An Adaptive Labeling Method for Dynamic XML Documents

Moad Maghaydah
Department of Computing
Macquarie University
Sydney, NSW 2109, Australia
moad@ics.mq.edu.au

Mehmet A. Orgun
Department of Computing
Macquarie University
Sydney, NSW 2109, Australia
mehmet@ics.mq.edu.au

Abstract

There is a growing need to support variant operations on XML documents such as insertions, merging, and concurrent access. The Dewey based labeling method, which is used in some XML data Management Systems, has been considered to be the most suitable technique to support dynamic XML documents. In this paper, we present a new adaptable and space-efficient labeling technique, called PoD (Prefixing on Demand), based on Dewey identifiers. Our technique minimizes the total label size that is generated for general XML documents while maintaining the document order. Furthermore, it supports insertion without relabeling any existing node by providing a parameterized insertion mechanism. Our technique also eliminates the need for the complex variable-length prefix-free algorithm that is used in many other proposed solutions. We also report on experimental label length evaluation between our approach and a recent Dewey based approach, namely ORDPATH, using well-known XML benchmarks.

1. Introduction

Many XML Data Management Systems (XMLMS for short) use Dewey like indices to support operations on dynamic XML documents which can not be supported effectively using the interval-based approaches [9, 10]; from inserting large sub-trees, or merging XML documents, without re-labeling the existing nodes, to fine-grained locking thereby avoiding access to external storage as much as possible [2].

In Dewey based approaches, first adapted to XMLMS by Tatarinov *et al* [1], the label for any particular node in an XML document is built by concatenating the local order values for all the nodes on the path from the root node to that node (e.g., 1.6.20.5). Therefore the label's structure is very critical, because it has a major impact on the label length, processing time, and insertion support, especially if the XMLMS is going to run on limited-resource computing devices (i.e., hand-held devices), or if the XMLMS will be supporting merging XML documents. In the techniques that are used for merging XML documents (e.g. three-way merge [7]), each XML node (elements, attribute, text nodes, etc.) is given a unique key (label) which is supposed to preserve document order and support parent-child relationships. This label is used to track the changes and to speed up the comparison process between the original document and the edited versions of that document. It is very well known that Dewey-based labeling methods support these features; however, the merging process can be made more efficient if the generated label size is kept small so that the whole merging process can be completed in the memory or at least with a minimal number of I/O operations.

To reduce the length of the labels that are generated for a particular XML document using the Dewey based labeling method, we need to reduce, as much as possible, label sizes at the top of the XML tree. Consider for example, that the label length of a given node which has say 50,000 descendant nodes is reduced by 4 bits, then the total label lengths for all the nodes that descend from that node would be reduced by at least 200,000 bits. This clearly represents a substantial space saving.

Tatarinov *et al* [1] encoded the labels for given XML documents in UTF-8 strings. The major drawback of UTF-8 is its inflexibility. Moreover its compression is poor for small ordinals, e.g., the label (1.1.1.1) uses four one-byte components.

Several approaches have been proposed based on the prefix-free algorithm and Dewey Identifiers [3, 4, 5, 11, 12, 17]. The main goal in those approaches is to reduce the length of the generated labels and the cost of relabeling when new nodes are inserted. ORDPATH [3] is a recent Dewey based labeling scheme which was refined and used by Haustein et al [12] as well. With its "careted in" technique, ORDPATH eliminates the need for relabeling any existing node when new nodes are inserted by using the odd numbers for labeling the ordered nodes and the even values as insertion points for the new nodes. However, this scheme has an impact on the total length of all labels, because the system can only use half the number of the total values that are available for labeling in any range and hence the longer labels will be used more often. In fact, insertions do not always happen between every single pair of nodes in

1-4244-1500-4/07/$25.00 ©2007 IEEE

an XML document. This means that, using "careted in" technique to provide insertion support could be costly, especially if bulk insertion is more likely to happen such as when merging large XML documents.

ORDPATH and a few other approaches [4, 5] use the variable length prefix-free technique to prefix a node's order value, which will uniquely identify each labeling range while maintaining the document order. For example, the order value 5 (101_b) would be encoded as (01101_b) and the order value 10 which falls in the next range of labels (0010_b) would be encoded as (1000010_b). The variable length prefix-free technique incurs processing time overhead, since the label processor needs to parse any given label bit by bit to find out the right prefix value.

To mitigate the problems with the current Dewey based techniques discussed above, we introduce a new labeling technique, based on Dewey identifiers, which maintains the document order, parent-child and ancestor-descendent relationships, and supports other variant operations. The new technique, called Prefixing on Demand (PoD), reduces the size of the average label length as well as the maximum label length for an XML document. It also provides short labels for small ordinals, because it reduces the label length for parent nodes at the top of the XML tree. PoD uses fixed-width prefixes, if and when they are needed, instead of variable length prefixes. The use of fixed-width prefixes will make it more efficient for any label parser or processor to handle the resulting labels. Moreover, PoD supports insertion and other update operations without any need for relabeling the existing nodes.

The rest of the paper is organized as follows. Section 2 overviews XMLMS. In Section 3 we discuss in detail the PoD technique. In section 4 we report on experimental label length evaluation between various configurations of PoD and ORDPATH, using well-known XML benchmarks. We show that our technique is more space-efficient than ORDPATH. Section 5 concludes the paper with brief summary and future research directions.

2. XMLMS Overview

XML Management Systems (XMLMS) are emerging as complete systems that can store XML documents and support operations on XML data like querying, insertions, merging and updates.

There are two major approaches in XMLMS; the first approach is the Native XML Management Systems; systems developed from scratch based on the semi-structured data model to handle XML data. Native XMLMS are supposed to store XML documents in their native format and have special indices (e.g., Dewey identifiers) to support queries on the structure and order of a given document. Emerging query languages, like XPath and XQuery, are used to access the data in native XMLMS. However, the native XMLMS are still at early stages with high cost.

The second approach is to reuse mature Relational DBMS to build Enabled XML Management Systems, which can support both relational, and XML data. The SQL language or modified versions of SQL are used to query both types of data. In Enabled XML Management Systems, an XML document is shredded down, labeled, and stored into relations (tables) based on certain mapping techniques. YoshiKawa *et al* [9] classify these mapping techniques into two main approaches: (1) Structure-Mapping: where a database schema is defined for each XML document DTD, and (2) Model-Mapping: where a fixed database schema is used to store the structure of all XML documents. The Model-mapping approach offers advantages over the Structure-mapping approach; first it can be a DTD and XML-Schema independent. Second, it is capable of supporting any sophisticated XML application, be it static or dynamic.

```
<People>
    <Person ID="ABC9999" Gender="M">
        <Name> John Smith </Name>
            <Phone> +61 2 8888 8888</Phone>
            <Address>
                <Work> … </Work
                <Home> … </Home>
            </Address>
    </Person>
</People>
```

Paths

Path_Id	Path
1	/People
2	/People/Person
3	People/Person/@ID
4	/People/Person/@Gender
5	/People/Person/Name
6	/People/Person/Phone

Elements

Label	Path_Id
1	1
1.1	2
1.1.1	5
1.1.2	6

Attributes

Label	Path_Id	Value
1.1.0.1	3	ABC9999
1.1.0.2	4	M

Text

Label	Value
1.1.1.1	John Smith
1.1.2.1	61288888888

Figure 1: A sample XML data and its relational schema

In our work, we follow the model-mapping approach where a fixed database schema is used to store any XML document. A basic database schema of four tables is proposed as follows:

- **Paths** table stores all possible path combinations in an XML document associated with unique path ids.
- **Elements** table contains identification information about all element nodes in the document.

- **Attributes** table contains all attributes and their values. And
- **Text** table contains the values for text nodes.

Figure 1 shows the relational schema mapping for a sample XML document.

3. Prefixing on Demand Labeling Scheme

This section discusses the Prefixing on Demand (PoD) labeling technique by first introducing the concept of Basic Labeling Unit (BLU) and then explaining the way in which insertion is supported in PoD.

3.1 Basic Labeling Unit (BLU)

The basic concept of our labeling scheme is to have a Basic Labeling Unit (BLU) with a fixed length (ℓ bits). Most of the values in the BLU will be used to label nodes based on their document order. In order to address a large number of nodes, some of the highest values within the BLU will be preserved for prefixing the extended labels.

The fixed length BLU eliminates the need for the variable length prefix-free strings and preserves the document order. This technique makes it possible to label more nodes using shorter labels before the need to use extended labels arises.

Proposal 1: A fixed number (ℓ) of bits can be used to label up to $\mathcal{M}$ XML elements within the same parent node where the label value (u) does not have a prefixing component and the value (u) is in the range:

$$0 \leq u < \mathcal{M} \qquad \textbf{(1-a)}$$

$$\mathcal{M} = 2^{\ell} - x \qquad \textbf{(1-b)}$$

where x: is the number of the highest values within the BLU that are preserved for prefixing the extended labels. The lengths of the extended labels extend beyond the BLU length. We call those values the set of prefix values $\mathcal{P}$, where:

$$\mathcal{M} \leq p < 2^{\ell}, \forall\, p \in \mathcal{P} \qquad \textbf{(2)}$$

Based on formulas (1-a), (1-b) and (2) given above, we have the following properties:

- We can label up to $\mathcal{M}$ child nodes of a particular XML node without any need for prefixing; where the label value $u \in \{0,\ldots, \mathcal{M}\text{-}1\}$ and the length $(u) = \ell$.
- Each value p in the set $\mathcal{P}$ is used to prefix a certain extended label.
- The number of the extended labels (x), their corresponding lengths ($\ell_1, \ell_2,\ldots, \ell_x$) and the distance between them ($\ell_j - \ell_i$) can all be adjusted to achieve an optimal label length based on the document structure and the average fan-out value if prior knowledge about a given XML document is available. For example, we might have 3 extended labels with lengths {5, 7, 10} or {6, 8, 12}. For another document, we might have 6 extended labels with lengths {6, 8, 10, 12, 16, 20}…etc.
- The extended label values are of the format: $p.u'$ where $p \in \mathcal{P}$ and $u' \in \{0,\ldots, 2^{\ell'}\}$.

Example 1: For $\ell = 3$ and $x = 2$, we will have that $\mathcal{M} = 2^3 - 2 = 6$ is the maximum number of child nodes that can be labeled without prefixing or using extended labels.
$p \in \mathcal{P} = \{6, 7\}$ prefixing values for the extended labels.
$\ell_i' \in \{5,7\}$; the length of the extended labels.
$u' \in \{0,..,32\}$ for $\ell_1 = 5$ and $u' \in \{0,..,128\}$ for $\ell_2 = 7$
Table 1 shows some Dewey based labels and their representations using PoD labeling scheme (the dot was added for easy of eye reading).

Table 1. PoD Representation for some Dewey labels

Dewey Based Label	PoD Labeling Scheme
1.5.2.1	001.101.010.001
1.20	001.110 01110
1.103.4	001.111 0100011.100

3.2 Insertion Support

The Dewey based labeling techniques for XML documents support insertion of new XML nodes with no need to relabel any of the existing nodes. One approach to achieve that is to use either the even or odd values as insertion points. In other words, the odd numbers, as an example, will be used to label the nodes based on their order in the XML document, and the even values will be considered as virtual parents for the new inserted nodes.

This approach has a size disadvantage regardless of the technique used to build the Dewey labels, because the actual number of values that are available for labeling in any range (i.e., 4-bit value, 6-bit…etc) would be half of the total number of values in that range. This means that the extended and longer labels would be used more often. For example, for a 4-bit string, the total number of values is $2^4 = 16$, but only half of that number (8) will be available for labeling XML nodes and to label more nodes (>8) within the same parent node, extended labels need to be used (.i.e., 6 or 8 bit length…etc).

PoD provides an alternative approach that can support insertion without relabeling and reduce the label length. Based on the BLU that is used in PoD, we allocate the maximum (the last) value in any given BLU as an insertion point instead of using it to prefix extended labels.

Proposal 2: Given a BLU with length (ℓ) and two consecutive nodes $\mathcal{N}_1$ and $\mathcal{N}_2$ with labels $\mathcal{V}_1$ and $\mathcal{V}_2$ respectively, where $\mathcal{V}_2 > \mathcal{V}_1$, if a new node $\mathcal{N}$ is inserted between $\mathcal{N}_1$ and $\mathcal{N}_2$ then the label for node $\mathcal{N}$ is:

$$\mathcal{V}_1.(2^{\ell}-1).1 \qquad (3)$$

Example 2: If a BLU with ℓ =4, $\mathcal{N}_1$ with label $\mathcal{V}_1$= 1.5.1, $\mathcal{N}_2$ with label $\mathcal{V}_2$ = 1.5.2, the values (12, 13, 14) in BLU are preserved to prefix the extended labels. Based on (3) above, the last value in BLU (15) is the insertion point.
The PoD representation for $\mathcal{V}_1$ and $\mathcal{V}_2$ is:

$\mathcal{V}_1$ = **1.5.1 (0001.0101.0001)**
$\mathcal{V}_2$ = **1.5.2 (0001.0101.0010)**

The label value ($\mathcal{V}$) of the new node $\mathcal{N}$ that to be inserted between $\mathcal{N}_1$ and $\mathcal{N}_2$ is:

$$\mathcal{V} = \mathcal{V}_1.15.1 = 1.5.1.15.1$$

And the PoD representation is:

1.5.1.15-1(0001.0101.0001.1111 0001)

3.3 Support for Merging XML Documents

In 3-way merge, a new XML document is produced based on the differences and matches between the original XML document and its edited versions. The existence of a unique ID for each node or element within the XML document provides and efficient way to identify the similar nodes between the original document and the modified versions; however, this unique ID is not available in many XML documents. In that case, a labeling mechanism is needed to identify each individual element or node.

PoD provides an efficient labeling method, which might provide useful input for matching and merging applications, because the PoD's label in combination with path ID can be used to identify each single node in the document. Moreover, the tag ID can be used if operations like move and copy have to be supported.

The PoD's label provides information about the ancestors of each node, its siblings, and its order within the whole document; this can efficiently support "find similar-node" algorithms where the nodes are compared based on their context by comparing the parents and siblings (right and left) of both nodes that we need to match.

The insertion support in PoD can be used to identify the new inserted nodes in the edited versions of the document without upsetting the labels of the existing nodes, as shown in Figure 2. With the combination of label and Path ID, from the given base document, we can identify some of the changes in the edited versions during the first parsing phase. This means less refining processes are needed.

Doc $_{base}$

```
<person>
        <name>  </name>
        <email> </email>
</person>
```

PoD Label	PathID	TagID
1	1	1 (person)
1.1	2	2 (name)
1.2	3	3 (email)

Doc $_{1}$

```
<person>
        <name>  </name>
        <phone> </phone>
        <email> </email>
</person>
```

PoD Label	PathID	TagID
1	1	1 (person)
1.1	2	2 (name)
1.1.15.1	4	4 (phone)
1.2	3	3 (email)

Figure 2: A sample base document (left) and its PoD labels. Doc1 (right) an edited version of base document and a suggested first-phase labeling using BLU of 4-bit length.

3.4 Features of PoD Labeling Scheme

The PoD Labeling scheme provides a size-efficient and simple numbering technique for dynamic XML documents.

In PoD, the value 0 of any BLU is used to create a virtual parent node for the attributes of any XML node, as shown in Figure 3, because the attribute nodes are certainly leaf nodes and the label lengths of XML attributes do not have that much impact on the total label size in most cases.

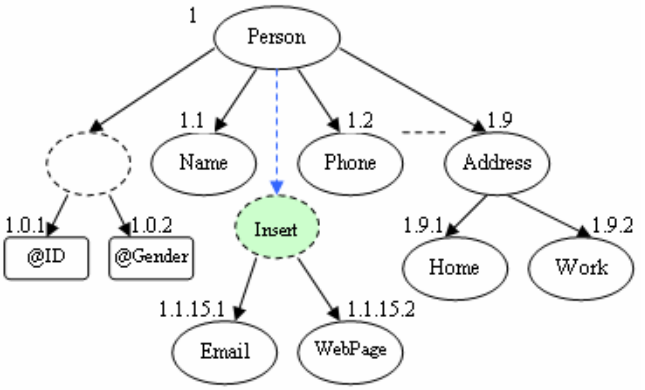

Figure 3: PoD labeling for a sample People database

One advantage of this technique is that it leaves the other values within the BLU to create short labels for element nodes, some of which might have children and grandchildren nodes.

PoD with its variable parameters (ℓ: length of the BLU, x: the number of prefix values, and ℓ_i`: the length of extended labels), can be easily configured to suit various document sizes and structures. However, prior knowledge about the document's size and structure can help in setting up a more efficient PoD with variant BLUs within the same document.

Double phase scanning can be used to analyze the document and set up proper BLU values based on the maximum and average fan-out values for each node's type in the document (that is, the unique path, not the unique instance of each path). For example, if the first phase parsing shows us that the path (/Regions) has only 6 child nodes (Asia, Africa, Europe…etc) and another path like (/Regions/Europe/Country) has 50 children, then we can assign (BLU=3) to /Regions and assign (BLU=5) to /Regions/Europe/Country which would give a more efficient label size than using a single BLU for the whole document. The main advantage of this technique is a smaller label size but the label structure would be more complex and it would naturally take longer time to parse it for insertion operations.

The PoD labeling scheme also eliminates the need for parsing the node labels bit-by-bit, which cannot be avoided in other variable length prefix free models. In PoD, any label-parsing function would pick up a fixed-width block of bits equal to the width of the BLU and process it all together, and based on the value, it can easily determine whether this block is a label value or a prefix value and how many bits it should pick up on the next run.

4. Label-length Experiments

We have conducted extensive experiments to evaluate the efficiency of PoD in reducing the average and maximum lengths of generated labels. We have compared our results against those of ORDPATH [3]. We did not include the interval-based approaches like XRel [9] in the tests, because those approaches do not provide efficient support for dynamic XML documents. Moreover, they use 2 or 3 integer-number components to build a label which means that the label size is always fixed (8 or 12 bytes) and that leads in most cases to a larger label size.

The tests were conducted on a 2.6MHz Pentium 4 machine with 512MB RAM. We used open source software for XML SAX Parser. We have used XML documents from four different XML benchmarks: XMark [14], XMatch-1 [15], Shakespeare's plays [16], and Michigan benchmark [17]. All documents were generated using the tools that were provided with each benchmark. The XML documents were parsed and stored into a simple relational schema primarily to measure the size of the generated labels.

The experiment aimed at the evaluation of PoD against small and medium size documents with different structures and without any prior knowledge about the document structures and sizes.

We have tested PoD using 6 variant configurations based on the BLU length and the number of values allocated for prefixing the extended labels. For each particular BLU length, there were 2 configurations: one for small documents and another for medium-size documents, as shown in Table 2.

Table 2. PoD configurations that were used in the test

BLU configs	Label Values	Prefixing Value/Length of Extended label(bit)
PoD3-S	{0,…,3}	(4/5)(5/7)(6/10)
PoD3-L	{0,1}	(2/5)(3/7)(4/9)(5/11)(6/14)
PoD4-S	{0,…,11}	(12/6)(13/8)(14/10)
PoD4-L	{0,…,9}	(10/6)(11/8)(12/10)(13/12)(14/16)
PoD5-S	{0,…,27}	(28/7)(29/9)(30/11)
PoD5-L	{0,…,25}	(26/6)(27/8)(28/10)(29/12)(30/16)

The results in figure 4 show that PoD can efficiently support XML document of different sizes and structures.

BenchMark			Shaks20	Xmach	Xmark	Michiga
Document			Hamlet	Largest 10 Docs	Auction	Doc1
Size (MB)			0.28	1	11	47
No. of Nodes			6665	10394	235737	670714
Label Length (Byte)	PoD3-S	Max Length	5	10	F*	16
		Avg. Length	3.88	4.18	F	11.52
	PoD3-L	Max Length	6	11	11	17
		Avg. Length	4.62	5.30	5.38	14.98
	PoD4-S	Max Length	4	8	F	10
		Avg. Length	3.22	3.69	F	8.37
	PoD4-L	Max Length	5	8	9	10
		Avg. Length	3.38	3.85	4.51	8.4
	PoD5-S	Max Length	5	8	10	12
		Avg. Length	3.51	3.98	5.09	10.21
	PoD5-L	Max Length	5	8	10	12
		Avg. Length	3.56	4.01	4.96	10.21
	ORDPATH A	Max Length	6	10	11	17
		Avg. Length	4.54	4.82	5.79	14.62
	ORDPATH B	Max Length	6	11	10	21
		Avg. Length	4.49	4.88	5.4	17.15

*Failed: the labeling space was not enough

Figure 4: The results of extensive label length experiments on several benchmark XML documents (PoD3, PoD4 and PoD5 versus ORDPATH)

The PoD3 configurations (BLU's length = 3bits) did not provide a significant advantage over ORDPATH, because the PoD3 configurations are very close to the idea of variable length prefix-free labeling where most values in the BLU are preserved for prefixing. However, it still has one advantage of the fixed width prefix.

PoD4 configurations performed the best among other configurations with reductions on the average and maximum label lengths by more than 20% of those in ORDPATH. The BLU of 4 bits strongly represents the idea of PoD.

On the other hand, PoD5 configurations also performed well, but BLUs with a length of more than 4 bits are not

optimal for general use. They might be beneficial to use for certain levels in the middle of the XML tree for some particular applications because of specific requirements on label sizes and processing time.

We have conducted another type of test for very large XML documents based on double-phase parsing and variable BLU configurations. The preliminary results (not discussed here) show that variable BLU as well as 4-bit pre set BLU outperformed ORDPATH by more than 30%.

5. Conclusions

We have proposed a new labeling technique based on Dewey Identifiers, called Prefixing on Demand (PoD). Our technique reduces the average label length as well as the maximum label length for general XML documents without any prior knowledge about the document structure (i.e., DTD or XSchema). Moreover, PoD with its parameterized features can use the XML metadata to further reduce label lengths for specific XML documents in applications where the label size or index is much more important than other considered factors. Furthermore, PoD minimizes the processing overhead by not using variable length prefix-free binary strings. Instead, PoD uses predefined fixed-width blocks that can be adjusted throughout a given XML document.

Our new labeling scheme allows the insertion of new nodes without the need for relabeling the existing nodes and without skipping any odd or even values, which makes it more suitable for bulk insertion. These features make PoD very suitable for variant operations on XML documents including merging.

In future work, we plan to evaluate the PoD's querying and insertion performance against those of other recent Dewey-based approaches. Further research will be conducted to evaluate tailored versions of PoD to suit particular applications such as those running on limited-resource computing devices, making use of PoD's parameterized features.

References

1. Tatarinov, I., et al., *Storing and Querying Ordered XML Using a Relational Database System*, in *ACM SIGMOD*. 2002. p. 204-215.
2. Härder, T., *XML Databases and Beyond - Plenty of Architectural Challenges Ahead*, in *ADBIS*. 2005, Tallinn, Estonia.
3. O'Neil, P., et al. *ORDPATHs: Insert-Friendly XML Node Labels*. in *the ACM SIGMOD International Conference on Management of Data*. 2004. Paris, France.
4. Böhme, T. and E. Rahm. *Supporting Efficient Streaming and Insertion of XML Data in RDBMS*. in *Third International Workshop on Data Integration over the Web*. 2004. Riga, Latvia.
5. Cohen, E., et al. *Labeling Dynamic XML Trees*. in *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. 2002. Madison, USA.
6. Florescu, D. and D. Kossmann, *Storing and Querying XML data using an RDBMS.* IEEE Data Engineering Bulletin, 1999. **22**(3).
7. Lindholm, T. A Three-Way Merge for XML Documents, in DocEng 2004, Milwaukee, Wisconsin, USA.
8. Shanmugasundaram, J., et al., *Relational Databases for Querying XML Documents: Limitations and Opportunities*, in *Proceedings of the 25th VLDB Conference*. 1999. p. 302-314.
9. YoshiKawa, A., *XRel: A path-based approach to Storage and Retrieval of XML Documents using Relational Database.* ACM Transactions on Internet Technology, 2001. **1**(1).
10. Zhang, C., et al. *On Supporting Queries in Relational Database Management Systems*. in *SIGMOD*. 2001. Santa Barbara, California USA.
11. Maggie, D. and Y. Zhang. *LSDX: A new Labeling Scheme for Dynamically Updating XML Data*. in *the 16th Australian Database Conference*. 2005. Newcastle, Australia.
12. Haustein, M.P., et al. *DeweyIDs - The Key to Fine-Grained Management of XML Documents*. in Proc. of the *20th Brazilian Symposium on Databases*. 2005. Uberlandia, Brazil.
13. Schmidt, A., et al. *XMARK: A Benchmark for XML Data Management*. in *Proceedings of the 28th VLDB Conference*. 2002. Hongkong, China.
14. Böhme, T. and E. Rahm. *Multi-user Evaluation of XML Data Management Systems with XMach-1*. in *EEXTT*. 2002. Hong Kong, China.
15. Bosak, J., *Shakespeare 2.00*. 1999, http://www.cs.wisc.edu/niagara/data/shakes/shaksper.htm.
16. Runapongsa, K., et al., *The Michigan Benchmark*, http://www.eecs.umich.edu/db/mbench/.
17. Maghyadah, M. and M. A. Orgun., *XMask: An Enabled XML Management System*, in ADVIS 2006, Izmir,Turkey.