A LOCKING PROTOCOL FOR DOM API ON XML DOCUMENTS

Yin-Fu Huang and Mu-Liang Guo
Institute of Electronic and Information Engineering, National Yunlin University of Science and Technology, 123 University Road, Section 3, Touliu, Yunlin, Taiwan 640, R.O.C.

Keywords: XML databases, concurrency control, DOM API, locking protocols.

Abstract: In this paper, we developed a new DOM API locking protocol (DLP) that adopts the DOM structure for locking. In order to enhance the concurrency and system performance, we studied operation conflicts more detailedly. The proposed DLP supports more update operations than the others, and does not imply more locking costs. Finally, we conducted several experiments to compare with the others and to observe the DLP performance under different workload parameters.

1 INTRODUCTION

With the widespread use of the eXtensible Markup Language (XML), more and more applications store, query, and manipulate their XML documents in XML database management systems. Therefore, the proper scheduling of concurrent queries and updates becomes an important issue, which would affect the system performance directly. Concurrency control is a mechanism used to schedule concurrent transactions such that the correctness of accessed data is guaranteed. Serializability (Gray, 1993) is a criteria for the correctness, which claims the processing results of transactions should be equal to the one produced by some serial execution. To ensure serializability, each transaction accessing data items must follow the rules prescribed by the concurrency control mechanism.

Among different methods, locked-based protocols are most popularly used in concurrency control schemes since they are simple to implement and ensure serializability. Although an XML document is presented with the tree structure and a query has the navigation property, tree and graph-based locking protocols are not suitable for XML documents. On the contrary, the two-phase locking protocol (2PL) is the most suitable for XML documents. Like other locked-based protocols, 2PL uses locks to prevent conflicting transactions from modifying shared data items. However, in the 2PL scheme, a transaction only gets locks in the growing phase, and then releases locks in the shrinking phase.

In the past, several protocols in concurrency control on XML documents have been proposed. These protocols based on structures and corresponding interfaces could be classified into several types. The DataGuide protocols (Grabs, 2002 and Pleshachkov, 2005) focus on XML-enabled DBMSs. An XML document stored in XML-enabled databases usually is not represented as a tree structure physically. Another alternative focuses on native XML databases. An XML document in native XML databases is stored as a tree structure. Besides, based on different interfaces, protocols could be re-classified into XPath-typed (Choi, 2003a, Choi, 2003b, Dekeyser, 2004, Izadi, 2007, Jea, 2006, and Zhang, 2004) and DOM-typed (Haustein, 2004 and Helmer, 2004).

In the paper, we defined eight operations used to access an XML document, and then proposed a locking protocol based on a DOM interface. DOM API is a popular language for accessing and manipulating data in native XML databases. It translates an XML document into a DOM tree, and supports more complex behaviors than merely tree traversal.

The remainder of the paper is organized as follows. Section 2 describes an XML document and a set of access and modification operations for our protocol. In Section 3, we proposed the protocol DLP based on the strict two-phase locking. Then, several experiments were conducted to compare DLP with other protocols in Section 4. Finally, we make conclusions in Section 5.

Huang Y. and Guo M. (2008).
A LOCKING PROTOCOL FOR DOM API ON XML DOCUMENTS.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - DISI*, pages 105-110
DOI: 10.5220/0001675701050110

2 RELATED WORK

In principle, our research is based on the DOM API (i.e., Application Program Interface) which is supported by most native databases to access and manipulate XML documents. Although DOM API provides several interfaces to manipulate XML documents, we only focus on elements, attributes, and texts since they are the most important theme of data sharing. Similar to XML document trees, the Document Object Model (i.e., DOM) represents XML documents as trees, as shown in Figure 1.

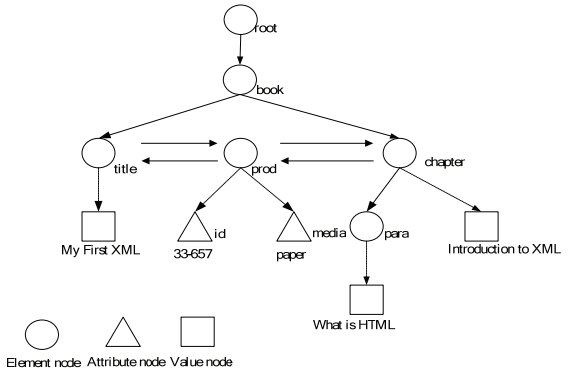

Figure 1: DOM tree representation.

Till now, there are two classifications in the previous researches on concurrency control manipulation with DOM API. One is XML Transaction Coordinator (Haustein, 2004) that provides a taDOM tree for storing XML documents, and proposed the taDOM protocol to ensure serializability. The other one is Natix that proposed Doc2PL, Node2PL, NO2PL, and OO2PL protocols (Helmer, 2004) to ensure serializability. Since OO2PL has been verified to have the best performance, our research stretches OO2PL. Although OO2PL acquires locks on the pointers of nodes (i.e., the first child, the last child, the previous sibling, and the next sibling), OO2PL only classifies operations into observer and mutator ones. In order to enhance the concurrency degree, our protocol distinguishes operations more detailedly.

3 DLP

3.1 Operation Conflicts

The operations in our protocol consist of eight types: ***R***, ***N***, ***IB***, ***AP***, ***UP***, ***RN***, ***RM***, and ***RP*** standing for Read, Navigate, Insert-Before, Append, Update, Rename, Remove, Replace, respectively. ***R*** and ***N*** are both read operations, but ***R*** is for the manipulation of nodes and ***N*** is for the navigation of paths.

We define a *transaction* ***T*** as a sequence of DOM API operations. Operation conflicts may occur when the operations from different transactions are interleaved with each other, thereby producing incorrect results. The criterion of correctness is based on the serializability of concurrent transactions. In order to analyze the conflicts between operations, we classify operations into content operations and structural operations. Content operations consisting of ***R***, ***UP***, and ***RN*** denote the ones which manipulate data values at nodes. Structural operations consisting of ***N***, ***IB***, ***AP***, ***RM***, and ***RP*** denote the ones which navigate or modify the structure of a DOM tree. The structural operations get involved in the pointers within a node.

Different from that ***R*** reads the content of a target node (i.e., node name or text value), ***N*** reads the pointer of each node (i.e., the first child or the next sibling, et al.) along the path specified by a transaction. Next, ***RM*** (or ***RP***) is similar to ***N***, but it modifies the pointer to the target node into nil (or the pointer to the replacing node). Finally, ***IB*** (or ***AP***) modifies the previous sibling pointer (or the last child pointer) of the target node into the pointer to the new node. However, not only the relevant pointer of the target node but also the pointers of related nodes should be modified together.

Basically, the two kinds of operations would not conflict with each other, since content operations only manipulate node values, whereas structural operations only deal with the DOM structure. However, always a transaction executing a content operation has to use structural operation ***N*** to reach the target node. Thus, while these two kinds of operations work on the same target node, the involved structural operations would conflict with themselves.

As mentioned above, we summarize the operation conflicts in Table 1 and 2. Within the matrix, symbols "○" and "×" denote the concurrent operations are OK and in conflict, respectively. Beside symbols "○" and "×", we also use symbol "△" to denote the concurrent operations are in conflict in some situation.

Table 1: Conflict matrix of content operations.

	R	UP	RN
R	○	○	×
UP	○	×	○
RN	×	○	×

Table 2: Conflict matrix of structural operations.

	N	IB	AP	RM	RP
N	○	•	•	×	×
IB	•	×	○	×	×
AP	•	○	×	×	×
RM	×	×	×	×	×
RP	×	×	×	×	×

3.2 DLP

In this section, we proposed a DOM API locking protocol (DLP) to ensure the serializability of concurrent transactions. DLP is with seven lock modes including S-, SC-, W-, N-, NC-, P-, and X-locks. While an operation intends to work on a node, it must acquire an appropriate lock. In general, S-lock, W-lock, and N-lock are acquired by content operations. Next, SC-lock and NC-lock are also acquired by content operations, but these two locks are only used for the nodes with predicates in a path. Finally, P-lock and X-lock are acquired by structural operations. Since content operations would not conflict with structural operations, there is no compatibility problem between the two types of lock modes; i.e., S-, SC-, W-, N- and NC- locks on nodes, and P- and X-locks on pointers. The seven lock modes used in DLP would be explained as follows.

(1) **S-lock**: S-lock is a shared lock designed for ***R***. S-lock is compatible with itself (for ***R***), SC-lock (for ***R*** with predicates), and W-lock (for ***UP***), but incompatible with N-lock (for ***RN***) and NC-lock (for ***RN*** with predicates).

(2) **SC-lock**: SC-lock is also a shared lock designed for ***R***, but is only used for the nodes with predicates in a path. Since the predicates with attributes or node values specified in the path are also parts of the path, the attributes or node values should not be modified by ***UP***. Thus, SC-lock is compatible with itself (for ***R*** with predicates), but incompatible with W-lock (for ***UP***), N-lock (for ***RN***), and NC-lock (for ***RN*** with predicates).

(3) **W-lock**: W-lock is a lock designed for ***UP***. Modifying attributes or node values is nothing to do with modifying node names, unless predicates are specified in the path for ***RN***. Thus, W-lock is compatible with N-lock (for ***RN***), but incompatible with itself and NC-lock (for ***RN*** with predicates).

(4) **N-lock**: N-lock is a lock designed for RN. N-lock is incompatible with itself and NC-lock (for ***RN*** with predicates).

(5) **NC-lock**: NC-lock is an exclusive lock designed for ***RN***, but is only used for the nodes with predicates in a path. Since the predicates with attributes or node values specified in the path are also parts of the path, the attributes or node values should not be modified by ***UP***. NC-lock is incompatible with itself.

(6) **P-lock**: P-lock is a shared lock designed for ***N***. Thus, P-locks are issued along the specified path from the root to the target node. Basically, P-lock is similar to S-lock, but the difference between them is P-lock for pointers and S-lock for contents. P-lock is compatible with itself, but incompatible with X-lock, since X-lock is for pointer modification.

(7) **X-lock**: All structural operations except ***N*** would modify the structure of a DOM tree. Whatever the operations are, the operations working on the same pointer would conflict with each other. Thus, X-lock is an exclusive lock designed for these operations. While these operations attempt to modify relevant pointers, they would issue X-locks on these pointers. X-lock is incompatible with itself.

As shown in Table 3, we summarize the lock compatibility in DLP. Within the matrix, symbols "○", "×", and "-" denote locks are compatible with, incompatible with, and not related to each other, respectively.

Table 3: Lock compatibility matrix.

	S	SC	W	N	NC	P	X
S	○	○	○	×	×	-	-
SC	○	○	×	×	×	-	-
W	○	×	×	○	×	-	-
N	×	×	○	×	×	-	-
NC	×	×	×	×	×	-	-
P	-	-	-	-	-	○	×
X	-	-	-	-	-	×	×

As an example shown in Figure 2, we observe the locks on a DOM tree, which are issued by three transactions following DLP. First, T_1 intends to read node n6 with path /n1/n2/n6='t1'. Since ***R*** must ensure the node value of node n6, it issues SC-lock on node n6. Besides, T_1 also issues P-locks for the down-link pointers of all preceding nodes from the root. Simultaneously, T_2 intends to update node n6 also along the same path. Thus, T_2 issues P-locks for the same path and W-lock on node n6. As a result, T_2 is declined because of existing SC-lock already issued by T_1. Meanwhile, T_3 attempts to remove node n3. Thus, T_3 issues X-locks for all relevant pointers to node n3. Since there is no lock

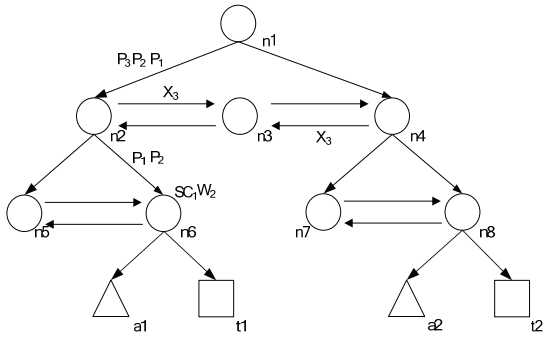

Figure 2: Locks issued on a DOM tree.

incompatibility, T_3 would remove node n3 successfully.

Besides, a possible schedule for the example following DLP is illustrated as shown in Figure 3. There, P- and X-lock with a subscript denote on what pointer they are issued; i.e., F for First child, L for Last child, P for Previous sibling, and N for Next sibling. T_1 and T_2 cannot be executed concurrently after step 7, since SC-lock on node n6 issued by T_1 would exclude W-lock on node n6 issued by T_2. T_2 can acquire W-lock on node n6 and continue execution only after T_1 releases SC-lock on node n6 at step 11.

	T_1	T_2	T_3
1	P_F-Lock(n1)		
2		P_F-Lock(n1)	
3			P_F-Lock(n1)
4	P_L-Lock(n2)		
5		P_L-Lock(n2)	
6			P_N-Lock(n2)
7	SC-Lock(n6)		
8			Upgrade X_N-Lock(n2)
9	Read (n6)		
10			X_F-Lock(n4)
11	Unlock(n6)		
12		W-Lock(n6)	
13			Remove (n3)
14	Unlock(n2)		
15		Update (n6)	
16			Unlock(n4)
17	Unlock(n1)		
18		Unlock(n6)	
19			Unlock(n2)
20		Unlock(n2)	
21			Unlock(n1)
22			

Figure 3: Schedule for the example.

4 PERFORMANCE EVALUATION

4.1 Simulation Model

In order to compare the performances among DLP, OO2PL, and taDOM, a simulation model is proposed, as shown in Figure 4. The model programmed with General Purpose Simulation System (i.e., ***GPSS World***) consists of a transaction generator, several queues, a concurrency control mechanism to schedule transactions under these three protocols, and an object mechanism to access document nodes.

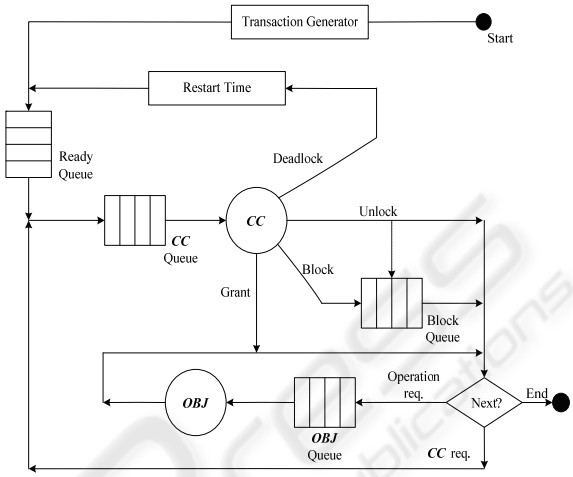

Figure 4: Simulation model.

4.2 Experiments

The workload parameters used in the simulation can be classified into two types (i.e., transaction behavior and system environment), as shown in Table 4. The transaction behavior related parameters include 1) number of operations in a transaction (i.e., NOT), 2) percentages of exclusive operations (i.e., PEO), 3) transaction arrival time (i.e., TAT), and 4) restart time (i.e., RT). The system environment related parameters include 1) document sizes (i.e., DS) and 2) degree of multiprogramming (i.e., DMP). Finally, Table 5 shows all operational time cost. In the experiments, OO2PL and DLP have the same time cost, whereas taDOM has 1.5~2 times cost of them. The reason is that since the document structure in taDOM is different from the original DOM structure, it takes more time to execute operations.

Table 4: Workload parameters.

transaction behavior	
NOT	10~130
PEO	25%, 50%,
TAT	20 (μs)
RT	3 (μs)
system environment	
DS(in nodes)	781,3906, 9531
DMP	1~18

Table 5: Operational time cost.

R	4(μs)	N	2(μs)	RM	6(μs)
UP	5(μs)	IB	8(μs)	RP	8(μs)
RN	5(μs)	AP	8(μs)	Lock	1(μs)

4.2.1 Varying Number of Operations in a Transaction

The experiment is to evaluate the performances of three protocols by varying the number of operations in a transaction. The workload parameters particularly specified in the experiment are 1) PEO 50%, 2) DS 781, and 3) DMP 10.

As shown in Figure 5, the response time rises definitely when the number of operations in a transaction increases, but DLP has the minimum response time among them. Besides, we also found that the curve slope of taDOM is sharper than the other two, due to its more operational time cost. As a result, DLP is the superior one among three protocols in the throughput as shown in Figure 6.

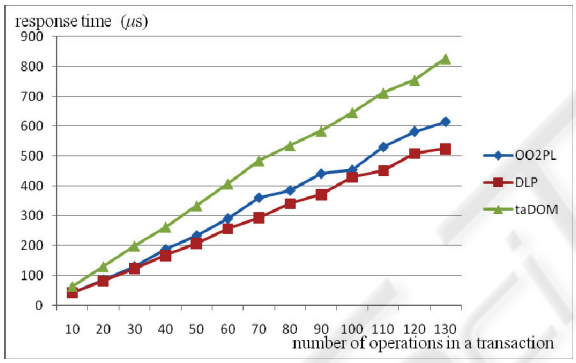

Figure 5: Response time vs. NOT.

As shown in Figure 7, the conflict ratio around the scale 70 is almost the highest for all three protocols. As increasing the number of operations in a transaction before reaching 70, the number of target nodes would increases, and this makes the conflict ratio higher. However, after reaching 70, the more target nodes are, the more the paths are in a transaction. Thus, the conflict ratio would not increase, and even somewhat decrease. Surprisingly, OO2PL has higher conflict ratio than the other two, since DLP and taDOM provide more lock modes than OO2PL.

4.2.2 Varying Degree of Multiprogramming

The experiment is to evaluate the performances of three protocols by varying the degree of multiprogramming. The workload parameters

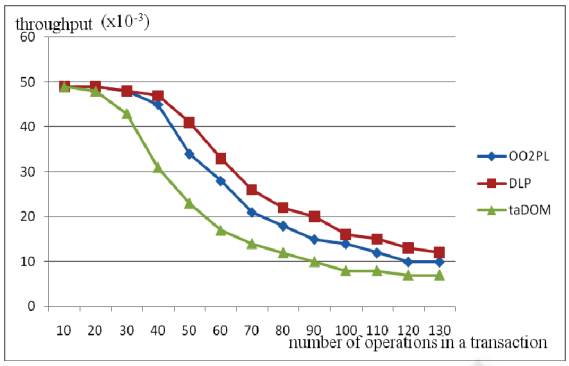

Figure 6: Throughput vs. NOT.

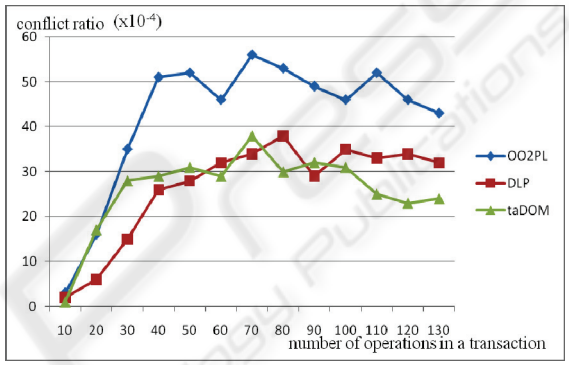

Figure 7: Conflict ratio vs. NOT.

multiprogramming. The workload parameters particularly specified in the experiment are 1) NOT 50, 2) PEO 50%, and 3) DS 781.

As shown in Figure 8, although the response time of OO2PL and taDOM increases slightly when the degree of multiprogramming increases, the changes are not obvious. Nevertheless, for the response time and the throughput as shown in Figure 8 and Figure 9, DLP is still the best one among them. Similar to Experiment 1, for the conflict ratio as shown in Figure 10, OO2PL has higher conflict ratio than the other two.

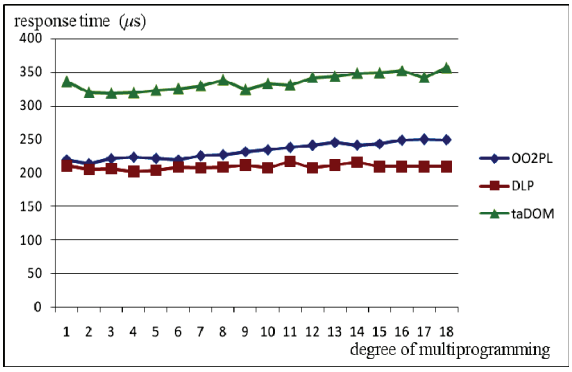

Figure 8: Response time vs. DMP.

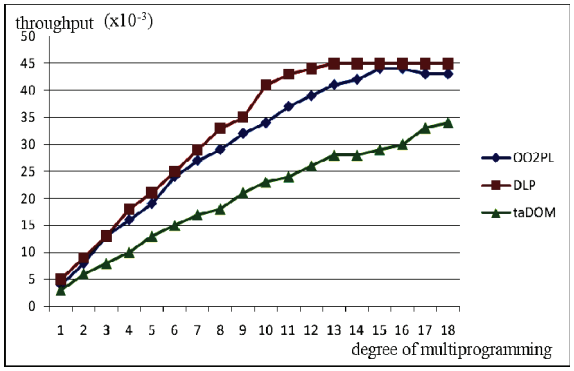

Figure 9: Throughput vs. DMP.

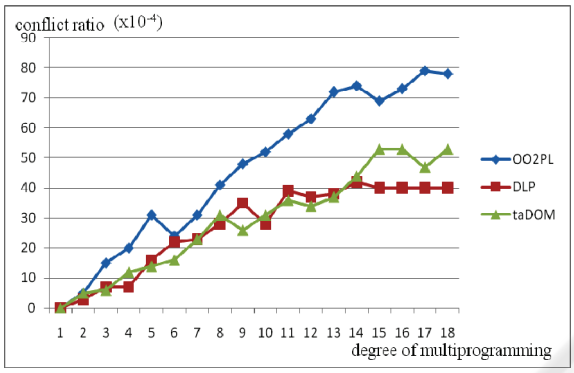

Figure 10: Conflict ratio vs. DMP.

5 CONCLUSIONS

In this paper, we proposed a locking protocol for DOM API, called DLP, to reduce the conflict ratio and then enhance the system throughput. In order to achieve the goals, we analyzed operation conflicts detailedly. The proposed DLP supports more update operations than the others. Furthermore, DLP also supports the specified predicate in the path, and does not imply more locking costs. To evaluate DLP, we conducted several experiments to compare with the others and to observe the DLP performance under different workload parameters. From the experimental results, we found that DLP has better performances than the others.

REFERENCES

Choi, E. H. and Kanai, T., 2003, XPath-based concurrency control for XML data, *Proc. the 14th Data Engineering Workshop*, pp. 302-313.

Choi, Y. and Moon, S., 2003, Lightweight multigranularity locking for transaction management in XML database systems, *Journal of Systems and Software*, Vol. 78, No. 1, pp. 37-46.

Dekeyser, S. and Hidders, J., 2004, A transaction model for XML databases, *World Wide Web: Internet and Web Information Systems*, Springer, Vol. 7, No. 1, pp. 29–57.

Grabs, T., Bohm, K., and Schek, H., 2002, XMLTM: efficient transaction management for XML documents, *Proc. the ACM International Conference on Information and Knowledge Management*, pp. 142-152.

Gray, J. and Reuter, A., 1993, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann.

Haustein, M. and Härder, T., 2004, Adjustable transaction isolation in XML database management systems, *Proc. the 2nd International XML Database Symposium (XSym)*, LNCS 3186, Springer, pp. 173-188.

Helmer, S., Kanne, C. C., and Moerkotte G., 2004, Evaluating lock-based protocols for cooperation on XML documents, *SIGMOD Record*, Vol. 33, No. 1, pp. 58-63.

Izadi, K., Asadi, F., and Haghjoo, M. S., 2007, XPLC: a novel protocol for concurrency control in XML databases, *IEEE/ACS International Conference on Computer Systems and Applications*, pp. 450-453.

Jea, K. F. and Chen, S. Y., 2006, A high concurrency XPath-based locking protocol for XML databases, *Information and Software Technology*, Vol. 48, No. 8, pp. 708-716.

Pleshachkov, P., Chardin, P., 2005, and Kuznetsov S., XDGL: XPath-based concurrency control protocol for XML data, *Lecture Notes in Computer Science*, Vol. 3657, pp. 145-146.

Zhang, W., Liu, D., and Sun, W., 2004, XR-lock: locking XML data for efficient concurrency control, *Proc. the 5th World Congress on Intelligent Control and Automation*, pp. 3921-3925.

Document Object Model, http://www.w3.org/DOM/.

Extensible Markup Language, http://www.w3.org/XML/.

GPSS World, http://www.minutemansoftware.com/.

Natix, http://pi3.informatik.uni-mannheim.de/~moer/natix.html.