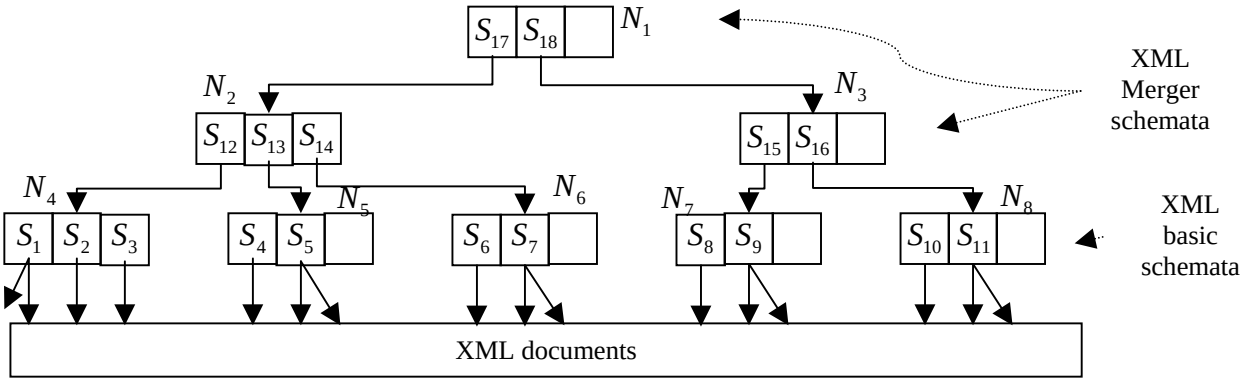
XSD is a data structure, which is designed to ease the process of finding relevant XML documents in a corpus and it is mainly used as an access means to XML repositories. Each non-leaf node entry in the directory contains a merger schema and each leaf directory entry point to the XML documents derived from the leaf schema. A leaf node entry E has the form $(S_{(E)}, X_{1(E)}, X_{2(E)}, \ldots, X_{n(E)})$, where $X_{1(E)}, X_{2(E)}, \ldots, X_{n(E)}$ are pointers to XML documents and $S_{(E)}$ is the schema whose instances are pointed to by $X_{1(E)}, X_{2(E)}, \ldots, X_{n(E)}$. Every leaf node may contain an arbitrary number of references (i.e. many XML documents may be derived from the same XML schema). A non-leaf node entry E has the form $(S_{(E)}, X_{(E)})$, where $S_{(E)}$ is a merger schema and $X_{(E)}$ is a pointer to a node containing n schemata with $m \le n \le M$, where M and m are the maximum and minimum numbers respectively of the schemata that can be accommodated in an XSD node. The relationship between m and M could be defined as $m \le M/2$. It is worth noting that while the leaf nodes may have an arbitrary number of pointers to XML documents, the non-leaf nodes have a limited number of pointers to child nodes. This is because the XSD structure is tuned to organise XML schemata, rather than XML sources. Limiting the number of child schemata between m and M is done for performance purposes. For instance, small values of M will create a deep XSD structure, whereas large values of M may create a wide XSD structure. In general, m and M may vary and different values may be used in a way that increases the performance.

Figure 5 shows the structure of a typical XSD organisation and illustrates the merger XML schemata as entries of non-leaf nodes and basic schemata as entries of leaf nodes. A basic XML schema is one that is obtained directly from a DTD or an XML document.

In Figure 5 nodes N_1, N_2 and N_3 contain merger schemata entries and the leaf nodes N_4, N_5, N_6, N_7 and N_8 contain XML basic schemata. For instance, the merger schema S_{16} in Figure 5 may be the XML schema in Figure 4 and the child schemata S_{10} and S_{11} may be the basic schemata in Figure 1(b) and Figure 3 respectively.

Two basic algorithms are proposed for maintaining the XSD structure. One for inserting new XML schemata and one for deleting XML schemata. The schemata allowed to be inserted or deleted are basic XML schemata.

Figure 5: XSD Tree.



Insertion: inserts a new basic schema S into an XSD structure whose root is T: *Insert(T,S)*.

1. If T is not a leaf, check the similarity between the new schema and each entry in T by measuring the distance between the new schema and a schema entry of T. The greater the distance is the less similar the new schema is. The distance is measured as the number of editing operations that should be performed to transform one schema to the other.
2. Let schema W be the most similar entry of T to the new schema, then invoke *Insert* $(X_{(w)}, S)$, where $X_{(W)}$ is the pointer to the child of W
3. If T is a leaf, then, check to see if there is room where the new schema can be accommodated. If so, merge the new schema with each single schema entry from the leaf up to the root of the XSD and update all the non-leaf nodes by merging each one with the new schema. That is, the new schema is merged first with the parent entry of the leaf, then with the grand father and so on until the root of XSD.

4. If the number of entries of a node exceeds M (the maximum number of lower level schemata that a merger schema can have), then the entries of this node are split into two disjoin groups. The splitting may be accomplished by utilising a partitioning algorithm (agglomerative or divisive), that breaks the set of entries into two groups so that entries within the group are close (similar) to each other and entries of different groups are dissimilar. Splits may propagate up to the root as far as the splitting in lower level makes nodes in higher levels to overflow. Splits are handled in a similar way as in B-tree. This splitting operation is used to keep the XSD structure balanced and it is done for performance purposes.

Deletion: deletes an existing basic schema *S* from the XSD structure whose root is *T*: *Delete (T, S)*.

For all the ancestors *X* of the leaf basic schema invoke *Separate (X, S)*. That is, the leaf schema contribution to merger schemata all the way up to the root is removed. This operation may cause underflow on the parent node of the leaf schema when the number of entries becomes less than *m* (the smallest number of entries that can be accommodated by a merger schema). In that case a new *Merge* operation between the parent node entries may be performed. In order to make sure that the number of children of any merger schema is between *m* and *M*, merging due to deletion may propagate up to the root.

There are also two other operations, which are used in the XSD structure. These operations are used to *disassociate* and *associate* XML documents with an existing leaf schema. However, these operations are trivial and can be realised by deleting or adding respectively a new reference to the corresponding leaf node of which, the XML document is an instance.

6 Query evaluation through XSD

An XML query is evaluated by searching for XML schemata that match the query. The matching of a query against an XML schema is accomplished as it has been already described in example 1. The search algorithm for finding relative XML schemata in the repository and eventually parsing all the XML documents that are derived from these schemata is called *Match* and it is described as follows:

Match algorithm objective: Given a path query *Q* and a XSD structure, whose root is *T*, find all XML documents that match the query. *Match (T, Q)* will return all XML documents in XSD structure, which are relevant (match) to the query *Q*. This is accomplished according to the following steps.

1. If *T* is not a leaf, check each entry of *T*. For each entry *C* of *T* that matches the query, invoke $Match(X_{(C)}, Q)$, where $X_{(C)}$ is the pointer from C to a child node.
2. If *T* is a leaf, check if *Q* matches any entry of *T* and if so, then return the pointers (or reference IDs) to the XML documents that are instances of those entries of T that match the query Q.

The *Match* algorithm starts from the root and move towards the leafs by matching the query with the intermediate merger schemata. All the nodes from the root to the desired leaf schemata (those that match the query) are visited and need to be searched. In that way, the exhaustive matching is avoided and eventually only relevant schemata to the query are searched. This limits the number of searching steps and on the other hand it returns exactly those XML documents that are relevant to the query. This has as a consequence, no unrelated document to be returned (100% precision) and all of the related documents to be considered in the result set (100% recall). The effectiveness obtained by using XSD structure is ideal in terms of precision and recall. Moreover, the searching space is limited and eventually the relevant XML documents are retrieved quickly avoiding exhaustive searching.

7 Conclusions

The advantages of the XSD scheme are as follows:

1. There is no need to parse non-relevant documents. This results in executing XML queries faster since the search space is limited only to the relevant XML documents.
2. The accuracy in answering XML queries is high. The result set is exactly the same as the one that would be obtained through exhaustive searching if the proposed XSD structure were not used.
3. Maintaining the XSD structure is not difficult since new XML schemata may be added or old ones deleted without performing time-consuming operations that requires reorganization of the whole XSD structure.

The proposed method for organizing XML schemata to accelerate query processing is indeed promising. However, further work is needed towards constructing a XSD based management system for making performance measurements. Such a system is under way and future work is planed based on measuring the behavior of the system in practice by considering a real XML document collection.

References

[1] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, Janet L. Wiener, "The Lorel query language for semistructured data", *International Journal on Digital Libraries*, Vol. 1 Issue 1 (1997), pp 68-88

[2] Brad Adelberg, "NoDoSE-a tool for semi-automatically extracting structured and semistructured data from text documents", In *Proceedings of ACM SIGMOD international conference on Management of data*, June 1 - 4, 1998, Seattle, WA USA , Pages 283 - 294

[3] Paolo Atzeni, Giansalvatore Mecca, Paolo Merialdo, "To Weave the Web", In *Proceedings of 23rd International Conference on Very Large Data Bases VLDB'97*, August 25-29, 1997, Athens, Greece, pages 206-215,

[4] Elisa Bertino, Giovana Guerrini, Isabella Merlo and Marco Mesiti, "An Approach to classify Semi-Structured Objects", In *Proceedings of the 13th European Conference on Object Oriented Programming (ECOOP'99)*, Lisbon, Portugal, June 1999, Rachid Guerraoui (Ed.), LNCS 1628, pp. 416-440, 1999

[5] Rodrigo A. Botafogo, "Cluster analysis for hypertext systems", In *Proceedings of the sixteenth annual international ACM SIGIR conference on Research and Development in Information Retrieval*, June 27 - July 1, 1993, Pittsburgh, PA USA, Pages 116 - 125

[6] Peter Bunemanm, "Semistructured data", In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (PODS97)*, May 11 - 15, 1997, Tucson, AZ USA, Pages 117-121

[7] Peter Buneman, Serge Abiteboul and Dan Suciu, "Data on the Web:From Relations to Semistructured Data & XML", Oct. 1999, Morgan Kaufmann

[8] Angela Bonifati, Stefano Ceri, "Comparative Analysis of Five XML Query Languages", *ACM Sigmod Record, March 2000*

[9] Tim Bray, Jean Paoli and C. M. Sperberg-McQueen "Extensible Markup Language (XML) 1.0", *W3C Recommendation* 10-February-1998, http://www.w3.org/TR/1998/REC-xml-19980210

[10] Douglass R. Cutting, David R. Karger, Jan O. Pedersen and John W. Tukey, "Scatter/Gather: a cluster-based approach to browsing large document collections", In *Proceedings of the Fifteenth Annual International ACM SIGIR conference on Research and development in information retrieval*, June 21 - 24, 1992, Copenhagen Denmark, Pages 318 - 329

[11] David A. Grossman and Ophir Frieder, "Information Retrieval:Algorithms & Heuristics", 08/1998, Kluwer Academic.

[12] Roy Goldman, Jason McHugh, Jennifer Widom, "From Semistructured Data to XML: Migrating the Lore Data Model and Query Language", In *Proceedings of the ACM International Workshop on the Web and Databases (WebDB'99)*, Philadelphia, Pennsylvania, USA, June 3/4, 1999

[13] Sudipto Guha, Rajeev Rastogi and Kyuseok Shim, "CURE: an efficient clustering algorithm for large databases", In *Proceedings of ACM SIGMOD international conference on Management of datam*, June 1 - 4, 1998, Seattle, WA USA, Pages 73 - 84

[14] S. Guha, R. Rastogi, and K. Shim, "ROCK: A Robust Clustering Algorithm for Categorical Attributes", In *Proceedings of IEEE 15th International Conference on Data Engineering*, 23 - 26 March, 1999 Sydney, Australia

[15] Roy Goldman and Jennifer Widom, "DataGuides: Enabling Query Formulation and Optimazation in Semistructured Databases", In *Proceedings of the 23rd VLDB Conference*, pp. 436-445, Athens, Greece, August 25-29, 1997

[16] Josheph B. Kruskal, "An Overview of sequence Comparison", In *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*, edited by David Sankoff and Joseph B. Kruskal, pp. 1-44, Addison Wesley, 1983

[17] Svetlozar Nestorov, Serge Abiteboul and Rajeev Motwani "Extracting schema from semistructured data", In *Proceedings of the ACM SIGMOD international conference on Management of data* June 1 - 4, 1998, Seattle, WA USA, Pages 295 - 306

[18] Y. Papakonstantinou and P. Velikhov, "Enhancing Semistructured Data Mediators with Document Type Definitions", In *Proceedings IEEE 15th International Conference on Data Engineering, (ICDE'99)* 23 - 26 March, 1999, Sydney, Australia.

[19] Jonathan Robie, Joe Lapp & David Schach, "XML Query Language (XQL)",In *Proceedings of The W3C Query Languages Workshop* (QL'98), Boston, Massachusetts, Dec 3-4, 1998,http://www.w3.org/TandS/QL/QL98/pp/xql.html

[20] Jason Tsong-Li Wang, Xiong Wang, King-Ip Lin, Dennis Shasha, Bruce A. Shapiro and Kaizhong Zhang, "Evaluating a class of distance-mapping algorithms for data mining and clustering", In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining* August 15 - 18, 1999, San Diego, CA USA Pages 307 - 311

[21] Oren Zamir and Oren Etzioni, "Web document clustering: a feasibility demonstration", In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, August 24 - 28, 1998, Melbourne Australia , Pages 46 - 54

[22] Tian Zhang, Raghu Ramakrishnan and Miron Livny, "BIRCH: an efficient data clustering method for very large databases", In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data* June 3 - 6, 1996, Montreal Canada, Pages 103 - 114

[23] Kaizhong Zhang and Dennis Shasha, "Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems", *SIAM Journal on Computing* Vol. 18, No. 6, pp. 1245-1262, December 1989.