Multi-GranularityUser Friendly List Locking Protocol for XML Repetitive Data

Eunjung Lee *ejlee@kyonggi.ac.kr*
Computer Science Department
Kyonggi University
Suwon, South Korea

Abstract

We proposeproposed a list data sharing model, which utilizes semantics expressed in DTD for concurrency control of shared XML trees. In this model, tree updating actions such as inserting and/or /deleting subtrees are allowed only for the repetitive parts. The proposed model guarantees that the resulting XML tree is valid even when applying tree update actions are applied concurrently. AlsoIn addition, we propose, a new multi-granularity locking mechanism called *list locking protocol* is proposed. This protocol locks on the (index) list of repetitive children nodes, so and thus it allows updates on to the descendents when the node's node child child's subtree is being deleted or inserted. This protocol is expected to show betterbe more accessible accessibility with less number ofand to produce fewer locking objects on XML data compared to the other locking methods on XML data. Moreover, the prototype system shows that list locking is well suited to user interface of shared XML clients by enabling/disabling corresponding edit operation controls.

Keywords: Locking, Shared XML, Repetition, Update.

1. INTRODUCTION

The rapid proliferation of the XML in many different application area areas results has resulted in a rapidly growing number of XML documents. Also, it becomes and allowed more possible that users workingwork concurrently on XML documents to share data. Isolating concurrent accesses has become becomes an important issue in XML database (DB) systems or distributed applications based on shared XML documents.[1]

Locking is the standard way to control concurrency in relational databasesDBs (RDBs). Multi-granularity locking is used to resolve the tradeoff between concurrency and overhead [1]. Rather than force a single locking granule for all transactions, multi-granularity allows a transaction to select a granularity level at which to obtain locks. For XML trees, however, a multi-granule lock on a node blocks the whole subtree and therefore reduces concurrency.

Recently, the topic of synchronization was picked up againreaddressed in the context of XML. Several locking approaches are have been proposed for shared XML data, such as OO2PL [8] and path lock [5]. Most of these researches research has focused on the performance of shared query processing and, therefore concerns is mainly concerned with mainly on reading a large number of nodes, rather than updating the tree structure, such as through subtree insertion and /deletion.

In this paper, we investigate a new efficient locking protocol for synchronizing concurrent tree structure update actions. Studying By studying many application examples, we found that structure update actions on a shared XML tree usually applied apply to the repetitive parts of the

[1] This work was supported by the GRRC program (GRRC Kyonggi 2011-B03) of Gyeonggi province. This work was supported by the GRRC program(GRRC Kyonggi 20101-B03) of Gyeonggi province.

tree. This motivates us to propose a *list data sharing model* for XML data, which restricts structure update actions only for the repetitive parts. The proposed model guarantees that the resulting XML tree is valid after applying the application of concurrent tree update actions.

A new multi-granularity locking (MGL) mechanism (MGL), called the *list locking protocol,* is proposed. This protocol locks on the (index) list of repetitive children nodes for handling the concurrent insertion and /deletion of subtrees. This protocol allows update actions on descendents during the insertion and deletion inserting/deletingof subtrees. List The list locking protocol is expected to show better accessibilitybe more accessible and to produce with less fewer number of locking objects for synchronizing structure update actions compared to the other locking methods on XML data. Moreover, the prototype system shows that list locking is well suited to user interface of shared XML clients by enabling/disabling corresponding edit operation controls.

This paper is organized as follows. Section 2 discusses related work. Section 3 gives describes the motivation for the development of our model. Section 4 describes our model of shared XML data and the locking algorithm. In section 5, we discuss the proposed method compared with previous methods. Finally, section 6 has ourdiscusses our conclusions and conclusion and future work.

2. RELATED WORK

A multiMGL-granularity locking protocol protocols on RDBs as well as on Object-oriented DBs (OODBs) is are a well well-established research area. Especially, it is studied, and particular focus has been placed on OODBs using DAGs, which to allow locks on granules of groups of objects. Lee et al. proposed a new MGL for composite objects [11], where in which collections of objects can have arbitrary intersection and inclusion relationships.

An XML tree has a hierarchical structure, which that is not a DAG. The node relations could be a general DAG if we use ID or IDREF attribute attributes are used as a relationrelations between nodes. However, we consider XML trees as simple hierarchical structurestructures. Manipulating shared actions on trees, tree locking considers shared trees with structure update actions [10]. It is based on two- phase locking, which allows locks only on individual nodes. Also, Tthe tree locking protocol does not allow for the deletion of deleting subtrees, but only simple nodes.

For allowing structure update actions, one of the most recent results studies on lock-based synchronization for XML trees is OO2PL [7,8] proposed by Helmer et al. OO2PL considers Considering structure updates as changing pointers between nodes, OO2PL and therefore locks on the pointers. Because it uses low-level physical data structure as locking units, Their the model is powerful enough to allow arbitrary type of node accesses access and structure updates since it uses low level physical data structure as locking units. However, if the structure updates on to XML trees should guarantee validity, then not all structure update actions we cannot allow allcan be allowed structure update actions. Therefore, in our opinion, allowing a method that allows any type of structure updates update, such as like OO2PL, is too general for a model sharing model of XML trees. They Helmer et al. mentioned noted that using document document-type information and validity to could enhance performance, but present presented only an informal way to of using dummy nodes for repetitive parts.

Dekeyser and Hidders proposed a path lock [5], a new fine granularity locking scheme of fine granularity based on path locks. They provide provided an algorithm to support general path notations such as //A//B for accessing data. Their method could allow both top-down and bottom-up query evaluation and locking, and therefore shows better efficiencyis more efficient. However, The path lock method only supports the deletion of leaf nodes, however, and not thedoes not support deletion of subtrees, either. Only deletion of leaf nodes is allowed. On the other hand,However, Grab et al. proposed DGLOCK, which controls concurrent accesses using predicates on data guide [6].

Recently, sSeveral recently published studies have shown implementations and their concurrency issues have been published. They have shown that effective and efficient locking protocols are essential to guarantee the ACID properties for XML processing and to achieve high transaction throughput. Most native XML database systems adopt one or more of locking protocols [12,13,–14]. However, we found that there are is still rooms room for improvement in concurrency efficiency, especially for insert and /delete operations, since because most of the current researches research are has focused on retrieval and data update operations. In this paper, we aim to propose a new locking protocol which that is feasible for the a cooperation cooperative environment with heavy a large number of insert and /delete operations.

3. MOTIVATION

In this section, we will look intoexamine sharing patterns in XML XML-based collaboration collaborative applications. From this observation, weWe have found that structure update actions such as subtree insertioninsertions and /deletions are often applied to repetitive parts in XML, represented as with the symbol “*” in a DTD. This motivates us to propose a new model for XML data sharing.

3.1 A Working Example

The following XML file includes order details and the process records. Lists 1 and 2 show instances of XML file and a DTD, respectively.

In a shared XML tree, the tree structure is updated with actions such as inserting and deleting subtrees. Allowing an arbitrary structure update might result in an invalid tree. Moreover, we should assume that more than one update action is applied concurrently to the shared tree. Therefore, in general, it is natural to design the shared tree such that update actions on the shared structure can be applied to the repetitive parts. For the example in Figure 1, adding/deleting an item to/from <order> and adding/deleting an order to/from <orders> are such candidates. In an XML tree, we can identify repetitive parts based on a given DTD, with nodes corresponding to symbols enclosed with * or +.

```
<orders>
 <order>
  <order-by>John</>
  <status>preparation</>
  <date>2001-08-01</>
  <address>…</>
  <item>
   <title>Little Bear</>
   <price>14.00</>
  </>
  <item>
   <title>Blue Horse</>
   <price>21.99</>
   <discount>5</>
  </item>
  <bonus-item>
   <code>BN-01</>
  </>
  <process-records>
   <record>
    <type>payment</>
    <date>
     2001-08-01</>
    <status> OK</>
   </>
   <record>
    <type>findstock</>
    <date>
     2001-08-02</>
    <corr-p>E. Lee</>
   </>
  </process-records>
 </order>
 <order>
  <status>shipped</>
  <date>2001-08-13</>
  <item>
   <title>Mountain</>
   <price>54.00</>
   <copies>2</>
   <discount>10</>
  </>
  <item>
   <title>River</>
   <price>46.50</>
   <discount>10</>
  </>
  <process-records>
   …
  </process-records>
 </order>
</orders>
```

TABLE 1: XML instance for orders.

```
<orders> := (<order>)*
<order> := <order-by>?<status><date><address>?(<item>|<bonus-item>)+<process-records>
<process-records> := (<record>)*(<cancel-log>|<close-date>|<return-log>)
<item> := <title><price><copies>?<discount>?
<record> := <type><date><corr-p>?<status>?
…
```

TABLE 2: DTD for orders instance in Table 1.

We found similar patterns from other examples in the literature, such as auction data and shopping baskets. In an auction status list, adding a new bid to the bid list and deleting an auction item from the list are applied to the repetitive parts. In a shared shopping basket, adding or deleting a new item is the same action. As shown from these examples, shared actions that update the tree structure are often applied to the repetitive parts of the tree.

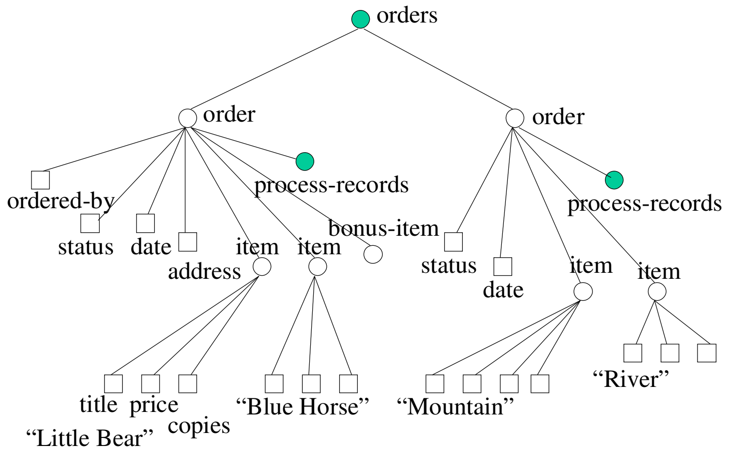


FIGURE 1: An example of an XML tree

We found similar patterns from other examples in the literature, such as auction data and shopping baskets. In an auction status list, adding a new bid to the bid -list and deleting an auction item from the list are applied to the repetitive parts. In a shared shopping basket, adding or /deleting a new item into basket is the same action. As shown from these examples, shared actions that update the updating tree structure are often applied to the repetitive parts of the tree.

3.2 List Data Sharing Model

Actions for sharing XML data are 1) Read(w) or Write(w, $value_{new}$) on a terminal node w, ; 2) Traverse(v), to read a list of children of the internal node v, ; 3) Insert(v, T_{new}, k), to insert a subtree T_{new} as k-th child of v, ; and 4) Delete(v, k), to delete the k-th subtree of v.

In general, concurrency control for of actions on terminal nodes like such as read/write data is possible in a same way toa manner similar to that of traditional RDBs. Therefore, we focus on the structure update actions such as subtree insertions and deletions. The main idea of this paper is to simplify the sharing model by restricting structure update actions (Insert and Delete) only for the repetitive parts of the tree. By doing so, the model can guarantee that the intermediate results of the shared tree is are always valid. Also, we can get an efficient locking/concurrency mechanism.

Restricting the insertion and /deletion for of repetitive parts might be a serious shortcoming in some cases, since because it allows neither insertion / deletion of the optional part (? in DTD), nor exchanges of the subtree structure defined by selections in DTD. However, designing those ones as * or + parts in DTD is not difficult and sometimes helpful in our experience, since because we can identify the shared parts of the tree in an earlier stage of development. Moreover, this can help users to understand quite easily the meanings meaning of the data quite easily andas well as their responsibility for collaborating on the data.

In the next section we will present a formal definition for the structure update actions for repetitive parts of a shared tree.

4. CONCURRENCY CONTROL FOR LIST DATA SHARING MODEL

4.1 *-Facting of XML Trees

DTD defines types of XML trees, where in which the children of a node are represented with sequential, optional (?), selective (|), and/or repetitive (* or +) parts[2]. Motivated by the facts fact that the * parts are the main target for shared structure update actions, we introduce an XML tree transformation called *-factoring before presenting our new locking protocol.

We call a the sequence of nodes $X_1X_2...X_n$ a *repeated part*, if they correspond to children of * or + nodes in a DTD graph. If the repeated part consists of one symbol X, then we call it a *list node* (<order> or <item> nodes in Figure Fig. 1). If the subsequent list of nodes belong belongs to a separate parent node, we call it a *list parent node* (<orders> and <process-records> in Figure Fig. 1). An XML tree is called **-factored* if a group of nodes for each repetitive part consists of a separate tree and there is a parent node for each repeated group. In a *-factored XML tree, every repetitive part has one list node and sibling list nodes are that belong to a list parent node. The example in Figure 1 is not *-factored since because the repeated parts of <item> and <bonus-item> do not have a separate list parent node.

We can transform a given XML tree to a *-factored one by introducing new nodes. Algorithm 1 presents the transformation method. This algorithm repeats the factoring if □$_i$ contains the repetitive parts in the next cycle. Therefore, it factors the outmost outermost level of *"s s at each cycle. Algorithm 1 repeats *O(n)* times if the number of repetitive parts in the tree is *n*, and 2 two new nodes are added in each transformation cycle.

Note that the *-factoring algorithm adds several new nodes to the tree. It is easy to recover the original tree, by simply mapping new nodes to null. Hereafter, we will consider only *-factored XML trees to simplify the discussion. Helmer et al. mentioned a similar approach to that added add dummy nodes to group groups of repetitive parts to get better performance [7]. However, they did not provide a formal model.

```
[Algorithm 1] *-Factoring
Input : a valid XML tree T with respect to the given DTD
Output : *-factored tree T'
1. Repeat the step 2 for every node w.
2. For all repetitive parts of the children of w, let $(v_{11}...v_{1m1})(v_{21}...v_{2m2})...(v_{n1}...v_{nmn})$ be a list
   corresponding to □* in DTD, where □ is repeated n times,
   i) For 1 ≤ j ≤ n, if $m_j > 1$, then add a node $v_j$ as a parent of $v_{j1}...v_{jmj}$ in T.
ii) Add a node v as a parent of $v_1..v_n$, in T, which are added in step C, and as a child of w.
```

2 In our model, we use thea notation * instead of * or + for repetitive parts, if the context is clear. We can assume that a delete action on the repetitive list is relevant only when there are is more than one children child in the list for satisfying the + condition.

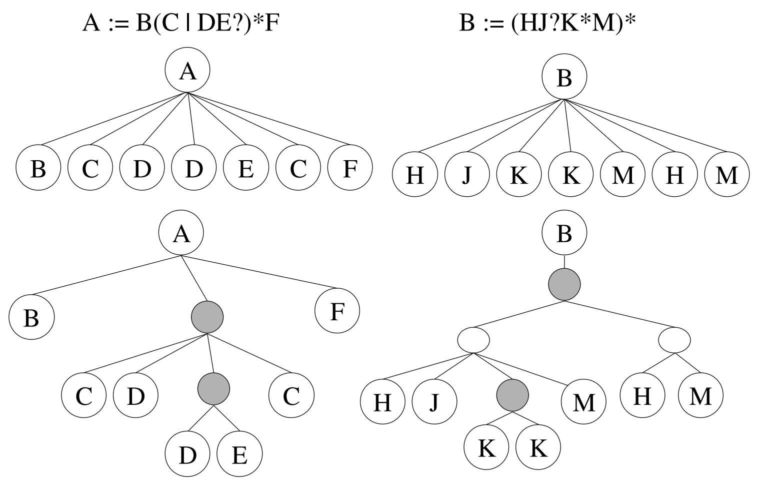


FIGURE 2: *-Factoring of Sample Subtrees.

Top, a production rule; middle, the corresponding tree instance; and bottom, the corresponding *-factored tree for the one above one. The shadowed nodes are newly introduced list parent nodes and the empty circles are new list nodes.

Now, we are ready to introduce the shared structure update actions used in our model.

[Definition 1] Let v_L be a list parent node in a *-factored tree T. Then, *list update action*s on v_L are defined as follows.
ListInsert(v_L, T_{new}, k) : Insert T_{new} as k-th child of v_L.
ListDelete(v_L, k) : Delete the k-th child subtree from v_L. If v_L is a (+)-list parent node, than this action is relevant only if *number-of-children*(v_L) > 1.

4.2 List Locking Protocol

In this paper, we propose a new locking protocol called a *list locking protocol* based on MGL(Multiple Granularity Locking)[1] called a *list locking protocol*. Shared The shared actions on an internal node are Traverse, ListInsert, and ListDelete, as defined in Definition 4.1. There are four types of locks.:

- Traverse lock (T) : A lock request for traversing children or child list nodes.
- List lock (L) : A lock requested request to the list parent node for inserting or /deleting a child tree. This will block other transaction transactions from accessing the same list.
- Delete lock (D) : A lock requested request to the root node of the deleted subtree for deleting.
- Intentional lock (I) : A lock requested request for accessing descendent nodes.

Table 1 shows the compatibility of the list locking protocol. In the table, O shows represents compatibility, X shows reflects conflict, and On means a case allowing write access with a notification.

	I	**T**	**L**	**D**
I	O	O	O	X
T	O	O	X	X
L	O	On	X	X
D	X	X	X	X

TABLE 3: Comparability Relation of the List Locking Protocol.

We did not differentiate intentional locks for traverse and list update updates in the compatibility relations. Note that, in this protocol, L lock is applied only for list parent nodes, and D lock is only for list nodes. In the above table, On shows a possibility ofmay read reading invalidated data. This can be prevented by using multi-version concurrency [2]. Structure update actions check in a new version list index and traverse actions check out the last version. We simply assume that the list update actions notify the clients, if necessary.

[Algorithm 2] List locking protocol:

[1] Locking phase

(1) To access any nodes in the tree, I(v_{root}) should be gained acquired first, where v_{root} is the root node of the tree.
(2) To gain a lock T, L, or I on a node *v*, I(*w*) should be acquired gained first, where *w* is a parent of *v*.
(3) To traverse the child nodes of v, T(*v*) should be acquired gained first.
(4) To delete a child tree of a node *v*, L(*v*) should be acquired gained first and D(v_d), where v_d is the root of the deleted tree.,
(5) To insert a child tree to a node *v*, L(*v*) should be acquired gained first before inserting a subtree.

[2] Release phase

(1) For a client to release a lock on *v*, the lock on all child nodes should be released first.
(2) If an update is allowed for a list parent node *v* with a dirty read, all clients with T(v) should be informed of the a data update. should be notified to all clients acquiring T(*v*) before releasing L(*v*).
(3)(2) For a shared action to get acquire another lock, it should release all of the locks it is currently holding. One exception is to upgrade T(*v*) to L(*v*), which should be allowed in order to update the list after visitingit is visited.

In the list locking protocol, intentional locks to all ancestor nodes first should be gained acquired first in a top-down manner. ThereforeThereafter, it is enough sufficient to get a delete lock on the subtree root node. Following The following two properties show that the list locking protocol guarantees the serializability of list update actions.

[Property 1] In the list locking protocol, no more than one list update actions action are is allowed to the same list parent node.

[Property 2] In the list locking protocol, a delete action is not allowed for any node whose descendent is currently being accessed.

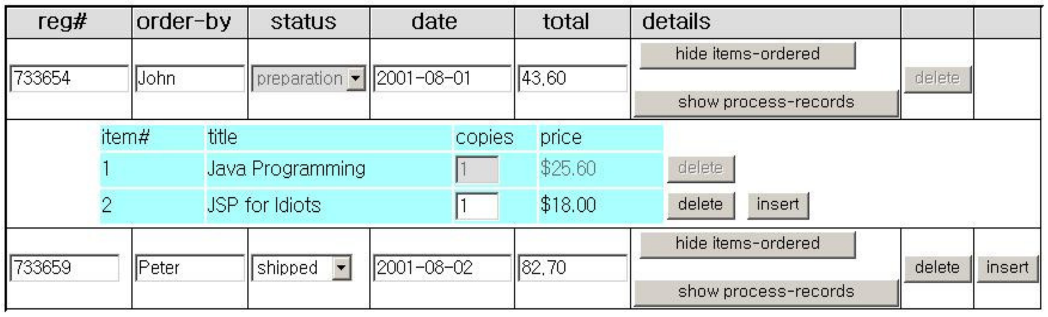
[Property 3] In the list locking protocol, two list update actions applied to different parent nodes can be executed concurrently unless one of the corresponding list parent nodes is deleted by the other action.

The list locking protocol considers structure traversal/update actions. In order toTo cover read/write actions on terminal nodes, the previous MGL could be used togetherwith the list locking protocol. If we model the data read/update as a node traversal, we could combine the proposed protocol with the original MGL on the data values, without conflict.

5. DISCUSSION

5.1 Summary

We have proposed a locking scheme for XML documents that allows the same document to be updated by more than one user. EspeciallyIn particular, we investigate investigated a synchronization method of for structure updates on to shared trees, such as subtree insertion insertions and deletiondeletions. The Our study on of XML sharing applications show showed that subtree insertion/ and deletion is applied for to the repetitive parts of the tree. This motivates our model, called awhich we call a list data data-sharing model, ; the model that restricts subtree insertionsinsertion and /deletiondeletions on the repetitive parts of the tree based on DTDs. This guarantees that the result tree valid even for concurrent structure update actions.



reg#	order-by	status	date	total	details		
733654	John	preparation	2001-08-01	43.60	hide items-ordered show process-records	delete	

item#	title	copies	price	
1	Java Programming	1	$25.60	delete
2	JSP for Idiots	1	$18.00	delete insert

reg#	order-by	status	date	total	details		
733659	Peter	shipped	2001-08-02	82.70	hide items-ordered show process-records	delete	insert

FIGURE 3: A Shared XML Data Interface Based on the Proposed Model.

Formally, we defined a *-factored XML tree and introduced a transformation method from arbitrary XML trees to *-factored ones based on DTDs. By introducing *-factored trees, we could simplify the concurrency control to handling handle a number of index lists. Also, locks on repetitive parts of the tree are introduced; a traverse lock(T) and a list modification lock(L), which consist of a list locking protocol based on MGL. The proposed locking protocol has the following advantages compared to the previous methods.

First, the result tree trees that result after applying structure update actions are guaranteed to be valid. Previous researches research did not consider the validity for of shared actions because of the complexity of the problem. However, we think it is important that the intermediate result trees are valid during XML data sharing. We introduced a relatively simple model for covering ensuring XML trees' validity.

Secondly, the locking protocol becomes efficient because we consider only the repetitive parts. Number The number of locking objects is relatively small. Moreover, the new locking protocol allows for updating other descendents while a child subtree is being deleted or /inserted.

5.2 Comparison

In this section, we compare the proposed method to several recent researches studies on XML data locking.

A locking protocol called OO2PL [7,8], is a general model which that uses the physical links as locking units. They say that theThe researchers argue that serializability should be the foundation for protocols and that thea lowest level of atomic actions should be isolated in order to prevent unwanted side effects. Therefore, OO2PL uses parent/child or sibling links as locking units. Also, they canlinks can be locked lock on links by IDREFs or IDs for direct accessesaccess. However, we believe that their this model is too general to be efficient for relatively simple XML data. Since Because they use arbitrary pointers/links are used for locking units, the number of locking objects should increase significantly. Also, two-phase locking they do notis used instead use of multi-granularity, , but two phase locking instead. Therefore, they couldso it was not possible to not take advantage of the native tree structure. They The researchers mentioned using DTD information to enhance efficiency by restructuring links to group repetitive parts of the children. HoweverAlthough, they describe a way to add dummy nodes to XML tree, no formal method is but did not provide a formal methodprovided.

Another type of protocols protocol that is often used for hierarchical data are is the so-called tree locking protocolsprotocol [10]. In these protocols, locks hold only for nodes do not hold forbut not for entire granules but only for nodes, i.e., when a node is locked its descendants are not also locked. However, there is also the restriction that a lock can only be acquired for a node if an identical or stronger lock was already obtained for the parent of the node.

Path lock [5] is has been proposed to support arbitrary path expressions for accessing nodes in XML trees. It can provide fine granule locks based on path locks. They give anAn algorithm is provided to evaluate locks for a given path expression, and to check compatibility with the previous locks. In this protocol, they can support path expressions such as //A//B can be supported with a minimal number of locking objects. Instead, evaluatingEvaluating compatibility with current locks is not easy. , however. Their The model does not support subtree deletion since because they need to evaluate path conditions must be evaluated in either a top-down or bottom-up waysmanner. We believe that deletingdeletions and /inserting insertions of subtrees are important update actions for shared XML trees. Sometimes, the sharing is more for editing and /managing XML trees rather than for searching and /querying. In that case, our protocol could be useful to serialize structure update actions including subtree deletions.

The main restriction of our study is not that it does not consider navigations through considering ID- and IDREF-based accesses. This is because our approach is focused on the (repeat) structure of the tree definition and updates on to the structure. We treat IDREFS as terminal nodes which can be inserted and/or deleted.

6. REFERENCES

[1] N. S. Barghouti, G. E. Kaiser., “Concurrency control in advanced database applications,”. AACM Computing Surveys, vol.23(3), pp.269-317, 1991.

[2] P. Bernstein, N. Goodman., “Multiversion concurrency control – theory and algorithms,”. ACM TransTransactions. On on Database Systems, vol. 8(4), pp.465-483, 1983.

[3] B. Bouchou, and M. Halfeld and, F. Alves., “"Updates and Incremental Validation of XML Documents,". The 9th International Workshop on Data Base Programming Languages (DBPL), 2003, pp.216-232.

[4] Stijn S. Dekeyser, Jan J. Hidders., “Path locks for XML Document collaboration,”. Proc. WISE'02, 2002, pp.105-114.

[5] Torsten T. Grabs, Klemens K. Bohm and, HansH.-Jorg J. Schek., “"XMLTM: Efficient Transaction Management for XML Documents,"”. CKIM’02, 2002, pp.142-152.

[6] S. Helmer, C. Kanne and, G. Moerkotte, “"Lock-based Protocols for Cooperation on XML Documents”.," Int.International Workshop on DB and Expert Systems Applications Conference (DEXA'03), 2003, pp.230-236.

[7] S. Helmer, C. Kanne and, G. Moerkotte, . “"Evaluating lock-based protocols for coorperation cooperation on XML documents”.," ACM SIGMOD Record, vol. 33(1), 2004, pp.58- – 63.

[8] KuenK.-Fang F. Jea, ShihS.-Ying Y. Chen and ShengS.-H.Hsien Wang., “Concurrency Control in XML Document Databases: XPath Locking Protocol,”. In: Proceedings of the 9th

International Conference on Parallel a6d and Distributed Systems (ICPADS 2002), 2002, pp.551-556.

[9] V. Lanin and D. Shasha., “Tree locking on changing trees,”. Technical Report 503, New York University, 1986.

[10] S.-Y. Lee and R.-L. Liou. “A multi-granularity locking model for concurrency control in object-oriented database systems,”. IEEE Trans.Transactions onOn Knowledge and Data Engineering, vol. 8(1), 1996, pp.144- -- 156.

[11] World wide web consortium, “XForms 1.0 Working draft,” http://www. w3. org/TR/**xforms,** Jan. 2002, [Jan. 28. 2002].

[12] E. Harder, C. Mathis, S. Bachle, K. Schmidt and, A. Weiner., “Essential performance drivers in native XML DBMSs,”. LNCS 5901, 2010, pp. 29-46.

[13] S. Bachle, T. Harder and, M. Haustein., “Implementing and optimizing fine-granular lock management for XML document Trees,”. DASFAA’09, 2009, pp.631-635.

[14] M.Haustein, T.Harder, “Optimizing lock protocols for native XML processing,” Data & Knowledge Engineering, vol.65(1), 2008, pp.147-173.

[15] H. Tan, X. Chen and, J. Gu, . “A transaction mechanism for native XML database,”. Proceedings of the 5th WSEAS International Conference on Applied Computer Science, 2006, pp. 486-490.

[16] P.Pleshachkov, P.Chardin, S.Kusenetzov, “SXDGL: Snapshot based concurrency protocol for XML data,” XSym 2007, LNCS 4704, 2007, pp.122-136.