A GRAPH-BASED CONCURRENcy CONTROL PROTOCOL FOR XML METADATA KNOWLEDGE BASES

TSUI-PING CHANG\textsuperscript{1} AND CHUN-HSIU YEH\textsuperscript{2}

\textsuperscript{1}Department of Information Technology
Ling Tung University
No. 1, Lingtung Rd., Taichung 408, Taiwan
apple@teemail.ltu.edu.tw

\textsuperscript{2}Department of Information Managements
Chung Chou Institute of Technology
No. 6, Lane 2, Sec. 3, Shanjiao Rd., Yuanlin Township, Changhua 510, Taiwan
chyeh@dragon.ccut.edu.tw

Received December 2009; revised March 2010

ABSTRACT. Providing efficient access for XML APIs in XML metadata knowledge bases (XMKBs) is crucial, since XML is widely used to integrate data in XMKBs. In XMKBs, the types, contents and locations of heterogeneous data are illustrated in XML documents to provide a uniform interface for users to manipulate this data. As users update and query information about this heterogeneous data, the XML documents are manipulated by XML APIs (i.e., DOM APIs) to ensure consistency. Concurrency control protocols can ensure a better performance of the DOM APIs to manipulate XML documents. However, the existing protocols (i.e., traditional locking protocols and XML-based protocols) are not sufficiently adequate for DOM APIs in XMKBs. Traditional graph-based locking protocols cannot deal with new DOM operations, and the existing XML-based protocols treat XML documents as trees, with no regard for the ID/IDREF(s) used in them. This paper proposes a new XML graph-based concurrency control protocol, tailored for XMKBs, named XGP. The novel feature of XGP is to analyze the DOM APIs on XML document graphical structures to increase the transactional performance of XMKBs. Our simulated results show that our XGP outperforms other concurrency control protocols in terms of higher throughput.

Keywords: Concurrency control, DOM, XML graph, XML metadata knowledge bases

1. Introduction. With the growth of the Internet, both the number and size of data sources available for public access is rapidly increasing. Thus, the need to combine data from different autonomous and heterogeneous data sources to provide public access to the Internet has become important [5,6,8,9,17,19]. However, most enterprises store data in relational database systems because of the reliability, scalability and performance associated with these systems. In addition, since many web-based systems publish their data using XML [4], a great deal of interesting and useful data can be found in valid and well-formed XML documents [4]. As a result, building an XML metadata knowledge base (namely XMKB) [5,6,8,9] which can provide unified access to diverse data sources is very desirable for linking data held in relational databases and XML documents.

Many XMKBs [5,6,8,9] are introduced to maintain the data source information (e.g., names, contents, types and locations) and meta-information about the relationship between paths of data sources. This is because XML has become the standard format to exchange information over the Internet, and it is suitable as metadata to integrate heterogeneous data. Thus, an XMKB contains multiple XML documents to describe the...
contents, types, locations and meta-information of heterogeneous data. These XML documents are updated when any data source is added or removed by users in XMKBs. Therefore, an XMKB must produce a uniform interface over these XML documents and provide the required APIs (e.g., DOM APIs [1]) to manipulate them in a uniform way.

In XMKBs, the XML documents exhibit a graphical structure and can be manipulated by DOM APIs as these documents are updated. In these documents, XML elements, attributes and texts describe the data types, contents and sources, while the parent/child, sibling and ID/IDREF(s) relationships state the association among them. Additionally, DOM [1] defines several interfaces such as element, characterdata and node interfaces, to access the elements, attributes and texts in XML documents. The element interface manipulates the attributes of the elements in XML documents, while the characterdata interface updates the texts of the elements or attributes. In addition, the node interface treats the elements, texts and attributes in XML documents as nodes and defines operations to access them. Through these interfaces, users can update the contents or structures of XML documents in XMKBs as the data types, contents or associations are changed. However, when more and more modifications and queries are performed in an XML document, it is crucial to provide efficient and correct access for DOM APIs in XMKBs.

Concurrency control [19] is one of the most important techniques providing efficient access in database systems. It improves system performance by concurrently executing multiple transactions. To this end, specific traditional graph-based locking protocols, DDG [7] may be considered for XMKBs, since they model data items in databases as graphs. In this protocol, a transaction can initially lock any data item. To lock any additional data items, the transaction must hold locks on the majority of the parents of the desired data item. Unfortunately, traditional locking protocols based solely on read/write operations are generally unsuitable for XMKBs, since they cannot deal with new DOM operations introduced in XML to manipulate document structures and obtain relationships among heterogeneous data. On the other hand, several XML-based concurrency control protocols [10-16] (namely XML protocols) are proposed to allow more DOM operations to be executed simultaneously by a rich set of locking models. However, these XML protocols are not suitable for the enhancement of access performance in XMKBs, since they treat documents as trees without regard to the ID/IDREF(s) used in them. Thus, there is a need for a new concurrency control protocol tailored for DOM operations in XMKBs.

The main contributions of our work are summarized as follows:

(1). We propose a new graph-based concurrency control protocol, XGP, tailored for XMKBs to enhance the performance of the system. Existing XMKBs enable users to query and update their XML documents when they change the content, type and locations of heterogeneous data. However, in these XMKBs, none of the proposed concurrency control protocol is tailored for XMKBs, and this degrades the performance of the system. Our XGP can enable multiple users to query and update these XML documents, and thus, enhance the performance of existing XMKBs.

(2). The graph structure of an XML document for concurrency control protocols is analyzed in this paper. To the best of our knowledge, no research considers the ID/IDREFs in XML documents, which result in them exhibiting a graph structure. Our analysis can be used in other research to design concurrency control protocols.

This paper is organized as follows. Section 2 describes the related works including the existing XMKBs, their XML documents and the DOM APIs which are used in XMKBs. Section 3 analyzes the DOM APIs on XML document graphical structures. Section 4 proposes a new XML graph-based concurrency control protocol, XGP, and proves the
correctness of the XGP schedules. Section 5 compares our XGP with the traditional graph-based concurrency control protocols and the XML protocols, while Section 6 shows the results of our simulation. Finally, Section 7 concludes this study.

2. Preliminaries. In this section, the existing XMKBs are firstly illustrated in Subsection 2.1. The XML documents in XMKBs are described in Subsection 2.2. The DOM APIs which are supported in XMKBs are then presented in Subsection 2.3. The classification of DOM APIs, based on XML document graphs, is described in Subsection 2.4, while the symbols used in this paper are defined in Subsection 2.5.

2.1. Existing XMKBs. There are several well-known research projects and prototypes, such as Mix [6,9] and SISSD [5], which use XML as the metadata to integrate data from heterogenous sources. Mix uses XML as the data model to integrate heterogenous data and an XML query language (namely XMAS), which is developed and used as a view definition language. Also, a distributed database XML metadata interface (namely DDXMI [9]) is built to describe the XML path information (a path for each node starting from the root) and semantic information about the elements and attributes of XML. Therefore, all of the XML documents in Mix can be manipulated by the XML APIs (e.g., DOM APIs) when users update information of heterogenous data. However, in Mix, concurrency control mechanisms are not considered and thus, the performance of the system cannot to be enhanced.

SISSD enables users to combine and query the data sources through a mediation layer. Such a layer is intended to establish and evolve an XMKB incrementally to assist the query processor in mediating between user queries and the distributed heterogeneous data sources, to translate such queries into sub-queries which fit each local data source, and to integrate the results. Additionally, SISSD has been built so that it generates a tool for a meta-user (who does the metadata integration) to manipulate the XML documents in the XMKB. As the contents, types and locations of heterogeneous data are updated by the meta-users, the elements, attributes and contents in these documents are changed. Like Mix, SISSD also does not consider any concurrency control mechanism to improve the system performance. Therefore, the query and update operations which are issued by users cannot to be executed concurrently, which results in a lower system performance in SISSD.

2.2. XML document graph. XML documents in XMKBs can be represented as graph structures, namely XML document graphs. XML elements, attributes and texts, used in XMKBs to describe the contents, types and sources of heterogeneous data, are mapped respectively to the element, attribute and text nodes in the corresponding XML document graph. Element and attribute nodes in an XML document graph are named by corresponding tag names and attribute names in the original XML document. Text nodes in an XML document graph are the result of the elements’ values and the attributes’ values in an XML document. Furthermore, the PC, IDREF and SB edges between nodes represent the parent/child, ID/IDREF(s) and sibling relationships respectively between XML elements, attributes and texts.

Figure 1 illustrates a Document Type Definition (DTD) [2] and its corresponding valid and well-formed XML document. A DTD defines the document structure with a list of legal elements and attributes, and a valid and well-formed XML document conforms to the rules of a DTD. Therefore, Figure 1(a) shows a DTD to describe the structure of an XML document, while Figure 1(b) shows that the XML document in XMKBs conforms to the rules of the DTD in Figure 1(a). In summary, Figure 1 is an XML document in an XMKB to describe the information about data source name, type and location. Consequently,
Figure 1. A DTD and valid and well-formed XML document

Figure 2. An XML document graph of Figure 1(b)

Figure 2 illustrates an XML document graph to describe the elements, attributes, texts and their relationships in the XML document in Figure 1.

In Figure 2, the XML document graph contains seven elements, (i.e., one DS_Information, two DS_Location, two name and two type), four attributes (i.e., two id and two number) and the texts of the elements and attributes. The DS_Location element has two attributes, namely, id and number which hold the information of the data source present in the system. Also, the DS_Location element has child elements, namely, name and type elements to describe the data source name and its type (relational database or XML document). In Figure 2, the element, attribute and text nodes are denoted by circles, rhombuses and rectangles, respectively. Also, the PC, IDREF and SB edges are denoted by the dotted, solid and double lines, respectively. It should be noted that, for simplicity, the symbols (e.g., a, b, c, etc.) are used to represent the element, attribute, and text nodes in Figure 2.

2.3. DOM APIs on XML document graphs. Several XMKBs use DOM APIs to manipulate XML documents, such as MIX [6] and SISSID [5]. DOM defines the logical structures of XML documents and provides a set of standardized operations to access nodes in XML document graphs. Therefore, programmers can use the operations (e.g., FirstChild, NextSibling, LastChild and PreviousSibling) to navigate nodes, the operations
Figure 3. A brief XML document graph from Figure 2

Table 1. Examples of DOM operations

<table>
<thead>
<tr>
<th>DOM operations</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node operations</td>
<td>FC, LC, NS, PS, ID, TN</td>
</tr>
<tr>
<td>Neighborhood operations</td>
<td>IB, AC</td>
</tr>
<tr>
<td>Sub-graph operations</td>
<td>RM, RC, SE</td>
</tr>
</tbody>
</table>

(e.g., `getElementById`, `getElementTagName` and `getSubElement`) to read nodes, and the operations (e.g., `InsertBefore`, `AppendChild`, `RemoveChild` and `ReplaceChild`) to manipulate nodes. A reference node \( x \) with an operation is indicated to navigate nodes. For example, the operation "\( x.\text{FirstChild} \)" navigates the first child node of \( x \). The nodes’ id or tag names are indicated to read nodes. For example, `getSubElement(x)` reads all of \( x \)’s descendant nodes. To modify nodes, a reference node \( x \), an \( x \)’s child node \( y \) and a new node \( k \), with an operation (e.g., `InsertBefore`, `AppendChild` and `ReplaceChild`) are indicated. The operation "\( x.\text{InsertBefore}(y, k) \)" inserts a new node \( k \) in front of \( y \), the operation "\( x.\text{AppendChild}(k) \)" appends a new last child node \( k \) of \( x \), while the operation "\( x.\text{ReplaceChild}(y, k) \)" removes all of the descendant nodes of \( y \) and replaces node \( y \) with a new node \( k \). Finally, to delete nodes, a reference node \( x \) and its child node \( y \) are indicated with an operation. For example, the operation "\( x.\text{RemoveChild}(y) \)" deletes node \( y \) and all of its descendant nodes.

2.4. Classifications. The DOM operations can be classified into node, neighborhood and sub-graph operations based on XML document graphs. Given a node \( x \) in an XML document graph, a node operation manipulates node \( x \), a neighborhood operation access node \( x \), \( x \)’s parent and sibling nodes, and a sub-graph operation operates node \( x \) and all of its descendant nodes. Firstly, as listed in Table 1, node operations include the operations (e.g., `FirstChild` (denoted by FC), `NextSibling` (NS), `LastChild` (LC), `PreviousSibling` (PS), `getElementTagName` (TN) and `getElementById` (ID)). Secondly, neighborhood operations include the operations (e.g., `InsertBefore` (IB) and `AppendChild` (AC)), and lastly, sub-graph operations include the operations (e.g., `getSubElement` (SE)), `RemoveChild` (RM) and `ReplaceChild` (RC)). Among these operations, FC, LC, NS and PS navigate a node’s first child, last child, next sibling and previous sibling, respectively. ID and TN read the nodes’ values through their attribute id or tag name. IB and AC insert new nodes into an XML document graph. RM and RC respectively delete and replace nodes, and SE reads all descendant nodes of a certain node.

Whether two DOM operations can be executed concurrently or not is decided based on the operations on document graphical structures. For example in Figure 3, the \( a.\text{SE}(e) \)
operation reads all $e$’s descendant nodes $b$, $c$, $d$, $f$ and $g$, and the $b$.IB$(c, k)$ operation inserts a new node $k$ before $c$. Operations $a$.SE$(e)$ and $b$.IB$(k, c)$ cannot be executed concurrently on the XML document graph in Figure 3, since via the ID/IDREF edge between nodes $e$ and $b$ (note that the ID/IDREF edge is indicated by the attribute node $e_1$), $a$.SE$(e)$ reads the nodes $b$, $c$ and $d$, which are updated by $b$.IB$(k, c)$.

2.5. Symbols. The symbols used in this paper are defined as follows. A transaction $T_i$ consists of a sequence of operations classified into node, neighborhood and sub-graph operations. Notation $C_{i,m}$ is used to indicate the node operated by an node operation $O_{i,m}$, which is the $m^{\text{th}}$ DOM operation in $T_i$. A neighborhood operation in $T_i$ is represented by $P_{i,m}.O_{i,m}(C_{i,m}, N_{i,m})$ and $C_{i,m}.O_{i,m}(N_{i,m})$, while a sub-graph operation is represented by $C_{i,m}.O_{i,m}(N_{i,m}, R_{i,m})$ or $C_{i,m}.O_{i,m}(R_{i,m})$. $R_{i,m}$ denotes an existing child node of $C_{i,m}$, $N_{i,m}$ denotes the new child node of $C_{i,m}$ to be inserted by $O_{i,m}$, and $P_{i,m}$ denotes the parent node of $C_{i,m}$. For any node operation $O_{i,m}$ (i.e., FC, LC, NS, PS, ID or TN), $C_{i,m}.O_{i,m}$ denotes that $C_{i,m}$ is read by $O_{i,m}$. $P_{i,m}.IB(C_{i,m}, N_{i,m})$ indicates that neighborhood operation IB inserts $N_{i,m}$ in front of $C_{i,m}$, and $C_{i,m}.AC(N_{i,m})$ denotes that AC appends $N_{i,m}$ to $C_{i,m}$’s last child node. $P_{i,m}.RC(C_{i,m}, N_{i,m})$ indicates that sub-graph operation RC replaces $N_{i,m}$ with $C_{i,m}$, respectively and $P_{i,m}.MC(C_{i,m})$ denotes that MC deletes $C_{i,m}$ and all of $C_{i,m}$’s descendant nodes, respectively. For example, in Figure 3, the node operation “$e$.LC” and the neighborhood operation “$e$.IB$(f, n_1)$” in transaction $T_1$ can be represented by “$C_{1,1}.O_{1,1}$” and “$P_{1,2}.O_{1,2}(C_{1,2}, N_{1,2})$”, respectively, where $C_{1,1}$ is node $e$, $O_{1,1}$ is node operation LC, $P_{1,2}$ is node $e$, $N_{1,2}$ is the new node $n_1$, and $C_{1,2}$ is node $f$, $O_{1,2}$ is neighborhood operation IB. As a result, $T_1 = (e.LC, e.IB(n_1, f))$.

3. XML Graphical Structure Analysis. In this section, the XML graphical structures are analyzed from the DOM operation perspective. The XML graph model is first defined, and then the differences between node, neighborhood and sub-graph operations on the XML graph are discussed. As a result, the conflict between two DOM operations on the XML graphs is analyzed.

3.1. XML graph model. In order to represent the structural information of the XML document graph, XML graphs are defined in Definition 3.1, and the features between nodes are defined in Definitions 3.2 and 3.3.

Definition 3.1. Let $G = (V, \{E_{IDREF} \cup E_{PC} \cup E_{SB}\})$ be the directed graph for the XML document, where $V$ is a set of nodes representing the elements, texts and attributes in the XML document. $E_{IDREF}$ is a set of ordered pairs of nodes in $V$ representing the ID/IDREF relationships between the nodes. $E_{PC}$ is a set of ordered pairs of nodes in $V$ representing the parent/child relationships between the nodes. $E_{SB}$ is a set of ordered pairs of nodes in $V$ representing the sibling relationships between the nodes.

Definition 3.2. Let $G_u = (\{u \cup V'\}, E')$ be the sub-graph of the XML graph $G$ where $u$ is a node in $V$; $V'$ is a set of nodes in $V$ connecting to node $u$ via some path; and $E'$ is a set of edges in $E_{IDREF}$, $E_{PC}$ and $E_{SB}$ connecting nodes in $\{u \cup V'\}$.

Definition 3.3. Let $N_u = (\{u \cup V'\}, E')$ be the neighborhood of the XML graph $G$ where $u$ is a node in $V$; $V'$ is a set of nodes in $V$ including $u$’s parent and sibling nodes; and $E'$ is a set of edges in $E_{IDREF}$, $E_{PC}$ and $E_{SB}$ connecting nodes in $\{u \cup V'\}$.

For example in Figure 3, the XML graph $G = (V, \{E_{IDREF} \cup E_{PC} \cup E_{SB}\})$, $V = \{a, b, c, d, e, f, g, b_1, b_2, e_1, e_2, t_1, t_2, t_3, t_4, t_5\}$, $E_{IDREF} = \{(b_1, b), (e_1, b)\}$, $E_{PC} = \{(a, b), (a, e), (b, b_1), (b, b_2), (b, c), (b, d), (e, e_1), (e, e_2), (e, f), (e, g), (b_1, t_1), (b_2, t_2), (c, t_3), (e_2, t_3), (f, t_4), (g, t_5)\}$ and $E_{SB} = \{(b, e), (c, d), (f, g)\}$. The XML subgraph $G_b = (\{b \cup V'\}, E')$,
where \{b \cup V'\} = \{b, c, d, b1, b2, t1, t2, t3\}, \ E' = \{(b, b1), (b, b2), (b, c), (b, d), (b1, t1), (b2, t2), (c, t3)\}. Furthermore, the XML neighborhood \(N_g = (\{g \cup V'\}, E')\), where \{g \cup V'\} = \{g, e, f\} and \(E' = \{(e, g), (e, f), (f, g)\}\).

The XML graph \(G\) can be operated by the node operations (i.e., FC, LC, NS, PS, ID and TN), the neighborhood operations (i.e., IB and AC), and the subgraph operations (i.e., RM, RC and SE). The node operations access nodes in \(G\), the neighborhood operations manipulate XML neighborhoods, and the sub-graph operations operate on XML sub-graphs. To ensure data consistency, when a node operation in a transaction \(T_i\) is accessing node \(x\) in \(G\), the node \(x\) is protected from being updated by another transaction \(T_j\). In addition, when a neighborhood operation in \(T_i\) is accessing node \(x\), the nodes in neighborhood \(N_x\) are protected.

On the other hand, when a sub-graph operation in \(T_i\) is accessing \(x\), the nodes in sub-graph \(G_x\) are protected. Therefore, the nodes in neighborhood \(N_x\) and in sub-graph \(G_x\) must be obtained for \(T_i\) to ensure data consistency.

### 3.2. DOM operation conflict

In this sub-section, the conflicts between DOM operations on the XML document graph are analyzed. Conflicts in operations affect the correctness of a concurrency control protocol. In the most general cases, two operations conflict if they operate on the same data item and one of them is a write operation [19]. Therefore, there is a conflict if one of any two node operations operating on the same node in the XML graph is a write operation. However, since sub-graph and neighborhood operations operate on several nodes in an XML graph, they may conflict even though they are operating on different nodes. Thus, the conflict between two DOM operations, one of which is a sub-graph or neighborhood operation, is defined through their operated XML graphical structures. It should be noted that, for simplify, in the following Definition 3.4, symbols \(x\) and \(y\) are used to represent nodes \(C_{1,1}\) and \(C_{2,1}\) which are operated by operations \(O_{1,1}\) and \(O_{2,1}\).

**Definition 3.4.** Two DOM operations \(O_{1,1}\) and \(O_{2,1}\), operating on nodes \(x\) and \(y\) respectively in different transactions \(T_1\) and \(T_2\), conflict if one of them is a write operation and they satisfy the following conditions:

1. Both operations \(O_{1,1}\) and \(O_{2,1}\) are node operations, and node \(x\) is the same as node \(y\) in the XML graph.
2. Both operations \(O_{1,1}\) and \(O_{2,1}\) are neighborhood operations, and \(x\) is the node in the neighborhood \(N_y\) or \(y\) is the node in the neighborhood \(N_x\).
3. Both operations \(O_{1,1}\) and \(O_{2,1}\) are sub-graph operations, and \(x\) is the node in the sub-graph \(G_y\) or \(y\) is the node in the sub-graph \(G_x\).
4. Operation \(O_{1,1}\) is a node operation, \(O_{2,1}\) is a neighborhood operation, and \(x\) is the node in the neighborhood \(N_y\).
5. Operation \(O_{1,1}\) is a node operation, \(O_{2,1}\) is a sub-graph operation, and \(x\) is the node in the sub-graph \(G_y\).
6. Operation \(O_{1,1}\) is a neighborhood operation, \(O_{2,1}\) is a sub-graph operation, and \(x\) is the node in the sub-graph \(G_y\) or \(y\) is the node in the neighborhood \(N_x\).

Definition 3.4 defines two DOM operations in different transactions as conflicting if one of them is a write operation. Condition (1) shows two node operations conflicting on the same node. Conditions (2) and (3) describe two neighborhood operations and two sub-graph operations conflicting on the same neighborhood and sub-graph respectively. Condition (4) illustrates a node and a neighborhood operation conflicting on the same neighborhood. Condition (5) describes a node operation and a sub-graph operation conflicting on the same sub-graph, while Condition (6) shows a neighborhood operation and a
suppose a node operation \(c \cdot \text{FC} \) and a neighborhood operation \(b \cdot \text{IB}(d, k) \) in different transactions operate on the same XML graph in Figure 3. The neighborhood \(N_d = (\{d \cup V'\}, E') \), where \(\{d \cup V'\} = \{b, c, d\} \) and \(E' = \{(b, c), (b, d), (c, d)\} \). Thus, according to Condition (4), these two DOM operations in different transactions are conflicting, since node \(c \) is the node in \(N_d \).

4. XGP Protocol. In this section, an XML graph-based protocol XGP for XMKBs is proposed. Subsection 4.1 describes the protocol rules of XGP and Subsection 4.2 proves the correctness of XGP.

4.1. XGP protocol rule. The XGP protocol includes share (S) and exclusive (X) lock modes. S locks are employed for the nodes which are read by node, sub-graph and neighborhood operations, while X locks are introduced for the nodes which are written by node, sub-graph and neighborhood operations. For example, suppose a sub-graph operation \(a \cdot \text{SE}(b) \) in transaction \(T_1 \) operates on the XML document graph in Figure 3, it reads the nodes in the sub-graph \(G_b \). Therefore, the S locks are requested for the nodes in \(G_b \), including the nodes \(b, c, d, b_1, b_2, t_1, t_2 \) and \(t_3 \).

Table 2 illustrates the compatibility of S and X locks. The column heading of Table 2 indicates the locks held on a given node \(x \), while the row heading of Table 2 indicates the locks being requested on node \(x \) by another operation. An “O” or a “1” in an entry indicates whether a held lock and a requesting lock are compatible or incompatible, respectively. Therefore, if an S lock is held on a given node \(x \) by a transaction, a requesting S lock can be held on node \(x \) by another transaction. On the other hand, if an X lock is held on node \(x \), a requesting S or X lock cannot be held on node \(x \).

Based on the above-mentioned lock modes, a lock-based protocol for the XML document graph, namely XML graph-based locking protocol (XGP), is proposed. XGP consists of three types of rules, namely, the two-phase locking rule, propagation rules and compatibility rule. To ensure its serializability, the two-phase locking rule regulates the acquisition and release of locks. On the other hand, to request locks, the propagation rules propagate locks in XML neighborhoods and sub-graphs, and the compatibility rule resolves any conflicts between concurrent operations. Suppose that \(O_{i,m} \) is a DOM operation in transaction \(T_i \) operated on node \(x \) (i.e., \(C_{i,m} \)) in the XML document graph. XGP is composed of the following three types of protocol rules:

- **Two-Phase Locking Rule.** All lock modes which are acquired or released must observe the two-phase locking protocol.

- **Propagation Rules.** (1). For a neighborhood operation, the S lock is a request for the parent node \(y \) of node \(x \) and the X locks are a request for the nodes in neighborhood \(N_x \) except for node \(y \). (2). For a sub-graph operation, the locks are propagated from node \(x \) to the nodes in sub-graph \(G_x \).

- **Compatibility Rule.** Requests for particular lock modes can be granted on condition that the compatibility matrix in Table 2 is respected.

The XGP protocol operates as follows. Consider two transactions \(T_1 \) and \(T_2 \) executing on the XML document graph \(G \) in Figure 3, where \(T_1 \) comprises three DOM operations
Table 3. Possible schedule of transactions $T_1$ and $T_2$ under XGP

<table>
<thead>
<tr>
<th>Steps</th>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a_{.FC}$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$b_{.FC}$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>$a_{.FC}$</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>$b_{.FC}$</td>
</tr>
<tr>
<td>5</td>
<td>$b_{.RM(c)}$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$T_1$ finishes</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>$b_{.IB(c, k)}$</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>$T_2$ finishes</td>
</tr>
</tbody>
</table>

$a_{.FC}$, $b_{.FC}$ and $b_{.RM(c)}$, and $T_2$ comprises three operations $a_{.FC}$, $b_{.FC}$ and $b_{.IB(c, k)}$. Among the DOM operations of transactions $T_1$ and $T_2$, the $FC$ operation is a node operation which requests $S$ locks on nodes $a$ and $b$, the $IB$ operation is a neighborhood operation which requests $X$ locks on nodes $c$ and $d$, and the $RM$ operation is a sub-graph operation which requests $X$ locks on nodes $c$ and $t3$. Table 3 illustrates a possible schedule of $T_1$ and $T_2$ under XGP. Initially, in Steps 1 and 2, $T_1$’s operations $a_{.FC}$ and $b_{.FC}$ are executed sequentially, and nodes $a$ and $b$ are locked by $S$ locks for $T_1$. Next, $T_2$’s operations $a_{.FC}$ and $b_{.FC}$ are executed respectively, in Steps 3 and 4, since conflicts do not occur due to Compatibility Rule. For $T_2$, nodes $a$ and $b$ are locked by $S$ locks. In Step 5, according to Propagation Rules, nodes $c$ and $t3$ are locked by $X$ lock for $T_1$. Next, $T_1$ finishes in Step 6. It should be noted that $T_2$’s operation $b_{.IB(c, k)}$ in Step 7 must wait until $T_1$ finishes, since it conflicts with $T_1$’s operation $b_{.RM(c)}$. Finally $T_2$ finishes in Step 8.

4.2. Correctness of XGP schedules. This subsection proves the conflict serializability of XGP. The following Propositions 4.1 and 4.2 formulate the properties of conflict serializability. Based on those propositions, Lemma 4.1 and Theorem 4.1 prove all of the schedules under XGP to satisfy conflict serializability.

**Proposition 4.1.** The two-phase locking protocol ensures conflict serializability.

**Proposition 4.2.** A schedule is not conflict serializable if, and only if, a cycle exists in the precedence graph of the schedule.

**Lemma 4.1.** The $S$ and $X$ locks under XGP generate no cycle in the precedence graph.

**Proof:** According to XGP, the $S$ and $X$ locks observe the two-phase locking rule. According to Proposition 4.1, XGP with lock modes restricted to $S$ and $X$ locks ensures conflict serializability. Besides, according to Proposition 4.2, no cycle is generated in the precedence graph of XGP with lock modes restricted to $S$ and $X$ locks.

**Theorem 4.1.** XGP ensures conflict serializability.

**Proof:** According to Lemma 4.1, all lock modes under XGP generate no cycles in the precedence graph. Therefore, the theorem is proved.

5. Comparison and Analysis. This section compares XGP with other protocols which can be used in XMKBs to improve system performance. The implementation issues of XGP are also discussed.
5.1. **Concurrency comparison.** In this subsection, our XGP is compared with other concurrency control protocols, including the traditional concurrency control protocol (i.e., 2PL [19]), graph-based protocol (i.e., DDG [7]) and the XML protocols [10,11,14,15].

2PL was originally developed and used in relational database systems. To the best of our knowledge, none of the current XMKBs (i.e., Mix and SISSD) uses 2PL for its concurrency control scheme. However, if the protocol of 2PL is used in these XMKBs, XGP is certain to outperform it, owing to the lower lock conflicts according to its **Propagation Rules** and **Compatibility Rule**. Suppose that 2PL is applied to the XMKBs with DOM operations. For the \texttt{b.IB}(c, k) operation in Figure 3, exclusive locks are requested for the element nodes \texttt{b}, \texttt{c} and \texttt{d}. Therefore, no operation can be executed concurrently on nodes \texttt{b}, \texttt{c} and \texttt{d}. In contrast, for XGP, according to the **Propagation Rules**, one exclusive lock is requested for node \texttt{c} and one share lock is requested for node \texttt{b}. It is apparent that other read operations (i.e., \texttt{b.FC}) can be executed concurrently with operation \texttt{b.IB} (c, k) in Figure 3. As a result, XGP allows more concurrency than 2PL.

There is one factor which ensures that XGP is more suitable for XMKBs than the graph-based protocols DDG. This is that DDG can exploit the rich structure of XMKBs to support the interleaved, concurrent execution of several users' requests, thereby improving the overall system performance. In DDG, a transaction can lock any data item initially. To lock any additional data item, the transaction must hold locks on the majority of the parents of the desired data item. DDG holds a great many locks on the nodes in XMKBs for a DOM operation. Suppose that DDG is applied to the XMKBs with DOM operations. For the \texttt{b.IB}(c, k) operation on Figure 3, share locks are requested for element nodes \texttt{a}, \texttt{e} and \texttt{b}, and exclusive locks are requested for element nodes \texttt{c} and \texttt{d}. However, according to the **Propagation Rules**, one exclusive lock is requested for node \texttt{c} and one share lock is requested for node \texttt{b}. As a result, XGP allows more operations in different transactions to be executed in XMKBs than DDG.

The existing XML protocols [10,11,14,15] cannot to be directly applied to XMKBs to increase the transactional performance. These protocols model XML documents as trees and hold locks on their nodes while transactions are being executed in these trees. Moreover, these XML protocols assume that there are no cycles in the underlying structure. Unfortunately, the structure of an XMKB will contain cycles, and as a result, these XML protocols are not suitable for XMKBs.

5.2. **Implementation issues.** This subsection has discussed implementation issues, including the implementation of lock mechanisms and the deadlock problem.

An efficient locking mechanism is necessary to implement XGP, since XGP is a lock-based protocol. [5,19] suggests several efficient mechanisms for lock-based protocols to store and access locks in a constant time. Among the mechanisms discussed, the hash table provides an efficient structure for implementing locks in XGP. The cost to check lock conflicts in the hash table is required.

Another important issue is the deadlock problem. Deadlocks may occur in general, since transactions may wait for each other if the operations in these transactions do not commute. In general, lock-based protocols may result in deadlocks, and our XGP is no exception. To fulfill this requirement, traditional deadlock detection and recovery algorithms [19] can be used. However, a costly deadlock-detection graph may need to be maintained. A more suitable method is to design a deadlock-free protocol without the need for deadlock detection, and this will be part of our future work.

5.3. **Experimental results.** This section illustrates our simulation study on XGP and compares its performance with that of DDG. It should be noted that the performance of 2PL and the existing XML protocols are not evaluated in our simulation, since [19]
has shown that DDG outperforms 2PL and the existing XML protocols cannot to be used for XML documents which have a graph structure. Our simulator is implemented in C++ language and runs on a Windows XP operating system. Simulation experiments are conducted on a PC equipped with a 3.2GHz CPU, 480 MB main memory, and 80GB hard disk. In the simulator, XML documents are generated using the xmlgen tool of the XMark project [3]. Table 4 lists the parameters and their settings in the experiments. Parameters $d$ denotes the document size, $idref$ controls the percentage of the ID/IDREF relationship between the nodes, $t$ illustrates the number of transactions, $w$ denotes the percentage of write operations among transactions, $o$ indicates the number of operations in a transaction, and $mpl$ controls the maximum number of transactions which can be executed in the system.

Two experiments are performed to illustrate the transactional performance under XGP and DDG. The first experiment observes the effect of the percentage of ID/IDREF relationships on the performance of XGP and DDG, and the second compares XGP and DDG under different percentages of write operations. The first experiment, the results of which are shown in Figure 4, observes the throughputs of XGP and DDG under different percentages of the ID/IDREF relationship. The parameter settings are $d = 100K$, $t = 200$, $o = 3$, $w = 30\%$, and $mpl = 4$. In Figure 4, each of the two curves decreases at $idref > 40\%$, and a possible reason for this is that the number of manipulated nodes increases at $idref > 40\%$. In addition, the XGP curve has a higher throughput than the DDG curve because the number of conflicts in XGP is smaller than that in DDG.
The second experiment (results shown in Figure 5) compares XGP and DDG in the throughput by varying the percentage of write operations. The parameter settings are $d = 100K$, $idref = 40\%$, $t = 200$, $w = 40\%$, $o = 3$ and $mpl = 4$. In Figure 5, both XGP and DDG decrease their throughputs as the number of write operations increases. However, XGP outperforms DDG on throughputs, and this is because XGP considers XML graphical structures from a DOM operation perspective which enables more operations to be executed concurrently.

### 6. Conclusion.

This paper presented an analysis of XML document graphical structures for DOM APIs in XMKBs to increase concurrency. These analyses were used to propose a new concurrency control protocol, XGP, to improve the performance of DOM APIs to manipulate XML documents in XMKBs. XGP was also proved to be correct; that is, schedules under XGP always generate serializable schedules. Further, the experimental results show that XGP can achieve better concurrency in terms of throughput. Our future work will include designing a deadlock-free protocol. Deadlocks generally exist in lock-based protocols, and our XGP is no exception. If a deadlock exists in XGP, a traditional deadlock detection and recovery algorithm should be maintained in the system, which requires system resources.

### Acknowledgment.

This work is partially supported by “NSC98-2218-E-275-001-**”. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

### REFERENCES


