taDOM: A Synchronization Concept for the DOM API

Michael P. Haustein, Theo Härder

University of Kaiserslautern
P.O. Box 3049, Kaiserslautern, Germany
{haustein, haerder}@informatik.uni-kl.de

Abstract. Storing, querying, and updating XML documents in multi-user environments requires data processing guarded by a transactional context to assure the well-known ACID properties, particularly with regard to isolate concurrent transactions.
In this paper, we introduce the taDOM tree, an extended data model which allows fine-grained lock acquisition for XML documents. For this reason, we design a tailored lock concept using a combination of node locks, navigation locks, and logical locks in order to synchronize concurrent accesses to XML documents via the DOM API. Our synchronization concept supports user-driven tunable lock granularity and lock escalation to reduce the frequency of lock requests both aiming at enhanced transaction throughput. Therefore, the taDOM tree and the related lock modes are adjusted to the specific properties of the DOM API.

1 Introduction

The use of the extensible markup language XML [1] for electronic data interchange leads to an enormous growth of the number and size of files keeping semi-structured XML data. In contrast, only structured data is managed by relational or object-relational database systems ((O)RDBMSs) so far. In order to allow combined processing of XML and relational data in an efficient way, it is indispensable also to maintain XML documents in these database systems, providing the well-known ACID properties [2] for semi-structured data.

Different approaches to store XML documents in a relational database system and their performance characteristics are discussed in detail in [10], [11], and [12]. As a major issue, the ways XML documents are stored in a relational database lead to different synchronization problems. If an XML document is contained in a single CLOB attribute, locking may only take place at the document level. In contrast, if an XML document is shredded and stored across several tables, insertion of a new XML element often results in several insert operations to relational database tables (depending on the shredding algorithm). Due to inadequate synchronization mechanisms, many database systems lock each of the affected tables entirely to prevent phantoms. Hence, such a crude method causes locking of document fragments by accident, even of unrelated documents which are shredded to those tables.

Our primary objective is to develop a mechanism for concurrency control of semi-structured data where the properties of the document access interface are explicitly taken into account. For this reason, we refer to the DOM API [3] (not discussed in this paper) in order to query, traverse, and update XML documents stored in a database. In Section 2, we introduce the taDOM tree, a data model to represent XML documents which is specifically tailored to the properties of the DOM API. Based on the taDOM tree, we present in Section 3 an adjusted lock concept to synchronize concurrent accesses to XML data which also supports tunable lock granularity and tunable lock escalation. Since the DOM API provides both, methods to traverse XML documents node-by-node and methods to apply simple queries, we use, in addition to node locks, a combination of navigation locks for the edges connecting the nodes, and logical locks in order to prevent phantoms. Finally, in sections 4 and 5, we give a brief overview of related work about XML synchronization and wrap up with conclusions and some aspects of future work.

2 taDOM Tree

Synchronizing concurrent accesses to XML documents calls for an appropriate logical representation of XML documents which supports fine-grained lock acquisition. To this end, we have developed the taDOM tree, an extension of the XML tree representation of the DOM [3]. To illustrate its construction principles, we consider a small XML fragment depicted in Figure 1, which is a modified version of the *bib.xml* document in [4].

```
<bib>
   <book year="1994" id="1">
      <title>TCP/IP Illustrated</title>
      <author>
         <last>Stevens</last>
         <first>W.</first>
      </author>
      <price> 65.95</price>
   </book>
   <book year="2000" id="2">
      <title>Data on the Web</title>
      <author>
         <last>Abiteboul</last>
         <first>Serge</first>
      </author>
      <author>
         <last>Buneman</last>
         <first>Peter</first>
      </author>
      <author>
         <last>Suciu</last>
         <first>Dan</first>
      </author>
      <price>39.95</price>
   </book>
   <book year="1999" bid="3">
      <title>The Economics of...</title
      <editor>
         <last>Gerbarg</last>
         <first>Darcy</first>
         <affiliation>CITI</affiliation
      </editor>
      <price>129.95</price>
   </book>
</bib>
```

Figure 1: Example of an XML fragment

Figure 2 shows the corresponding taDOM tree which represents the structure of the XML fragment. In order to construct the taDOM tree, we introduce three new node types: *document root*, *attribute root*, and *string*. These new node types only appear within the taDOM tree. Locks on the taDOM tree are always acquired transparently by a lock manager component. This proceeding guarantees that the behavior of the DOM API will not change from the users point of view.

The top of the taDOM tree consists of the document root, followed by the <*bib*> element—the outer element of the document. Attribute nodes are not directly connected to the element nodes. Instead an attribute root is connected to each element and organizes the attributes of the corresponding element as descendant nodes. In order to build a (frequently requested) *NamedNodeMap* containing all attributes of an element, only a single lock for shared access has to be acquired for the attribute root.

In the taDOM tree, XML elements are represented by element nodes and XML attributes by attribute nodes. The text within elements is handled by text nodes whereas the actual values of text nodes or attribute nodes are stored in string nodes. The existence of a string node is hidden to the taDOM API users. The users read or update values as usual by method invocations on the attribute or text nodes, but reading or updating a node value causes an implicit lock acquisition on the corresponding string node.

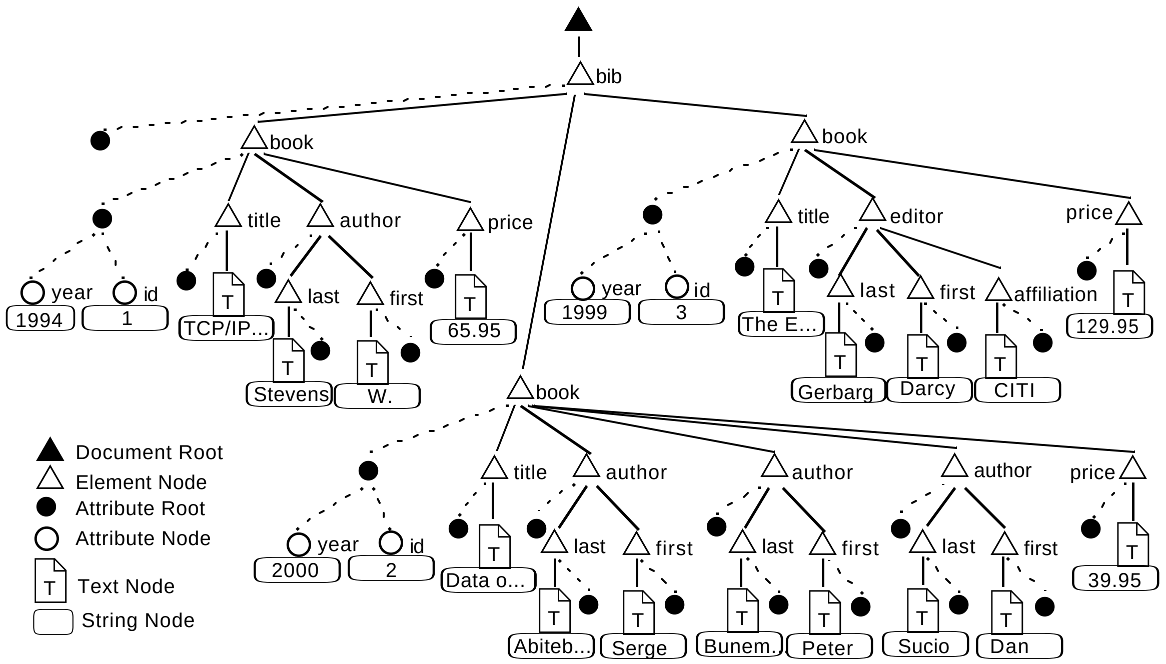


Figure 2: A sample taDOM tree

3 Synchronization of Document Access

Accessing a node requires the acquisition of a lock for the corresponding node. This procedure is described in Section 3.1. Lock granularity and lock escalation management is considered in Section 3.2. In order to synchronize the traversal of an XML document we present so-called *navigation locks* in Section 3.3. Synchronization of query access methods by means of logical locks is explained in Section 3.4.

3.1 Node Locks

While traversing or modifying an XML document, a transaction has to acquire a lock for each node before accessing it. The currently accessed node will be called *working node* in the following. Its lock mode depends on the type of access to be performed. Since the DOM API not only supports navigation starting from the document root, but also allows jumps „out of the blue" to an arbitrary node within the document, locks must be acquired in either case for the sequence of nodes from the document root downwards to the working node. Figure 3 gives an overview of the compatibility matrix for the lock modes defined as an extension of the well-known DAG locking ([5]).

	-	IX	NR	CX	LR	SR	U	X
IX	+	+	+	+	+	-	-	-
NR	+	+	+	+	+	+	-	-
CX	+	+	+	+	-	-	-	-
LR	+	+	+	-	+	+	-	-
SR	+	-	+	-	+	+	-	-
U	+	+	+	+	+	+	-	-
X	+	-	-	-	-	-	-	-

Figure 3: Compatibility matrix for lock modes

- The NR mode (node read) is requested for read access to the working node. Therefore, for each node from the document root downwards to the working node, a NR lock has to be acquired.
- The LR mode (level read) locks an entire level of nodes in the taDOM tree for shared access. For example, an LR lock for an attribute root, locks all attributes for the *getAttributes()* method.
- To lock an entire sub-tree of nodes (specified by the working node as the sub-tree root) in read mode, the SR mode (sub-tree read) is requested for the working node.
- To modify the working node (updating contents or deleting the entire node with its sub-tree), an X lock (exclusive) is acquired for the working node. This implies the request of a CX lock (child exclusive) for its parent node and an IX lock (intent exclusive) for each further node to the document root.
- A U lock supports read with (potential) subsequent write access and prevents granting further R locks for the same object. A U lock can be converted to an X lock after the release of the existing read locks.

3.2 Tunable node lock granularity and lock escalation

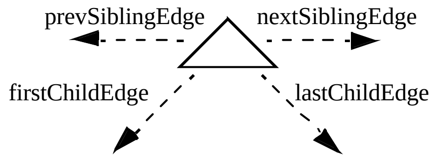
Both the SR lock, which locks an entire sub-tree in the taDOM tree in shared mode, and the X lock, which locks an entire sub-tree exclusively, enable tunable lock granularity and lock escalation.

To tune the lock granularity of nodes for each document, the parameter *lock depth* is introduced. Lock depth describes the lock granularity by means of the number of node levels (from document root) on which locks are to be held. If a lock is requested for a node whose path length to the document root is greater than the lock depth, only a SR lock for the ancestor node belonging to the lock-depth level is requested. This allows a transaction to traverse a large document fragment in read mode without acquiring any additional node locks. In the same way, several X locks can be merged to a single X lock at a higher level.

In a similar way, lock escalation can be realized. To tune the lock escalation, we introduce two parameters, the *escalation threshold* and the *escalation depth*. A lock manager component scans the taDOM tree at prespecified intervals. If the manager detects a sub-tree in which the number of locked nodes of a transaction exceeds the percentage threshold value defined by the escalation threshold, the locks held are replaced with a suitable lock (SR or X lock) at the sub-tree root, if possible. The escalation depth defines the maximal sub-tree depth from the leaves of a taDOM tree to the scanned sub-tree root.

3.3 Navigation Locks

In addition to the access of nodes, the DOM API also provides for methods which enable the traversal of XML documents. A sequence of navigational method calls or modification (IUD) operations—starting at a known node within the taDOM tree—must always yield the same sequence of result nodes within a transaction. Hence, a path of nodes evaluated by a transaction must be protected against modifications of concurrent transactions. For this reason, we introduce *virtual navigation edges* and corresponding *navigation locks* for element and text nodes within the taDOM tree.



	-	ER	EU	EX
ER	+	+	-	-
EU	+	+	-	-
EX	+	-	-	-

Figure 4: Virtual navigation edges and compatibility matrix for navigation locks

While navigating through an XML document and traversing the navigation edges, a transaction has to request a lock for each edge. Corresponding to the well-known R/U/X locks for relational records or tables ([5]) we offer the ER, EU, and EX lock modes:

- An ER lock (edge read) is acquired, before an edge is traversed in read mode. For example, such an acquisition may happen by calling the *getNextSibling()* or *getFirstChildNode()* method.
- An EX lock (edge exclusive) is requested, before an edge is modified. It may occur when nodes are deleted or inserted.
- The EU lock for edge updates eases the starvation problem of write transactions (see Section 3.1).

3.4 Logical Locks

The DOM API also provides for methods to get elements by their tag names or by an Id attribute. The existence of an attribute specified by its name can be checked. Read access to XML documents using these methods requires prevention of the phantom anomaly.

We can prevent phantoms in our hierarchical data model by using coarser lock granules. But coarse lock granules can enhance lock conflicts or deadlock situations. Especially, the *getElementById(...)* method, which can be only invoked for the document root, would always cause an undesirable read lock for the entire document.

To extend our synchronization concept by logical locks we introduce three lock tables which maintain the requested locks as illustrated in Figure 5. Lock compatibility inside each table is handled by the conventional R/U/X compatibility matrix.

LocksTagnameQuery

TAID	lock	scopeNID	tagname
...	R/U/X	...	...

LocksAttributeQuery

TAID	lock	NID	attribute
...	R/U/X	...	...

LocksIDQuery

TAID	lock	ID
...	R/U/X	...

Figure 5: Lock tables for logical locks

- Table *LocksTagnameQuery* maintains the logical locks for element name requests by execution of the method *getElementsByTagName(...)*.
- Table *LocksAttributeQuery* synchronizes the execution of *hasAttribute(...)* queries. This requires the acquisition of an R lock for the corresponding queried attribute name before getting the result.
- Table *LocksIDQuery* maintains logical locks for ID attributes queried by the method *getElementById(...)*. Since this is only supported at the document level, this table does not store a scope node ID.

Insertion of new nodes requires the acquisition of corresponding X locks in the logical lock tables. Insertion of sub-trees (specified by the root node) in the taDOM tree requires the traversal of the entire sub-tree and lock acquisition for each element or attribute found in the sub-tree.

4 Related Work

So far, there are relatively few publications on synchronizing query and update access to XML documents. In [9], transactional synchronization for XML data in client/server web applications is realized by means of a check-in/check-out concept. In order to detect synchronization conflicts, read and write sets are validated against the original document stored in the server. Lam et al. [6] discuss synchronization for mobile XML data with respect to handheld clients which query XML fragments by XQL. The authors present an efficient mechanism to determine whether two XQL expressions overlap. Hehner et al. [8] discuss, also based on the DOM API, several isolation protocols (both pessimistic and optimistic) for XML databases. Node locks are not acquired in a hierarchical context, and lock granularity is fixed for each protocol. XMLTM, an efficient transaction management for XML documents, is presented in [7] where XML documents are stored in relational databases having special XML extensions and an additional transaction manager. Both [7] and [8] do not treat attribute nodes in a special way or discuss varying lock granules.

5 Conclusion and Future Work

Synchronizing concurrent access to XML documents is an important practical problem. So far, only few extensions exist in commercial database systems which poorly support concurrent document processing primarily due to coarse-grained locking. On the other hand, concurrent XML processing has not received much attention in the scientific world yet.

We have first presented the taDOM tree, an extended data model of the DOM representation of XML documents. The taDOM tree provides lock acquisition at very fine-grained levels. Based on the taDOM tree, we have introduced a concept of combined node locks, navigation locks, and logical locks to synchronize concurrent access tailored to the DOM API. The combination of the three lock types also offers rich options to enhance transaction concurrency, for example, tunable lock granularity and lock escalation in order to increase throughput. Whether our concept is expressive enough to synchronize non-navigational APIs (like XQuery which works on node collections) is part of future work.

Our next step is a system implementation in order to evaluate the synchronization concept for different workloads. We plan to explore the effects on concurrency and performance for varying lock granularities and lock escalation thresholds. Future steps include the conjunction of XML synchronization with native XML storage methods and joined transactional processing of XML and relational data.

References

[1] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (2nd Ed.). W3C Recommendation (Oct. 2000)

[2] T. Härder and A. Reuter. Principles of Transaction-Oriented Database Recovery. ACM Computing Surveys 15(4), (Dec. 1983) 287-317

[3] World Wide Web Consortium. Document Object Model (DOM) Level 2 Core Specification, Version 1. W3C Recommendation (Nov. 2000)

[4] World Wide Web Consortium. XML Query Use Cases. W3C Working Draft (Nov. 2002)

[5] J. Gray and A. Reuter. Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers Inc. (1993)

[6] F. Lam, N. Lam, and R. Wong. Efficient Synchronization for Mobile XML Data. Proc. 11th Int. Conference on Information and Knowledge Management, McLean, Virginia, USA (Nov. 2002) 153-160

[7] T. Grabs, K. Böhm and H.-J. Schek. XMLTM: Efficient Transaction Management for XML Documents. Proc. 11th Int. Conference on Information and Knowledge Management, McLean, Virginia, USA (Nov. 2002) 142-152

[8] S. Helmer, C.-C. Kanne, and G. Moerkotte. Isolation in XML Bases. Int. Report, Faculty of Mathematics and Comp. Science, Univ. of Mannheim, Germany, Volume 15 (2001)

[9] S. Böttcher and A. Türling. Transaction Synchronization for XML Data in Client-Server Web Applications. Proc. Workshop on Web Databases, Annual Conf. of the German and Austrian Computer Societies, Vienna (2001) 388-395

[10] D. Florescu, D. Kossmann. A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database. Rapport de Recherche, No. 3680, INRIA, Rocquencourt, France (1999)

[11] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang. Storing and Querying Ordered XML Using a Relational Database System. Proc. ACM SIGMOD, Madison, Wisconsin (June 2002) 204-215

[12] M. Yoshikawa and T. Amagasa. XRel—A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases. ACM Transactions on Internet Technology 1(1), (Aug. 2001) 110-141