Desirable Properties for XML Update Mechanisms

Martin F. O'Connor Mark Roantree

Interoperable Systems Group, School of Computing,
Dublin City University, Glasnevin, Dublin 9, Ireland
{moconnor,mark}@computing.dcu.ie

ABSTRACT

The adoption of XML as the *default* data interchange format and the standardisation of the XPath and XQuery languages has resulted in significant research in the development and implementation of XML databases capable of processing queries efficiently. The ever-increasing deployment of XML in industry and the real-world requirement to support efficient updates to XML documents has more recently prompted research in dynamic XML labelling schemes. In this paper, we provide an overview of the recent research in dynamic XML labelling schemes. Our motivation is to define a set of properties that represent a more holistic dynamic labelling scheme and present our findings through an evaluation matrix for most of the existing schemes that provide update functionality.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Query processing—*XML*

General Terms

Algorithms, Theory, Languages.

Keywords

Semi-structured data, labelling scheme, XML update.

1. INTRODUCTION

XML [24] has been adopted as the *defacto* standard for data exchange on the World Wide Web and increasingly so in industry as the interchange format for enterprise applications and web services. The key ingredient to the successful adoption of XML is its expressive and extensible nature. XML is a versatile markup language capable of labelling the information content of diverse data sources including structured and semi-structured documents. Semi-structured data is such that the structure is not necessarily known in advance, it is often self-describing (as is the case with XML) and consists of irregular and non-uniform organisation.

Updates in XML (EDBT Workshop Proceedings), March 22, 2010, Lausanne, Switzerland.

Many labelling schemes have been proposed for XML data with an overview provided in [20]. Labelling schemes can be broadly categorised under three headings: containment schemes, prefix schemes and prime number schemes. Most of the research to date has focused on the construction of labelling schemes capable of efficient query processing and query optimisation over static XML data. However, as the volume of XML data increases and the adoption of XML repositories in mainstream industry becomes more widespread, there is a requirement for a labelling scheme that can support efficient updates.

In this paper, we focus on dynamic labelling schemes supporting XML updates. There are many types of XML update operations possible based on the underlying update operations on a tree structure: leaf node, internal node and subtrees updates. Furthermore, the elements in an XML document are intrinsically ordered and this order must be maintained in the presence of updates in order to comply with the semantic requirements of W3C XPath and XQuery languages.

Each of the dynamic labelling schemes proposed to date have differing characteristics offering distinct advantages and limitations with respect to one another, and in terms of queries supported, their update costs and encoding size. Most of the evaluations provided by researchers has focused on comparative performance analysis. To the best of our knowledge, no comprehensive analysis of existing dynamic labelling schemes has been performed with a view to identifying the key characteristics of a robust dynamic labelling scheme, the number and specification of the core properties they should encode and the essential requirements they should satisfy. The output from such an analysis will be a template of properties that would form a key component in the evaluation of any new dynamic labelling scheme in conjunction with a comparative performance evaluation.

1.1 Contribution

In this paper, we provide a survey and review of the principle dynamic node labelling schemes proposed to date. From our analysis we define a template of properties that are representative of the characteristics of a *good* dynamic labelling scheme. We will use this template to assess and critique each of the labelling schemes in our review, detailing precisely the characteristics supported and to what degree. This exhaustive analysis will play a valuable role in providing input and metrics into the development of a new encoding scheme for

facilitating XML updates.

This paper is structured as follows: in §2, we discuss the terminology involved in indexing and update mechanisms for XML databases; in §3, our main body of work is in the examination of present dynamic labelling schemes with a view to identifying desirable properties; in §4, we describe the main issue or point of failure for many of these schemes; in §5, we re-examine existing work in light of our template of desirable properties; and finally in §6, we offer conclusions.

2. BACKGROUND TERMINOLOGY

In the context of XML, the terms *labelling scheme* and *encoding scheme* are often used interchangeably. To begin, it is important to clarify what these terms mean and the relationship between them. An overview of the core terminology will help to set the scope and define the context of our work.

2.1 XML Tree Basics

Both labelling schemes and encoding schemes are defined over a *tree* representation of XML data and not the textual XML document. This is a direct consequence of the XPath data model defining its operations in terms of a tree representation of an XML document. Thus, we commence by outlining the relationship between an XML document and a *tree* and explain why XPath was defined to operate on the tree representation of XML.

The basic structure underlying an XML document is an ordered rooted tree. The leaves in the tree correspond to the data values (text) and the internal nodes correspond to XML elements. The tree is an abstract datatype. There is no defined API and no defined data representation, only a conceptual model that defines the objects (nodes) in a tree, their properties and their relationships [10]. The decision to define XPath operations in terms of a tree representation of an XML document and not the textual XML document was motivated by the fact that an XML document may be constructed from many data sources, such as a view over relational data or an XML fragment constructed by an application in memory. In such cases, it is not required or even desirable to put an XML document through an XML parser each time it is to be processed. In fact, it would put an unnecessary overhead on the query processing costs to convert these into textual documents and parse them before XPath evaluation could proceed. Furthermore, trees do not have values in the XPath data model precisely because they are an abstract datatype. Thus, in order to pass a tree as an argument to an XPath function, one passes the root node of the tree.

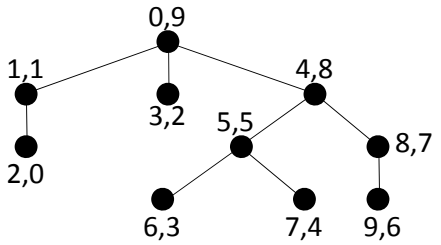
2.2 Labelling Schemes

An XPath processor must be capable of distinguishing between nodes to evaluate an expression and this motivates usage of a labelling scheme. The purpose of a labelling scheme is to provide unique labels for each node in the XML tree. Figure 1(a) presents a sample XML document in textual format and figure 1(b) illustrates the XML tree representation of the sample document labelled using a preorder/postorder[1] labelling scheme.

[1]Preorder and postorder tree traversals are introduced in §3.1.1

```
<book>
  <title genre="Fantasy"> Wayfarer </title>
  <author> Matthew Dickens </author>
  <publisher>
    <editor>
      <name> Destiny Image </name>
      <address> USA </address>
    </editor>
    <edition year="2004"> 1.0 </edition>
  </publisher>
</book>
```

(a) A sample XML file in textual format



(b) A Preorder/Postorder labelled tree representation of the sample XML file

Figure 1: Two representations of a Sample XML file

Definition 1. A labelling scheme for XML assigns unique identifiers (labels) to each node in the XML tree and these labels facilitate node ordering.

Labels may be numeric, alphabetic or alphanumeric. The node labels must be unique because XPath requires all its operators to eliminate duplicate nodes from their result sequences based on node identity. Furthermore, as the resulting node sequence must be returned in document order, an XML labelling scheme should also facilitate the identification of node order.

Although a labelling scheme's primary function is to provide ids, often they are constructed to also capture some of the structural semantics of the XML tree itself. For example, prefix-based labelling schemes[2] incorporate the labels of the parent and ancestors in the label of the node itself and thus permit parent-child and ancestor-descendant evaluations. Preorder/postorder labelling schemes permit ancestor-descendant evaluations but not parent-child evaluations. We shall examine the properties of the various labelling schemes in the next section but it is sufficient to say at this point that labelling schemes incorporate *some* of the structural semantics of an XML tree. The precise details of the structural semantics captured are determined by the properties of the labelling scheme employed.

2.3 Encoding Schemes

However, no labelling scheme captures the node type, the element or attribute names nor the content of the XML data and thus, lack the sufficient information required to permit full XPath query evaluations. These requirements

[2]Prefix-based labelling schemes are introduced in §3.1.2

Pre	Post	Node Type	Parent (Pre)	Name	Value
0	9	Element		book	
1	1	Element	0	title	Wayfarer
2	0	Attribute	1	genre	Fantasy
3	2	Element	0	author	Matthew Dickens
4	8	Element	0	publisher	
5	5	Element	4	editor	
6	3	Element	5	name	Destiny Image
7	4	Element	5	address	USA
8	7	Element	4	edition	1.0
9	6	Attribute	8	year	2004

Figure 2: An XML encoding of the sample XML File

motivate the need for an XML encoding scheme. An XML encoding scheme is constructed upon a labelling scheme and augments it with the information necessary to perform full XPath query evaluations. A sample encoding scheme is illustrated in figure 2. The XML encoding scheme should also permit the full reconstruction of the textual XML document that corresponds to the XML tree representation.

Just as there are many labelling schemes for XML, there may be several different encoding schemes proposed for each labelling scheme. The exact contents of an XML encoding scheme is often determined by the type and purpose of the underlying data. If the XML data is rarely updated, an encoding scheme may bear the cost of storing extra information in order to facilitate faster query performance, at the expense of slower update performance. A more formal definition of an XML encoding scheme now follows:

Definition 2. An XML encoding scheme codifies the structure of the node sequence in the XML tree and the properties and content of each node.

2.4 Summary

Many of the labelling schemes proposed to date assume that underlying XML data is static. If updates to XML documents are supported, efficiently updating node labels and maintaining document order in the presence of such updates, offers significant challenges. In this paper, we focus specifically on *dynamic labelling schemes* for XML. The principle challenges in the development of an efficient and dynamic encoding scheme for XML lie primarily in the specification of an efficient and dynamic labelling scheme upon which the encoding scheme is constructed. The properties of the labelling scheme and the number of structural relationships it encapsulates necessarily determine the quantity and properties of the supplementary information required by the XML encoding scheme.

3. RESEARCH INTO XML UPDATES

As dynamic labelling schemes are most suited to XML updates, in this section, we provide a detailed analysis of differing forms of dynamic schemes. We begin with an overview of the characteristics of dynamic labelling schemes, before providing our analysis of two of the major dynamic types: containment schemes and prefix schemes.

3.1 Dynamic Labelling Schemes

XML update operations can be broadly classified as either *structural updates* or *content updates*. Content updates refer to changes in the underlying data values such as element content, comments and processing instructions and the names of elements and attributes. Structural updates refer to the insertion or deletion of nodes in the XML tree. We omit from this survey, the dynamic labelling schemes [21, 4, 26] that do not support the maintenance of document order under updates.

Document order is defined for all nodes in the XML tree and corresponds to the order in which the first character of each element occurs in the XML document. There are three generic approaches to capturing document order in a labelling scheme [22]: Global order, Local order and Hybrid order (a hybrid of the local and global order). With global order, a node is assigned an identifier that represents the element's absolute position in the document. With local order, an identifier representing the position of the node relative to its siblings is assigned. A global order approach tends to be efficient for query processing but unsuitable for a dynamic labelling scheme because insertions modify the positional values of all nodes after the inserted node. A local order approach is more update friendly in that only the following siblings (and their descendants) of an inserted node must be modified. However, local order presents difficulties in the evaluation of the following and preceding axis as no global order information is available. The hybrid approach attempts to strike a balance between the strengths and weaknesses of global and local order. Most of the dynamic label schemes proposed to date follow the hybrid approach.

Many dynamic labelling schemes for XML have been proposed and these can be broadly categorised under two headings: containment schemes and prefix schemes. Our detailed analysis of these schemas can now commence.

3.1.1 Containment Schemes

Containment based labelling schemes (otherwise known as Interval based labelling schemes or Region encoded labelling schemes) exploit the properties of tree traversal to maintain document order and to determine various structural relationships between nodes. Thus, we begin by introducing tree traversal and then provide an overview of the containment based labelling schemes proposed to date.

Tree traversal is the process of visiting each node in a tree data structure. Tree traversal provides for sequential processing of each node in what is, by nature, a non-sequential data structure (e.g., semi-structured data). Such traversals are characterised by the order in which the nodes are visited. In preorder traversal, each node u is visited and assigned its

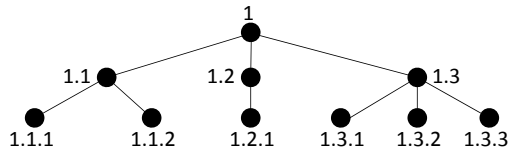


Figure 3: DeweyID Labelled XML tree.

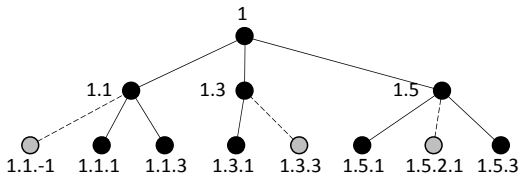


Figure 4: ORDPATH Labelled XML tree.

preorder traversal rank *pre(u)* before its children are recursively traversed from left to right. It is worth observing at this point that the act of parsing an XML document in document order, that is, processing each line from left to right and from top to bottom, corresponds to a preorder traversal of the XML document tree. Thus, preorder traversal maintains the document node order of an XML document. In postorder traversal, a node u is assigned its postorder traversal rank *post(u)* after all its children have been traversed from left to right.

The concept of containment based or interval based labelling schemes for tree structured data was first proposed in [6]. The author made use of tree traversals to determine the ancestor-descendant relationships between any given pair of nodes. He proposed that node u is an ancestor of node ν in a tree τ iff u occurs before ν in the preorder traversal of τ and after ν in the postorder traversal. To be specific, in [9], the author shows that the evaluation of a location step on a major XPath axis (ancestor, descendant, following, preceding) amounts to a rectangular region query in the pre/post labelled plane. Most containment based schemes adopt a global ordering approach to document order.

Several variations of the containment based labelling scheme have been proposed [9, 31, 1, 30] that record the begin position and end position of each element in the XML document and optionally their level. The begin and end positions may be generated by performing a depth-first traversal of the tree and sequentially assigning a number at each visit. Each non-leaf node will be traversed twice, once before visiting all its descendants and once after. Leaf nodes will always contain content values and not structural information and are thus, considered by the XML encoding scheme and not the labelling scheme. The level of an element is its nesting depth in the document. For any pair of nodes, u and ν, u is an ancestor of ν iff u.begin $<$ ν.begin and ν.end $<$ u.end. Essentially, this states that the interval of u contains the interval ν. By incorporating the level information in the label identifiers, this labelling scheme permits the evaluation of the parent-child axis. Node u is a parent of node ν iff u is an ancestor of ν and u.level = ν.level - 1. Node u is a sibling of node ν if they share the same parent and are at the same level.

In [23], a hybrid ordering approach is adopted whereby sectors are used instead of intervals and mathematical formulae are presented to determine ancestor-descendant and document-order relationships between label pairs. The limitation with all of the above labelling schemes is that a significant number of labels may need to be recomputed when a node is inserted. Several extensions were proposed [17, 9, 11] which permit gaps in the labelling schemes to facilitate future insertions gracefully. However, these solutions serve to increase the label size through the sparse allocation of labels and only postpone the relabelling process until the interval gaps have been consumed by the update process. In [2], they propose the use of real (floating point) numbers for label identifiers instead of integers to facilitate an arbitrary number of insertions between two labels. However, computers represent floating point numbers with a fixed number of bits and thus in practice the solution is similar to an integer representation of labels with sparse allocation and consequently suffers from the same limitations. In other words, none of these solutions are scalable.

3.1.2 Prefix Schemes

In a prefix labelling scheme, the label of a node in the XML tree consists of the parent's label concatenated with a delimiter and a positional identifier of the node itself. The positional identifier indicates the position of the node relative to its siblings and incorporates the *Local Order* approach for document order. For a pair of nodes u and ν, u is an ancestor of ν iff label(u) is a prefix of label(ν). The node's position combined with its ancestors' labels provides a path vector that uniquely identifies the absolute position of the node in the document - capturing *Global order*. Thus, a prefix labelling scheme follows the *Hybrid order* approach to document order.

DeweyID [22] is a prefix labelling scheme adapted from the Dewey Decimal Classification system [5] for the organisation of library collections. Figure 3 illustrates a DeweyID labelled XML tree. DeweyID is a naive prefix scheme whereby the positional identifier of the n^{th} child of a node is assigned the integer n and this is concatenated to the parent's label and a delimiter. The insertion of new nodes requires the relabelling of any follow-sibling nodes (and their descendants) which can have significant costs.

The OrdPath labelling scheme [18] is conceptually similar to the DeweyID and permits the insertion of new nodes in arbitrary positions in the XML tree without the need to relabel existing nodes. The XML tree is initially traversed in document order and nodes are labelled with positive, odd integers only (beginning with 1). Even-numbered and negative integer component values are reserved for later node insertion into an existing tree. The ORDPATH labels are not stored as dotted-decimal strings but rather in compressed binary representation to enable efficient XPath evaluations. Figure 4 illustrates an ORDPATH labelled XML tree; the grey nodes indicating newly inserted nodes. A new node inserted to the right of all existing child nodes is labelled by adding two to the positional identifier of the right-most child node (e.g.: node 1.3.3 in figure 4). In a similar fashion a new node inserted to the left of all existing child nodes is labelled by adding -2 to the positional identifier of the

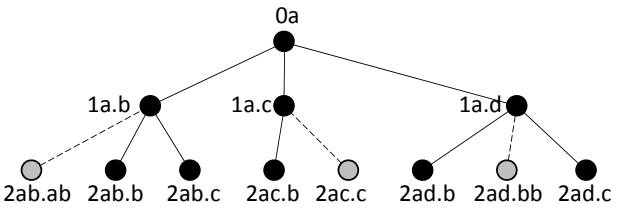


Figure 5: LSDX Labelled XML tree.

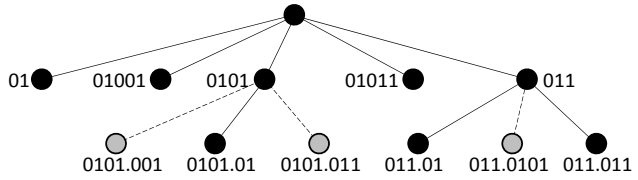


Figure 6: ImprovedBinary Labelled XML tree.

left-most child node (e.g.: node 1.1.-1 in figure 4). Lastly, a new node is inserted between two consecutive nodes using a *careting-in* technique whereby the positional identifier of the new node is assigned the even-number that sits between the two odd positional identifiers of its neighbour siblings, and then concatenating a new component consisting of an odd number (e.g.: node 1.5.2.1 in figure 4). Subtree insertions may be serialised as a sequence of nodes and inserted individually. The level or depth of each node in the tree may be determined by counting the number of odd component values in the label. ORDPATH labels permit the evaluation of ancestor-descendant, parent-child and sibling-order relationships. The limitations of the ORDPATH labelling scheme are a result of the variable length labelling scheme employed in conjunction with the waste of half of the total numbers by virtue of labels ending in odd numbers. This can result in increases storage costs in the case of frequent updates as well as expensive comparative label evaluations between sibling nodes of varying length. Furthermore, the ORDPATH labelling scheme cannot completely avoid the relabelling of existing nodes due to the overflow problem (more details are provided in the next section).

A labelling scheme quite similar to ORDPATH is presented in [3] called the DLN (Dynamic Level Numbering) scheme which adopts a fixed bit-length for component values and supports arbitrary insertions through the addition of *suffix* values between any two consecutive positional identifiers. However, under frequent updates, the fixed label size may overflow and thus, this scheme will succumb to the same limitations as the DeweyID scheme using sparse allocation of labels.

In [4], two prefix-based labelling schemes are proposed which assign bit codes as the positional identifiers in node labels. The first approach has a label growth rate of one-bit such that the positional identifier of the first child of node u is 0, of the second child is 10, of the third child is 110 and of the n^{th} child is (n-1) ones with a 0 concatenated at the end. The second approach has a double-bit label growth rate. As a result, both approaches tend to have significant label sizes and consequently large storage costs and expensive comparative evaluation costs for even modest document sizes.

In [7], the LSDX labelling scheme is proposed which employs both integers and letters in the construction of a node's label. The root node of the tree is label *0a*, where the integer component *0* represents the level or depth of the node and the alphabetic component *a* represents the positional identifier. Figure 5 illustrates an LSDX labelled XML tree; the grey nodes indicating newly inserted nodes in an existing tree. During the initial XML tree construction, the first child of every node uses the letter *b* instead of *a* to permit future insertions before the first child. If the previously assigned positional identifier is *z*, then the next identifier will be *zb*. A new node inserted to the left of all existing child nodes is labelled by taking the existing leftmost child label and prefixing an *a* to its positional identifier (e.g.: node 2ab.ab in figure 5). A new node inserted to the right of all existing child nodes is labelled by taking the existing rightmost child label and lexicographically incrementing the last letter of its positional identifier (e.g.: node 2ac.c in figure 5). A new node is inserted between two existing nodes by lexicographically incrementing the positional identifier of the new node such that it is greater than its left neighbour and less than its right neighbour (e.g.: node 2ad.bb in figure 5). Essentially, the labelling scheme supports updates such that the labels of sibling nodes will always have their alphabetic components lexicographically ordered. LSDX permits the evaluation of ancestor-descendant, parent-child and sibling-based relationships. Furthermore, labels are not persistent and may be reassigned upon deletion.

The authors of LSDX acknowledge that the label size grows very quickly for nodes with hundreds of siblings and they propose an improved version of their labelling scheme in [8] called Compressed Dynamic Labelling Scheme or Com-D for short. The basic concept is to compress reoccurring letters within a label by prefixing the repetitive letter(s) with an integer indicating the number of repetitions. For example, the positional identifier *aaaaabcbcbcdddde* would be rewritten as *5a3(bc)4de*. In [12] a labelling scheme very similar to LSDX is proposed and differs only in the method to determine the positional identifier of a node. However, LSDX and the two labelling schemes derived from it do not always produce unique node labels for several corner-case update scenarios and therefore they are unsuitable for use as dynamic labelling schemes for XML. Examples and illustrations of the labelling collisions that may occur using the LSDX labelling scheme are outlined in [19].

In [13], the authors propose a prefix-based labelling scheme call ImprovedBinary that uses bit strings in conjunction with a recursive algorithm to assign unique and persistent labels to each node in the XML tree. Figure 6 illustrates an ImprovedBinary labelled XML tree; the grey nodes indicating newly inserted nodes in an existing tree. When the XML tree is initially constructed, the root node is assigned the empty string. Initially the leftmost child of the root node is assigned the positional identifier *01* and the rightmost child of the root node is assigned the positional identifier *011*. From this point onwards, the Labelling algorithm is a recursive function that takes three inputs; an array of nodes (corresponding to all sibling children of a given node), the label of the leftmost sibling node and the label of the right-

most sibling node. An `AssignMiddleSelfLabel` function is invoked to compute a binary string (positional identifier) for the middle node residing between the leftmost and rightmost sibling nodes (e.g.: node 0101 in figure 6). The middle node is determined using the simple calculation ((1 + n) / 2)) where n is the number of sibling nodes passed to the Labelling algorithm. The `AssignMiddleSelfLabel` function takes into account both values of the leftmost and rightmost nodes as well as their lengths to compute a binary string identifier that is minimal in length while ordered lexicographically between the leftmost and rightmost node labels. This is always possible due to a useful property of the algorithm that ensures the computed binary string always ends with *1*. Finally the labelling algorithm uses the new left and right node labels to recursively call itself until each node in the XML tree has a label.

There are three types of node insertions possible. To insert a new node before the first sibling node, the positional identifier of the inserted node is assigned the identifier of the first sibling node with the last *1* changed to *01* (e.g.: node 0101.001 in figure 6). To insert a new node after the last sibling node, the positional identifier of the inserted node is assigned the identifier of the last sibling node with an extra *1* concatenated (e.g.: node 0101.011 in figure 6). To insert a node between any two nodes, the `AssignMiddleSelfLabel` function is used to compute the new positional identifier of the node (e.g.: node 011.0101 in figure 6).

The ImprovedBinary labelling scheme ensures that the positional identifiers and node prefixes are lexicographically ordered and consequently node labels are lexicographically ordered when performing component by component comparisons. This labelling scheme permits the evaluation of ancestor-descendant, parent-child and sibling-based relationships. However, the label sizes can grow quite rapidly. In particular, repeated insertions before the first sibling node and after the last sibling node has a bit-growth rate of 1 for each insertion. Also, the ImprovedBinary labelling scheme cannot completely avoid the relabelling of existing nodes due to the overflow problem.

4. THE OVERFLOW PROBLEM

All of the labelling schemes surveyed thus far cannot avoid the overflow problem. In this section we outline the overflow problem and then provide an overview of the dynamic labelling schemes that overcome this problem. All of the labelling schemes outlined in this section are orthogonal to the different classifications of labelling schemes; in other words, they may be applied to and used in conjunction with existing containment schemes, prefix schemes and prime number based schemes.

The overflow problem concerns both fixed length labelling schemes and variable length labelling schemes. It should be clear that all fixed length labelling schemes are subject to overflow once all the assigned bits have been consumed by the update process and consequently require the relabelling of all existing labels. It is not obvious that most variable length labelling schemes designed to date are also subject to the overflow problem. Variable length codes require the size of the code to be stored in addition to the code itself. Thus, if many nodes are inserted into the XML tree, at some point the original fixed length of bits assigned to store the size of the code will be too small and overflow, requiring all existing nodes to be relabelled. This problem has been named the overflow problem in [14].

The authors of the ImprovedBinary labelling scheme proposed a novel dynamic quaternary scheme called QED [14] that can completely avoid the relabelling of nodes in the presence of updates. The QED labelling scheme is conceptually similar to the approach taken by the ImprovedBinary scheme. However, instead of using a binary string, a quaternary code is employed consisting of four numbers *0*, *1*, *2*, *3* and each number is stored with two bits, i.e.: *00*, *01*, *10*, *11*. Moreover, the number *0* is reserved for use as a separator and only *1*, *2*, and *3* are used in the QED code itself. The Labelling algorithm is also a recursive function and operates in a similar manner to its counterpart in the ImprovedBinary scheme. The distinction arises from the fact that the ImprovedBinary scheme is based on the $(\frac{1}{2})^{th}$ node position whereas the QED scheme is based on $(\frac{1}{3})^{th}$ and $(\frac{2}{3})^{th}$ node positions. The `AssignMiddleSelfLabel` function is replaced with the `GetOneThirdAndTwoThirdCode` function. Thus, rather than computing a QED code for the middle node, two QED codes (positional identifiers) are computed, one each for the $(\frac{1}{3})^{th}$ and $(\frac{2}{3})^{th}$ nodes that reside between the leftmost and rightmost sibling nodes. The `GetOneThirdAndTwoThirdCode` function takes into account the values of the leftmost and rightmost sibling nodes as well as their lengths to compute two QED codes that always have the following lexicographic order properties: Left node $< (\frac{1}{3})^{th}$ node $< (\frac{2}{3})^{th}$ node $<$ Right node. The Labelling algorithm recursively calls itself until all nodes in the XML tree have been labelled. The key mechanism employed to overcome the overflow problem is the use of the separator *0* (2 bits) to separate the different codes instead of explicitly storing the size of each variable code. The QED codes may vary in size but the size of the separator *0* remains constant. Each number in the QED code will always be represented by two bits and due to the properties of the labelling scheme, the numbers will never have the 2-bit value *00*, which has been reserved as the separator.

The QED codes are lexicographically and not numerical ordered. Furthermore, the properties of the QED labelling scheme ensure that an infinite number of QED codes may be inserted between any two consecutive labels without the need to relabel existing nodes and document order will be maintained. The QED labelling scheme is orthogonal to the different classifications of labelling schemes. However, when applied to prefix based labelling schemes, in the case that nodes are repeatedly inserted at a fixed position, the size of the QED-Prefix label increases rapidly. The authors of QED attempted to address this problem in [15] and proposed a novel Compact Dynamic Binary String (CDBS) labelling scheme which is a highly compact adaptation of the ImprovedBinary labelling scheme with more efficient update costs. However, these improvements were made possible through the use of fixed length bit encoding of the labels and thus, are subject to the overflow problem mentioned earlier. A more compact version of QED is presented in [16] called the Compact Dynamic Quaternary String (CDQS) labelling scheme, which can completely avoid relabelling existing nodes in the presence of node insertions.

Labelling Schemes	Document Order	Encoding Rep.	Persistent Labels	Xpath Eval.	Level Enc.	Overflow Prob.	Orthogonal	Compact Enc.	Division Comp.	Recursion Alg.
XPath Accelerator [9]	Global	Fixed	N	P	F	N	N	F	F	F
XRel [30]	Global	Fixed	N	P	F	N	N	F	F	F
Sector [23]	Hybrid	Fixed	N	P	N	N	N	P	F	N
QRS [2]	Global	Fixed	N	P	N	N	N	P	F	F
DeweyID [22]	Hybrid	Variable	N	F	F	N	N	N	F	F
Ordpath [18]	Hybrid	Variable	F	F	F	N	N	N	N	F
DLN [3]	Hybrid	Fixed	N	F	F	N	N	N	F	F
LSDX [7]	Hybrid	Variable	N	F	F	N	N	N	F	F
ImprovedBinary [13]	Hybrid	Variable	F	F	F	N	N	N	N	N
QED [14]	Hybrid	Variable	F	F	F	F	F	N	N	N
CDQS [16]	Hybrid	Variable	F	F	F	F	F	F	N	N
Vector [27]	Hybrid	Variable	F	P	N	F	F	F	F	N

Figure 7: Evaluation Framework

A novel compact dynamic labelling scheme for XML is proposed in [27] based on vector encoding. The Labelling algorithm follows a similar approach as the QED algorithm, in that initially the leftmost and rightmost nodes are assigned fixed values - vectors (1,0) and (0,1) respectively. Thereafter, a recursive function is called that assigns to the middle node a vector that equals the sums of two vectors that corresponds to the start and end positions in each iteration. Document order is maintained among nodes based on the numerical order of the gradients of the vector labels. Although the gradient of a vector is defined in terms of division, this labelling scheme exploits the property permitting the comparison of the gradient of two vectors via multiplication; that is G(A) > G(B) iff y1x2 > x1y2.

The vector labelling scheme has many of the advantages of the QED labelling scheme. It is orthogonal to the different classifications of labelling schemes. The authors provide empirical evidence to show that the update processing costs are less expensive than QED and in particular, under skewed insertions (frequent insertions at a fixed position), the vector label growth rate is much slower than QED under similar conditions. The authors state the vector labelling scheme completely avoids the relabelling of existing nodes in the presence of updates by using UTF-8 encoding to process delimiters. UTF-8 [29] is a variable-length character encoding for unicode. UTF-8 uses a variable number of bytes (one to four bytes) to encode different integer values. However, given that the largest integer that may be encoded with a single UTF-8 4byte instance is 2^{21}, it is unclear how the vector labelling scheme uses UTF-8 to process delimiters for larger integer values and thus avoid the overflow problem.

5. EVALUATION FRAMEWORK

In §3, we provided an outline of the principle dynamic labelling schemes for XML currently available and indicated their key advantages and limitations. From our analysis, we have attempted to extract the core properties that are representative of the characteristics of a *good* dynamic labelling scheme for XML. These properties should constitute the principle components of an evaluation template or framework of metrics by which all new and existing labelling schemes could be evaluated. To the best of our knowledge, an evaluation template of properties for dynamic schemes in XML has not been presented in current research.

5.1 Framework Properties

In this section, these properties are introduced and described briefly. Our evaluation framework is then presented in figure 7. The properties are described as Full (F) compliance; Partial (P) compliance and No (N) compliance. In addition, our evaluation framework will describe the document ordering method and the encoding representation employed by each labelling scheme.

- **Document Order.** This property indicates the approach adopted by the labelling scheme to maintain document order. The advantages and limitations of each approach were outlined in §3.1
- **Encoding Representation.** This property indicates if the labelling scheme requires a fixed-length storage representation or a variable-length storage representation.
- **Persistent Labels.** The labelling scheme assigns node labels that are not just unique but persistent. Persistent labels ensure that all deletion and insertion operations on the XML tree do not effect existing nodes.
- **XPath Evaluations.** The value of a node label permits the evaluation of ancestor-descendant, parent-child and sibling-based relationships. Enabling the evaluation of the above relationships from the node label alone contributes significantly to the reduction of XPath processing costs.
- **Level Encoding.** The level or nesting depth of a node in the XML tree can be determined from the label value of a node. By encoding the level information in the label of a node, it eliminates the need for an extra join operation within an XML encoding scheme when processing several XPath axes.

- **Overflow Problem.** The labelling scheme is *not* subject to the overflow problem presented in §4 and consequently completely avoids relabelling under various update scenarios.

- **Orthogonal Labelling Scheme.** The labelling scheme may be applied to and used in conjunction with existing containment schemes, prefix schemes and prime number based schemes.

- **Compact Encoding.** The labelling scheme support a compact storage representation and maintains a reasonably constrained growth rate under various update scenarios such as: frequent random updates, frequent uniform updates and skewed frequent updates (frequent updates at a fixed position).

- **Division Computation.** The labelling scheme is *not* required to perform division computations when initially assigning labels to the XML tree or during an update operation. Division computation may lead to floating-point errors when processing very large numbers.

- **Recursive Labelling Algorithm.** The labelling scheme does not employ a recursive algorithm to compute and assign labels during the initial construction of the XML tree. The use of a recursive algorithm is more computationally expensive because it requires multiple passes of the XML tree.

5.2 Analysis of Results

An analysis of the evaluation framework presented in figure 7 provides interesting results. No two labelling schemes share the same properties. This is a positive finding as there is no *one size fits all* solution to the XML update problem. There is a natural tension between the requirements of query optimisation and those of update efficiency. The evaluation framework can provide assistance in the selection of a dynamic labelling scheme for an XML repository by enabling the database designer or data modeller to select the labelling scheme that is most suitable for their requirements. For example, a repository that may want to record document history and enable version control would select a labelling scheme supporting persistent labels. Alternatively, an XML repository that is expected to consume very large documents on a regular basis may consider a labelling scheme that is not subject to the overflow problem. From the evaluation template, it can be seen the CDQS labelling scheme satisfies the greater number of properties and thus, may be considered as the labelling scheme that is most generic.

6. CONCLUSIONS

In this paper, we sought to provide a set of desirable properties for labelling and encoding of XML documents to support an efficient updating mechanism. As part of this research, we examined many of the available schemes and focused on the primary issue of accommodating structural updates in XML schemas and documents. The output was a clear identification of those properties that were essential to good update mechanisms and those that were required key engineering decisions before they could be included, or in the manner in which they could be used.

While the motivation behind this research was to create a list of desirable (and sometimes essential) properties for update mechanisms, we have omitted several schemes. These include the Prime Number labelling scheme [25] and the newer DDE labelling scheme [28]. Using our existing framework, we will now seek to evaluate these and other schemes, although it is possible that we may accommodate some new features in our template as new schemes are incorporated in our study.

7. ACKNOWLEDGEMENTS

This work is supported by Enterprise Ireland under grant CFTD/07/201.

8. REFERENCES

[1] S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas, and D. Srivastava. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In *ICDE*, pages 141–152, 2002.

[2] T. Amagasa, M. Yoshikawa, and S. Uemura. QRS: A Robust Numbering Scheme for XML Documents. In *ICDE*, pages 705–707, 2003.

[3] T. Böhme and E. Rahm. Supporting Efficient Streaming and Insertion of XML Data in RDBMS. In *DIWeb*, pages 70–81, 2004.

[4] E. Cohen, H. Kaplan, and T. Milo. Labeling Dynamic XML trees. In *PODS*, pages 271–281, 2002.

[5] M. Dewey. Dewey Decimal Classification (DDC) system. `http://www.oclc.org/dewey/`.

[6] P. F. Dietz. Maintaining Order in a Linked List. In *STOC*, pages 122–127, 1982.

[7] M. Duong and Y. Zhang. LSDX: A New Labelling Scheme for Dynamically Updating XML Data. In *ADC*, pages 185–193, 2005.

[8] M. Duong and Y. Zhang. Dynamic Labelling Scheme for XML Data Processing. In *OTM Conferences (2)*, pages 1183–1199, 2008.

[9] T. Grust. Accelerating XPath Location Steps. In *SIGMOD Conference*, pages 109–120, 2002.

[10] M. Kay. *XPath 2.0 Programmer's Reference*. Wiley, 2004.

[11] D. D. Kha, M. Yoshikawa, and S. Uemura. An XML Indexing Structure with Relative Region Coordinate. In *ICDE*, pages 313–320, 2001.

[12] A. A. Khaing and N. L. Thein. A Persistent Labeling Scheme for Dynamic Ordered XML Trees. In *Web Intelligence*, pages 498–501, 2006.

[13] C. Li and T. W. Ling. An Improved Prefix Labeling Scheme: A Binary String Approach for Dynamic Ordered XML. In *DASFAA*, pages 125–137, 2005.

[14] C. Li and T. W. Ling. QED: A Novel Quaternary Encoding to Completely Avoid Re-labeling in XML Updates. In *CIKM*, pages 501–508, 2005.

[15] C. Li, T. W. Ling, and M. Hu. Efficient Processing of Updates in Dynamic XML Data. In *ICDE*, page 13, 2006.

[16] C. Li, T. W. Ling, and M. Hu. Efficient Updates in Dynamic XML Data: from Binary String to Quaternary String. *VLDB Journal*, 17(3):573–601, 2008.

[17] Q. Li and B. Moon. Indexing and Querying XML Data for Regular Path Expressions. In *VLDB*, pages 361–370, 2001.

[18] P. E. O'Neil, E. J. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: Insert-Friendly XML Node Labels. In *SIGMOD Conference*, pages 903–908, 2004.

[19] V. Sans and D. Laurent. Prefix based Numbering Schemes for XML: Techniques, Applications and Performances. *PVLDB*, 1(2):1564–1573, 2008.

[20] A. Silberstein, H. He, K. Yi, and J. Yang. BOXes: Efficient Maintenance of Order-Based Labeling for Dynamic XML Data. In *ICDE*, pages 285–296, 2005.

[21] I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld. Updating XML. In *SIGMOD Conference*, pages 413–424, 2001.

[22] I. Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasundaram, E. J. Shekita, and C. Zhang. Storing and Querying Ordered XML using a Relational Database System. In *SIGMOD Conference*, pages 204–215, 2002.

[23] R. Thonangi. A Concise Labeling Scheme for XML Data. In *International Conference on Management of Data (COMAD '06)*. Computer Society of India, December 2006.

[24] World Wide Web Consortium. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, W3C Recommendation edition, November 2008.

[25] X. Wu, M.-L. Lee, and W. Hsu. A Prime Number Labeling Scheme for Dynamic Ordered XML Trees. In *ICDE*, pages 66–78, 2004.

[26] G. Xing and B. Tseng. Extendible Range-Based Numbering Scheme for XML Document. In *ITCC (2)*, pages 140–141, 2004.

[27] L. Xu, Z. Bao, and T. W. Ling. A Dynamic Labeling Scheme Using Vectors. In *DEXA*, pages 130–140, 2007.

[28] L. Xu, T. W. Ling, H. Wu, and Z. Bao. DDE: From Dewey to a Fully Dynamic XML Labeling Scheme. In *SIGMOD Conference*, pages 719–730, 2009.

[29] F. Yergeau. *UTF-8, A Transformation Format of ISO 10646*, Request for Comments (RFC) 3629 edition, November 2003.

[30] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura. XRel: A Path-based Approach to Storage and Retrieval of XML Documents using Relational Databases. *ACM Trans. Internet Techn.*, 1(1):110–141, 2001.

[31] C. Zhang, J. F. Naughton, D. J. DeWitt, Q. Luo, and G. M. Lohman. On Supporting Containment Queries in Relational Database Management Systems. In *SIGMOD Conference*, pages 425–436, 2001.