Serializability of Nested Transactions in Multidatabases *

Ugur Halici[1], Budak Arpinar[2] and Asuman Dogac[2]

Software Research and Development Center
[1]Dept. of Electrical Engineering, [2]Dept. of Computer Engineering
Middle East Technical University (METU), 06531 Ankara Turkiye
halici@rorqual.cc.metu.edu.tr, {asuman, budak}@srdc.metu.edu.tr

Abstract. *The correctness of nested transactions for multidatabases differs from that of flat transactions in that, for nested transactions the execution order of siblings at each related site should also be consistent. In this paper we first propose a simple but powerful theory for the serializability of nested transactions in multidatabases and then a technique called Nested Tickets Method for Nested Transactions (NTNT). The NTNT technique provides correctness of nested transactions in multidatabases without violating the local autonomy of the participating DBMSs. The algorithm is fully distributed, in other words there is no central scheduler. The correctness of the NTNT technique is proved by using the developed theory.*

1 Introduction and Related Work

A multidatabase system (MDBS) is a software that allows global applications accessing data located in multiple heterogeneous, autonomous DBMSs by providing a single database illusion. A multidatabase environment supports two types of transactions: local transactions submitted directly to a single Local DBMS (LDBMS), and executed outside the control of MDBS and global transactions that are channeled through the MDBS interface and executed under the MDBS control. The objectives of a multidatabase transaction management are to avoid inconsistent retrievals and to preserve the global consistency in the presence of updates.

Transaction management has always been one of the most important parts of a DBMS [GR 93]. The research on transaction management for centralized DBMSs is first extended to distributed DBMSs [BHG 87, HD 89, HD 91] and then to multidatabases. The transaction management for flat transactions in multidatabases have received considerable attention in recent years and correctness criteria have been defined [ZE 93] and several concurrency control techniques have been suggested [BGS 92, ZE 93, GRS 94]. In [GRS 94]

* This work is partially being supported by the Turkish State Planning Organization, Project Number: AFP-03-12DPT.95K120500, by the Scientific and Technical Research Council of Turkey, Project Number: EEEAG-Yazilim5, by Motorola (USA) and by Sevgi Holding (Turkey)

a ticket method is suggested to enforce serializability of global transactions in a MDBS environment. However it has been observed that nested transactions are more suitable to distributed environments since they provide more general control structures and support reliable and distributed computing more effectively. Nested transactions [M 85] facilitate the control of complex persistent applications by enabling both fine-tuning of the scope of rollback and safe intra-transaction parallelism. As a result nested transactions have become integral parts of some important standards, e.g. OMG's Common Object Services Specification (COSS). OMG's transaction service specification supports nested transactions along with flat transactions in a distributed heterogeneous environment based on the CORBA architecture [OMG 94]. Yet, to the best of our knowledge there is no technique suggested for the correctness of nested transactions in multidatabases, although some multidatabase projects have decided to use nested transaction model in their implementations [HFBK 94, DDK 96].

Principles and realization strategies of multilevel transaction management is described in [W 91]. A multi-level transaction approach to federated DBMS transaction management is discussed in [SWS 91].

DOM Transaction Model [BOH 92] for multidatabases allows closed nested and open nested transactions. InterBase Transaction Model [ELLR 90] is based on nested transaction model and allows a combination of both compensatable and non-compensatable subtransactions. However the correctness theory has not yet been developed for the models mentioned above.

In this paper we have developed a simple, neat and powerful theory for the serializability of nested transactions in multidatabases. Note that the theory provided in [BBG 89] for nested transactions could have been generalized to multidatabases. However the theory developed in [BBG 89] is very general in the sense that it takes semantics of transactions into account by allowing compatible transactions. Thus to prove the correctness of a concurrency control technique, commutativity and pruning concepts are used. We are able to develop a simpler theory, provided in Section 3, by not taking the semantics of transactions into account.

We then present a technique called Nested Tickets Method for Nested Transactions (NTNT) that provides for the correct execution of nested transactions in multidatabases. It should be noted that the concurrency control techniques developed for flat multidatabase transactions do not provide for the correctness of nested transactions in multidatabases because for nested transactions a consistent order of global transactions is not enough; the execution order of siblings at all levels must also be consistent at all sites.

The main idea of NTNT technique is to give tickets to global transactions at all levels, that is, both the parent and the child transactions obtain tickets. Then each global (sub)transaction is forced into conflict with its siblings through its parent's ticket at all related sites. The recursive nature of the algorithm makes it possible to handle the correctness of different transaction levels smoothly. NTNT technique also produces correct executions for flat transactions, flat transactions being a special case of nested transactions.

NTNT technique is fully distributed and does not violate the autonomy of participating LDBMSs. A transaction manager using the NTNT technique is implemented within the scope of the METU Interoperable DBMS (MIND) project [DDK 96, DEO 96]. MIND is based on OMG's object management architecture and is developed on top of a CORBA [OMG 91] compliant ORB, namely, DEC's Object Broker. A generic database object is defined through CORBA IDL and an implementation is provided for each of the participating DBMSs (namely, Oracle7[2], Sybase[3], Adabas D[4] and MOOD (METU Object-Oriented DBMS) [DAO 95]). Among these DBMSs Sybase and Adabas D support nested transactions. Therefore the restrictions of a global transaction to Sybase and Adabas D servers can be nested transactions, the others are flat transactions.

The paper is organized as follows: In Section 2 nested transaction models for centralized and multidatabase systems are given. Section 3 introduces a serializability theory for nested transactions in multidatabases. In Section 4, NTNT technique and its correctness proof are presented. We conclude with Section 5.

2 Nested Transactions

A nested transaction is a tree of transactions, the subtrees of which are either nested or flat transactions. The transaction at the root of the tree is called the top-level transaction. The others are called subtransactions. A transaction's predecessor in the tree is called a parent and subtransaction at the next lower level is called a child. The ancestors of a transaction are the parent of the subtransaction and recursively the parents of its ancestors. The descendants of a transaction are the children of the transaction and recursively the children of its descendants. The children of one parent are called siblings. We use the term (sub)transaction to refer to both top-level transaction and subtransactions.

2.1 A Nested Transaction Model

In the following we summarize a nested transaction model [CR 91] that we use in our work. Let t_0 be the top-level transaction, t_p be a root or a subtransaction, t_c be a subtransaction of t_p, and t_a be the ancestors of t_c.

i. Abort Rule: All the children t_c must be aborted if the parent transaction t_p aborts. A child transaction can abort independently without causing the abortion of its ancestors.

ii. Commit Rule: The parent transaction t_p cannot *commit* until all its children t_c commit or abort. The child transaction will *finally commit* only if it has *committed* and all its ancestors have *finally committed*.

iii. Visibility Rule: The child transactions t_c can view the partial results of their ancestors t_a, plus any results from committed detached transactions. Also they can view the partial results of their committed siblings due to following

[2] Oracle7 is a trademark of Oracle Corp.
[3] Sybase is a trademark of Sybase Corp.
[4] Adabas D is a trademark of Software AG Corp.

delegation rule.
iv. Delegation Rule: At commit, child transaction t_c delegates its objects to parent transaction t_p. So all changes done by a child transaction become visible to the parent transaction upon the child transaction's commit. The effects of delegation can be found in [CR 91].
v. Conflict Rule Between A Child Transaction and Its Ancestors: Consider a child transaction t_c and its ancestors t_a and conflicting operations p and q: t_a can not invoke q after t_c invokes p. □

It should be noted that rule v. prevents parent/child parallelism.

2.2 A Nested Transaction Model for Multidatabases

In distributed systems such as multidatabases, nested transaction model provides more general control structures to support reliable and distributed computing more effectively [HR 93].

A nested transaction submitted to a multidatabase may have to be executed in several LDBMSs if the related data is scattered across a number of sites. Operations submitted by (sub)transactions are executed by LDBMSs' Data Managers (DM) and they are called as DM operations. If a (sub)transaction in the hierarchy has a DM operation in a LDBMS, the operation is dispatched to the related site. If a LDBMS does not support nested transactions, their effect with respect to hierarchical domains of recovery can be simulated by using savepoints [GR 93].

Since each (sub)transaction of a nested transaction is failure-atomic, restrictions of a (sub)transaction to sites must be executed as an atomic unit. So, atomicity rule is defined for nested multidatabase transactions as follows:

Let t be a (sub)transaction and t^k be the restriction of t at sites $k = 1, 2, ..., n$.
i. Atomicity Rule: All the restrictions of a (sub)transaction t at sites $k = 1, 2, ..., n$ should be aborted if t aborts.

3 A Serializability Theory for Nested Transactions in Multidatabases

Before presenting the serializability theory of nested transactions in multidatabases, we provide an intuitive explanation.

It is possible to view a nested transaction as a tree, where the leaf nodes contain the DM operations and intermediate nodes represent the subtransactions. Note that this tree is not necessarily balanced. When a nested transaction is executed, the (sub)transactions that conflict on the same data item must be ordered in such a way that the order of their conflicting DM operations are preserved. Another important point is that when two subtransactions are ordered, this imposes an order between their parents. To be able to express these concepts formally we define an *ordered hierarchy* where the ordering imposed by the leaf nodes are delegated to the upper nodes in the hierarchy. With this ordered hierarchy definition it is possible to formally model an ordering within a

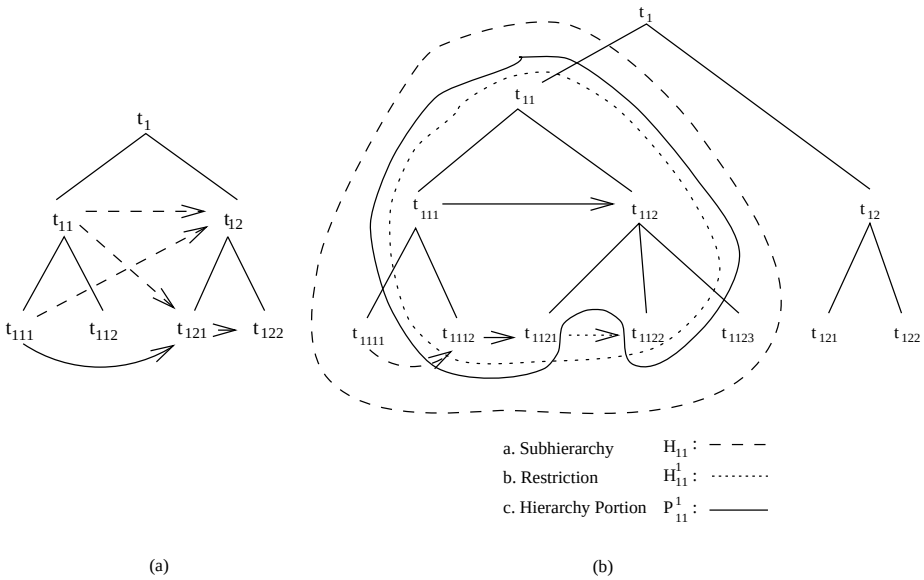

Fig. 1. (a) Illustration of delegation axiom, (b) Subhierarchy, restriction, and hierarchy portion

tree. Furthermore by assuming an imaginary root transaction for all submitted transactions it is possible to model an execution history of nested transactions through the ordered hierarchy definition.

In order to extend the theory to distributed DBMSs, we define *restriction* of a hierarchy to represent the executions at different sites. And to extend the theory further to multidatabases, where there is no global control on local transactions, we define *global portion* and *local portions* of an execution.

Definition 1. An **ordered hierarchy** (or shortly a **hierarchy**) is a tuple $H = (\rightarrow, O, T)$ where O is a set of nodes, T is a tree on O, and $\rightarrow$ is a nonreflexive and antisymmetric relation on O satisfying the following axioms for any $a, b \in O$

a. parent-child order[5] : $parent(a) \rightarrow a$

b. transitivity: if $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$

c. delegation: if $a \rightarrow b$ and

i. if $parent(b) \notin ancestors(a)$ then $a \rightarrow parent(b)$

ii. if $parent(a) \notin ancestors(b)$ then $parent(a) \rightarrow b$.□

In fact the relation defined is an ordering relation with further restrictions imposed by Definition 1.a and 1.c. The closure obtained by by applying the axioms of the hierarchy definition repeatedly is denoted by *.

Figure 1.(a) presents an example to clarify the delegation axiom. Given an

[5] Note that ordered hierarchy definition takes only sibling parallelism into consideration assuming the conflict rule given in Section 2.1.v. Therefore we have chosen parent's preorder priority in Definition 1.a.

ordering $t_{111} \rightarrow t_{121}$, t_{111} and t_{12} (which is $parent(t_{121})$) are ordered as $t_{111} \rightarrow t_{12}$ (from Definition 1.c.i). Also t_{121} and t_{11} (which is $parent(t_{111})$) are ordered as $t_{11} \rightarrow t_{121}$ (from Definition 1.c.ii). Yet although t_{121} and t_{122} are ordered as $t_{121} \rightarrow t_{122}$, since t_{12} (which is $parent(t_{122})$) is also one of the $ancestors(t_{121})$, the order is not delegated upwards. Finally, t_{11} and t_{12} are ordered as $t_{11} \rightarrow t_{12}$ (from Definition 1.c.i and ordering $t_{11} \rightarrow t_{121}$).

Definition 2. Let $H = (\rightarrow, O, T)$ be a hierarchy, and T_i be a complete subtree of T rooted at the node $t_i \in O$, and let T_i^k be a part of T_i such that T_i^k is also a tree rooted at the node t_i but $leaves(T_i^k) \subseteq leaves(T_i)$. A **restriction $\mathbf{H_i^k}$** is the tuple $H_i^k = (\rightarrow_i^k, O_i^k, T_i^k)$ where O_i^k is the set of nodes related to T_i^k and $\rightarrow_i^k$ is the restriction of the order $\rightarrow$ to O_i^k. If $T_i^k = T_i$ then the restriction is denoted as $H_i = (\rightarrow_i, O_i, T_i)$ and called as **subhierarchy**. A **hierarchy portion $\mathbf{P_i^k}$** of H on restriction H_i^k is the hierarchy tuple $P_i^k = (\rightarrow_i^{Pk}, O_i^k, T_i^k)$ satisfying $\rightarrow_i^{Pk} \subseteq \rightarrow_i^k$. If T_i is T itself, then restriction and portion related to part T^k are denoted as $H^k = (\rightarrow^k, O^k, T^k)$ and $P^k = (\rightarrow^{Pk}, O^k, T^k)$ respectively.□

Figure 1.(b) shows: a subhierarchy H_{11} rooted at t_{11} with $\rightarrow_{11} = \{t_{1111} \rightarrow t_{1112}, t_{1112} \rightarrow t_{1121}, t_{1121} \rightarrow t_{1122}, t_{111} \rightarrow t_{112}\}^*$, a restriction H_{11}^1 with $\rightarrow_{11}^1 = \{t_{1112} \rightarrow t_{1121}, t_{1121} \rightarrow t_{1122}, t_{111} \rightarrow t_{112}\}^*$, and a hierarchy portion P_{11}^1 with $\rightarrow_{11}^{P1} = \{t_{1112} \rightarrow t_{1121}, t_{111} \rightarrow t_{112}\}^*$. Note that $\rightarrow_{11}^{P1} \subseteq \rightarrow_{11}^1 \subseteq \rightarrow_{11}$ and $leaves(T_{11}^1) \subseteq leaves(T_{11})$. In Figure 1.(b) transitive edges and edges from parent to child are not shown for the sake of simplicity.

Proposition 1. Given hierarchy $H = (\rightarrow, O, T)$ then a restriction $H_i^k = (\rightarrow_i^k, O_i^k, T_i^k)$ is also a hierarchy since it satisfies Definition 1.□

In the following definition, a partial order represents an irreflexive, antisymmetric, and transitive relation.

Definition 3. A hierarchy $H = (\rightarrow, O, T)$ is said to be **partially (totally) ordered** iff $\rightarrow$ is a partial (total) order on O.□

Definition 4. A subhierarchy $H_i = (\rightarrow_i, O_i, T_i)$ of a hierarchy $H = (\rightarrow, O, T)$ is said to be **isolated** in H iff for any t_{im}, $t_{in} \in O_i$, $t_l \in O - O_i - ancestors(t_i)$, the following holds: $not(t_{im} \rightarrow t_l \rightarrow t_{in})$ and either $t_l \rightarrow t_{im}$ or $t_{im} \rightarrow t_l$.□

Definition 5. A subhierarchy $H_i = (\rightarrow_i, O_i, T_i)$ of a hierarchy $H = (\rightarrow, O, T)$ is said to be **hierarchically isolated** in H iff every subhierarchy H_{ij} (including H_i itself) of H_i is isolated in H.□

Definition 6. A hierarchy $H = (\rightarrow, O, T)$ said to be **serial** if H itself is hierarchically isolated and $\rightarrow$ is a total order.□

Definition 7. A hierarchy $H = (\rightarrow, O, T)$ is said to be **serializable** if there exists a serial hierarchy $H^+ = (\rightarrow^+, O, T)$, such that $\rightarrow \subseteq \rightarrow^+$.□

Theorem 1. A hierarchy $H = (\rightarrow, O, T)$ is serializable iff $\rightarrow$ is a partial order.

Proof: (if) Consider the preorder traversal of T where the siblings are traversed in consistency with the order $\rightarrow$. If the siblings are not ordered by $\rightarrow$ then their traversal order is immaterial. Since $\rightarrow$ is a partial order and since $\rightarrow$ is closed under delegation axiom, such a traversal exists. Let $\rightarrow^+$ be the total order determined by such a preorder traversal. Obviously $\rightarrow \subseteq \rightarrow^+$ and $H^+ = (\rightarrow^+, O, T)$ is serial. Therefore $H = (\rightarrow, O, T)$ is serializable by definition.

(only if) Assume $H = (\rightarrow, O, T)$ is serializable and $\rightarrow$ is not a partial order

(Note that our partial order relation is irreflexive, antisymmetric and transitive). Since $\rightarrow$ is transitive by definition of a hierarchy, there should be $a \rightarrow a$ for some $a \in O$. However, this in turn implies that there is no total order satisfying $\rightarrow \subseteq \rightarrow^+$, which means H is not serializable.□

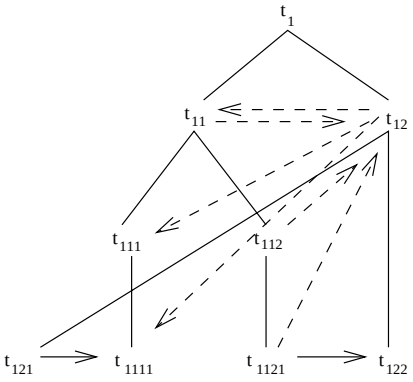

Fig. 2. An unserializable hierarchy

As an example to an unserializable hierarchy consider Figure 2. An initial order is given as $\{t_{121} \rightarrow t_{1111}, t_{1121} \rightarrow t_{122}\} \subseteq \rightarrow$. From the definition of ordered hierarchy, $\rightarrow$ also contains the following set obtained by applying the delegation axiom of Definition 1.c repeatedly: $\{t_{1121} \rightarrow t_{12}, t_{112} \rightarrow t_{12}, t_{11} \rightarrow t_{12}, t_{12} \rightarrow t_{1111}, t_{12} \rightarrow t_{111}, t_{12} \rightarrow t_{11}\}$. $\rightarrow$ is not a partial order because of $t_{12} \rightarrow t_{11}$ and $t_{11} \rightarrow t_{12}$ and hence the hierarchy in Figure 2 is not serializable.

Definition 8. A **nested transaction** T is a tree on $O = O_{dm} \cup O_{tr}$ where O_{dm} are the nodes representing the DM operations and O_{tr} are the nodes corresponding the abstract operations representing (sub)transactions, such that $\{leaves(T)\} = O_{dm}$ and $t = root(T)$ is the node representing the abstract operation corresponding to T and any subtree T_i rooted at $t_i \in \{child(t) - leaves(T)\}$ is a subtransaction defined recursively.□

We assume an imaginary top-level transaction such that any transaction submitted by the users is a subtransaction of it. Thus it is possible to model the execution history of nested transactions as an ordered hierarchy.

Definition 9. An **execution history** is an ordered hierarchy $H = (\rightarrow, O, T)$ where T is a transaction on O and $\rightarrow = (\rightarrow_{dm} \cup \rightarrow_{ep})^*$ where $\rightarrow_{dm}$ is the ordering requirements on the leaf nodes due to execution order of conflicting DM operations, $\rightarrow_{ep}$ is the ordering requirement due to execution policy[6]. A

[6] If there are additional ordering requirements due to transactions $\rightarrow$ can be written as $\rightarrow = (\rightarrow_{dm} \cup \rightarrow_{ep} \cup \rightarrow_{ts})^*$ where $\rightarrow_{ts}$ is the transaction specific ordering requirements. However it is easier here to assume any execution policy to cover such requirements, that is $\rightarrow_{ts} \subseteq \rightarrow_{ep}$.

subhierarchy of an execution is called a subexecution and a hierarchy portion of it is called an execution portion.□

Two DM operations are in **conflict** if one of them is a write operation, they operate on the same data item and they belong to different parents in the transaction tree. Note that in the transaction tree, the parent of a DM operation is the (sub)transaction itself that issued the DM operation.

We take the serializability of an ordered hierarchy as the correctness criterion of executions. Therefore as a consequence of Theorem 1 an execution history $H = (\rightarrow, O, T)$ is correct iff $\rightarrow$ is a partial order. At this point it should be noted that to provide the correctness of executions it is sufficient to find a total order consistent with the order of conflicting DM operations. In other words $\rightarrow_{dm}$ is the order to be preserved. Yet, a concurrency control technique while trying to guarantee the consistent order of DM operations may introduce a more restrictive ordering. We denote the ordering that stems from the execution policy as $\rightarrow_{ep}$. As an example, in a technique that allows only serial executions, $\rightarrow_{ep}$ itself is a total order.

In centralized databases only a single site contributes to the execution and the serializability of the execution can be checked easily. In distributed databases there are several sites contributing to the execution.

Definition 10. A **distributed execution** $H = (\rightarrow, O, T)$ is an execution history such that $O_{dm} = \cup_k(O^k_{dm})$ where O^k_{dm} is the set of DM operations on data items stored at site k, for $k = 1, .., n$. The execution of H at site k is the restriction $H^k = (\rightarrow^k, O^k, T^k)$ such that $leaves(T^k) = O^k_{dm}$.□

Notice that $O^k_{dm} \cap O^l_{dm} = \phi$ when $k \neq l$, however this is not true in general for O^k_{tr} and O^l_{tr}, since $O^k_{tr} \cap O^l_{tr}$ gives root nodes corresponding to subtransaction trees having DM operations at both sites.

$H = (\rightarrow, O, T)$ with restrictions $H^k = (\rightarrow^k, O^k, T^k)$ at site k, for $k = 1, .., n$, satisfies $O = \cup_k(O^k)$, $T = \cup_k(T^k)$, and $\rightarrow = (\cup_k \rightarrow^k)^*$. $\rightarrow$ contains the order enforced by the distributed execution policy, $\rightarrow_{dep}$ in addition to $\cup_k \rightarrow^k_{dm}$; on the other hand $\rightarrow^k$ contains also $\rightarrow^k_{dep}$.

In distributed DBMSs, the concurrency control information related to hierarchy restrictions are completely available and can be used to decide on the serializability of the execution. In distributed DBMSs the restriction $H^k = (\rightarrow^k, O^k, T^k)$ of H at site k is known.

However, in multidatabases, the complete information about H is not available. A local scheduler at site k knows only the local execution portion $L^k = (\rightarrow^{Lk}, O^k, T^k)$ where $\rightarrow^{Lk} \subseteq \rightarrow^k$ and does not have the complete information on the restriction $H^k = (\rightarrow^k, O^k, T^k)$.

Furthermore a global scheduler in multidatabases have knowledge only about global execution portion, $G = (\rightarrow^G, O^G, T^G)$ while a distributed DBMS scheduler has the complete information about $H = (\rightarrow, O, T)$.

Definition 11. On a multidatabase having sites $k = 1, .., n$, a **multisite execution** $H = (\rightarrow, O, T)$ is an execution history such that

• $O = O^G \cup (\cup_k O^{Lk})$, $O^G = \cup_k O^{Gk}$, and $T = T^G \cup (\cup_k T^{Lk})$, $T^G = \cup_k T^{Gk}$ and $\rightarrow = ((\cup_k \rightarrow^{Lk}) \cup \rightarrow^G)^*$,

• H has restrictions $H^k = (\rightarrow^k, O^k, T^k)$ at site k, for $k = 1,..,n$ where $O^k = O^{Gk} \cup O^{Lk}$ and $T^k = T^{Gk} \cup T^{Lk}$ with a local execution portion $L^k = (\rightarrow^{Lk}, O^k, T^k)$, $\rightarrow^{Lk} \subseteq \rightarrow^k$, $\rightarrow^{Lk} = (\rightarrow^{Lk}_{dm} \cup \rightarrow^{Lk}_{lep})^*$ where $\rightarrow^{Lk}_{lep}$ is the ordering enforced by local execution policy,

• H has restriction $H^g = (\rightarrow^g, O^G, T^G)$ to T^G with a global execution portion $G = (\rightarrow^G, O^G, T^G)$, $\rightarrow^G = ((\cup_k \rightarrow^{Gk}_{dm}) \cup \rightarrow_{gep})^*$ where $\rightarrow_{gep}$ is the ordering due to global execution policy, $\rightarrow^G \subseteq \rightarrow^g$,

• G has restriction $G^k = (\rightarrow^{Gk}, O^{Gk}, T^{Gk})$ at sites $k = 1,..,n$, $\rightarrow^{Gk} \subseteq \rightarrow^k$, $\rightarrow^{Gk} \subseteq \rightarrow^G$ and $\rightarrow^{Gk}_{dm} \subseteq \rightarrow^{Lk}_{dm}$ (but not necessarily $\rightarrow^{Gk} \subseteq \rightarrow^{Lk}$ or $\rightarrow^{Lk} \subseteq \rightarrow^{Gk}$).□

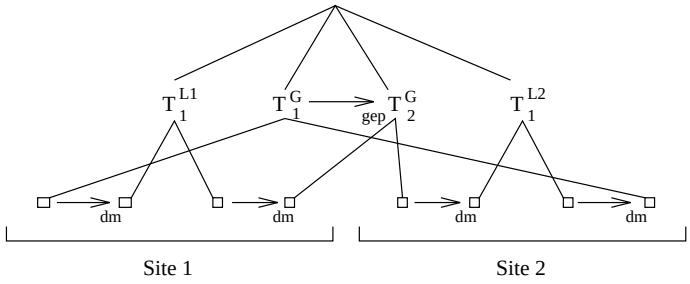

Fig. 3. A Multisite execution H

Figure 3 depicts a multisite execution H where the edges due to axioms of Definition 1 are not demonstrated for the sake of simplicity. Figure 4.(a) shows local execution portions L^1 and L^2 of H at site 1 and site 2 respectively. Note that local scheduler at site 1 does not have the complete knowledge of $\rightarrow^1$ which contains orderings that come from local scheduler at site 2 such as $T^G_2 \rightarrow T^G_1$. This is symmetrically true for the local scheduler at site 2. Also ordering due to global execution policy is hidden from the local schedulers. In Figure 4.(a), these orderings which are not included in $\rightarrow^{L1}$ and $\rightarrow^{L2}$ are depicted as dotted lines and the delegated orderings are displayed as dashed lines. In Figure 4.(b), the global execution portion G of H is given. $\rightarrow^G$ does not contain orderings coming from local schedulers due to conflicting DM operations with local transactions at those sites and these orderings are also depicted as dotted lines.

One of the necessary condition for serializability of H is that $(\cup_k \rightarrow^k)^*$ should not introduce any cycles, which is not satisfied in the example shown in Figure 3 and Figure 4.

Definition 12. A multisite execution is said to be **EGOL (enforcing global order locally on siblings)** iff $a \rightarrow^{Gk} b$ implies $a \rightarrow^{Lk} b$ whenever $parent(a) = parent(b)$, and $a, b \in O^G$, and $a, b \in O^k$ for any $a, b \in O$.□

Definition 13. A multisite execution is said to be **ELOT (enforcing local ordering transparency for siblings)** iff $a \rightarrow^{Lk} b$ implies $a \rightarrow^{Gk} b$ whenever $parent(a) = parent(b)$, and $a, b \in O^G$, and $a, b \in O^k$ for any $a, b \in O$.□

If an execution is EGOL and ELOT, then the order of the siblings are con-

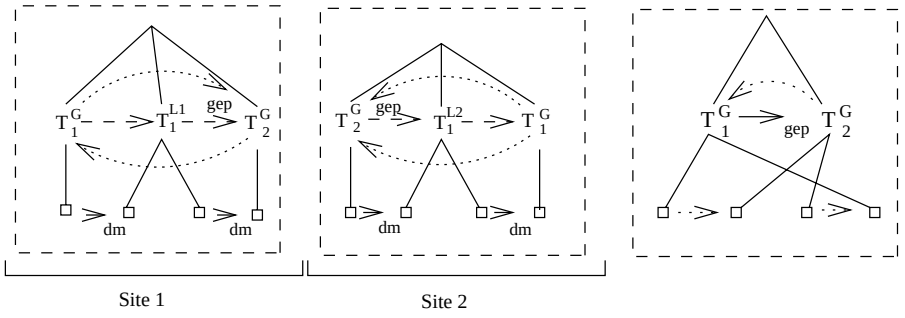

Fig. 4. (a) Local execution portions L^1 and L^2 of H at site 1 and site 2, (b) Global execution portion G of H

sistent at each site as shown in the following Lemma.

Lemma 1. If a multisite execution $H = (\rightarrow, O, T)$ is EGOL and ELOT then $a \rightarrow^{Lk} b$ implies $a \rightarrow^{Ll} b$ for any site k, l whenever $parent(a) = parent(b)$, and $a, b \in O^l$ and $a, b \in O^k$ for any $a, b \in O$.

Proof: $a \rightarrow^{Lk} b$ implies $a \rightarrow^{Gk} b$ by ELOT property which inturn implies $a \rightarrow^{G} b$ since $G^k = (\rightarrow^{Gk}, O^{Gk}, T^{Gk})$ is the restriction of G at sites k. $a \rightarrow^{Gl} b$ since $a, b \in O^l$ and since $G^l = (\rightarrow^{Gl}, O^{Gl}, T^{Gl})$ is the restriction of G at sites l. Furthermore, since the restriction is EGOL, $a \rightarrow^{Gl} b$ implies $a \rightarrow^{Ll} b$.□

Theorem 2. Let $H = (\rightarrow, O, T)$ be a multisite EGOL and ELOT execution having serializable local execution portions $L^k = (\rightarrow^{Lk}, O^k, T^k)$ at site k for $k = 1, .., n$. Then H is serializable iff the global portion $G = (\rightarrow^{G}, O^{G}, T^{G})$ is serializable.

Proof: (if) Due to EGOL and ELOT properties of H and Lemma 1, and due to serializability of L^k for $k = 1, .., n$ and serializability of G; all the orderings in $\rightarrow^G$ and $\rightarrow^{Lk}$, $k = 1, .., n$ are consistent for any siblings a, b. Therefore a preorder traversal $\rightarrow^+$ (total order) of T exists for any siblings a and b such that $\rightarrow^+$ is consistent with the following:

1. If $a, b \in O^G$ and $a \rightarrow^G b$ then a is traversed before b,
2. If $a, b \in O^k$ and $a \rightarrow^{Lk} b$ for any k then a is traversed before b,
3. Otherwise the ordering of a and b is immaterial

and this preorder traversal satisfies $\rightarrow \subseteq \rightarrow^+$. Since $\rightarrow^+$ is a total order consistent with $\rightarrow$, H is serializable.

(only if) If H is serializable then $\rightarrow$ is a partial order by Theorem 1. Since $\rightarrow^G \subseteq \rightarrow$ by definition this in turn implies $\rightarrow^G$ is a partial order. Therefore H^G is serializable by Theorem 1.□

4 Nested Tickets Method for Nested Transactions

In this section, a technique for global concurrency control of nested transactions in multidatabases, called Nested Tickets Method for Nested Transactions (NTNT) is presented.

NTNT ensures global serializability of nested multidatabase transactions without violating autonomy of LDBMSs. It is assumed that LDBMSs' schedulers guarantee local serializability of nested transactions.

We present the NTNT technique by referring to the pseudocode of the algorithm. To be able to provide a neat recursive algorithm, we imagine all the global transactions to be children of a virtual transaction called OMNI. When OMNI transaction starts executing, it creates a siteTicket(OMNI) at each site whose default value is 0. Then we imagine that OMNI transaction executes forever. Since it is an imaginary transaction, it does not need to commit finally to make the updates of its children persistent.

GlobalBegin(T_i^G) assigns a globally unique and monotonically increasing ticket number denoted as $TN(T_i^G)$ to all transactions denoted by T_i^G when they are initiated, that is, both the parent and the child transactions at all levels obtain a ticket. A Ticket Server object in MIND provides tickets and guarantees that any new subtransaction obtains a ticket whose value is greater than any of the previously assigned ticket numbers. Since any child is submitted after its parent, this automatically provides that any child has a ticket number greater than its parent's ticket. When the first DM read or DM write operation of a subtransaction T_i^G is to be executed at a local site, *LocalBegin(T_i^G, k)* is executed which starts all ancestors of the subtransaction if they are not initiated at this site yet. Next, each child transaction reads the local ticket created by its parent at this site (this ticket is created for the children of $parent(T_i^G)$, i.e. $siblings(T_i^G)$), and checks if its own ticket value is greater than the stored ticket value in the ticket for $siblings(T_i^G)$ at this site. If it is not, the transaction T_i^G is aborted at all related sites and resubmitted to MIND using the algorithms given in *GlobalAbort(T_i^G)* and *GlobalRestart(T_i^G)*. Otherwise, T_i^G sets the local ticket created by its parent to its own ticket value ($TN(T_i^G)$) and creates a site ticket, $siteTicket(T_i^G)$ with default value 0 for its possible future children. As a result, all siblings of a subtransaction accessing to some site k are forced into conflict through a ticket item created by the parent of these siblings at site k. The pseudocode of the algorithm to check ticket values is presented in *LocalCheckTicket(T_i^G, k)*. This mechanism makes the execution order of all siblings of a subtransaction to be consistent at all related sites since the execution is EGOL and ELOT by the use of tickets. In other words, the consistency of serialization order of the siblings are provided by guaranteeing them to be serialized in the order of their ticket numbers. If a transaction is validated using the *LocalCheckTicket(T_i^G, k)* algorithm then its read and write operations on any item x are submitted to related LDBMS by *LocalWrite(x)*, *LocalRead(x)* algorithms and committed by *GlobalCommit(T_i^G)*. *GlobalCommit(T_i^G)* is executed after all children of T_i^G commit or abort due to Commit Rule in Section 2.1.ii. *GlobalCommit(T_i^G)* coordinates the 2PC protocol and if all LDBMSs replied Ready then commits this subtransaction.

The NTNT Algorithm:

GlobalBegin(T_i^G):

Get global ticket for T_i^G so that
$TN(T_i^G)$:=lastTicketNo+1;
lastTicketNo:=$TN(T_i^G)$; □

LocalBegin(T_i^G, k):

If parent(T_i^G, k) has not started at site k yet then
LocalBegin(parent(T_i^G), k);
Forward begin operation for T_i^G as child of parent(T_i^G) to Local Transaction Manager (LTM) at site k;
else
Forward begin operation for T_i^G as child of parent(T_i^G) to LTM at site k;
LocalCheckTicket(T_i^G, k);
If check FAILs then GlobalRestart(T_i^G); □

LocalCheckTicket(T_i^G, k):

If T_i^G is not OMNI then
If siteTicket(parent(T_i^G)) > $TN(T_i^G)$ then FAIL;
else
siteTicket(parent(T_i^G)):=$TN(T_i^G)$;
create(siteTicket(T_i^G)) at site k with default value 0; □

LocalWrite(x), LocalRead(x):

If the site(x) is being visited for the first time by T_i^G then LocalBegin(T_i^G, k);
Forward the read/write operation to Local Data Manager on behalf of T_i^G; □

GlobalAbort(T_i^G):

for each related site send LocalAbort(T_i^G) message to LTM at site k; □

GlobalRestart(T_i^G):

GlobalAbort(T_i^G);
GlobalBegin(T_i^G); □

GlobalCommit(T_i^G):

wait until all children(T_i^G) commit or abort;
for each related site k send PrepareToCommit(T_i^G) message to LTM at site k;
If all LTMs have replied Ready
for each related site k send Commit(T_i^G) message to LTM at site k;
If any site fails to PrepareToCommit then GlobalAbort(T_i^G); □

An Example: In the following, an example is provided to clarify the NTNT technique. Assume a multidatabase system with two LDBMSs at sites 1 and 2. User transactions can be arbitrarily nested and each (sub)transaction can issue read and write operations denoted as $r(a)$ and $w(a)$ respectively.

Figure 5 depicts the execution of two nested multidatabase transactions T_1^G and T_2^G, and a local transaction T_1^{L2}. Global transaction T_1^G has two subtransactions T_{11}^G and T_{12}^G, and T_2^G has one subtransaction T_{21}^G. At site 1, first T_1^G writes a, then T_{11}^G writes a, and then T_{21}^G reads a. Therefore, T_1^G and T_{21}^G directly conflict at site 1 and the serialization order of the transactions is $\{T_1^G \rightarrow T_{21}^G\} \subseteq \rightarrow^1$.

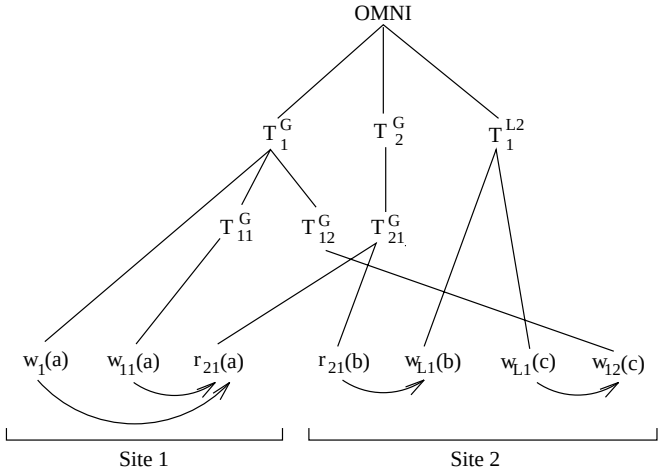

Fig. 5. A Schedule of Nested Multidatabase Transactions

Using the delegation axiom in Definition 1.c the serialization order of T_1^G and T_2^G at site 1 is $\{T_1^G \rightarrow T_2^G\} \subseteq \rightarrow^1$. At site 2, T_{21}^G reads b and later T_{12}^G writes c. Therefore, there is no direct conflict between T_{21}^G and T_{12}^G at site 2. However, a local transaction T_1^{L2} writes b and c, and thus T_{21}^G and T_{12}^G conflict indirectly at site 2. Therefore the serialization order is $\{T_{21}^G \rightarrow T_1^{L2} \rightarrow T_{12}^G\} \subseteq \rightarrow^2$ at site 2. Using the delegation axiom the serialization order of T_1^G and T_2^G at site 2 is $\{T_2^G \rightarrow T_1^G\} \subseteq \rightarrow^2$. Because of the local autonomy, the indirect conflict between siblings T_{12}^G and T_{21}^G at site 2 cannot be detected at the global level without a technique like NTNT. Although local schedules for nested transactions are serializable, the complete schedule is not serializable because the local schedules at sites 1 and 2 are not consistent with a total order $\{\rightarrow^1 \cup \rightarrow^2\} \subseteq \rightarrow$ defined on transactions T_1^G and T_2^G.

NTNT technique works for this example as follows: Assume the tickets obtained from the ticket server to be as follow: $TN(OMNI) = 0$, $TN(T_1^G) = 1$, $TN(T_2^G) = 2$, $TN(T_{11}^G) = 3$, $TN(T_{21}^G) = 4$, $TN(T_{12}^G) = 5$ and let $siteTicket(OMNI) = 0$ at each site.

Execution at site 1:

T_1^G is accepted since $siteTicket(parent(T_1^G)) = siteTicket(OMNI) = 0 < TN(T_1^G) = 1$ and $siteTicket(OMNI)$ is set to 1 and $siteTicket(T_1^G)$ is created with default value 0. Thus $w_1(a)$ is executed. Since $siteTicket(parent(T_{11}^G)) = 0 < TN(T_{11}^G) = 3$, $siteTicket(parent(T_{11}^G))$ is set to 3 and $siteTicket(T_{11}^G) = 0$ is created and $w_{11}(a)$ is executed. Similarly $siteTicket(parent(T_2^G)) = siteTicket(OMNI) = 1 < TN(T_2^G) = 2$, T_2^G is accepted and $siteTicket(OMNI)$ becomes 2 and $siteTicket(T_2^G)$ is created with default value 0. $r_{21}(a)$ is executed because $siteTicket(parent(T_{21}^G)) = 0 < TN(T_{21}^G) = 4$ and $siteTicket(parent(T_{21}^G))$ is set to 4 and $siteTicket(T_{21}^G)$ is created with default value 0.

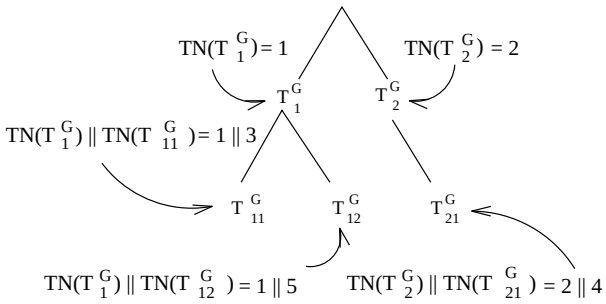

Fig. 6. Illustration of serialization order assignment through || (concatenation) operation

Execution at site 2:
T_2^G is accepted since $siteTicket(parent(T_2^G)) = TN(OMNI) = 0 < TN(T_2^G) = 2$ and $siteTicket(OMNI)$ is set to 2. $siteTicket(T_2^G)$ is created with default value 0. T_{21}^G is accepted and $r_{21}(b)$ is executed since $siteTicket(parent(T_{21}^G)) = 0 < TN(T_{21}^G) = 4$. Yet T_1^G at site 2 is rejected and aborted at all sites since $siteTicket(parent(T_1^G)) = siteTicket(OMNI) = 2$ which is not less than $TN(T_1^G) = 1$.

Correctness Proof of the Method:
Theorem 3. NTNT method produces serializable multisite executions.
Proof: To prove the serializability of any $H = (\rightarrow, O, T)$ produced by NTNT method we apply Theorem 2 through the following steps: **1)** We have only sibling parallelism and all the siblings are enforced into conflict with each other through their parent's ticket at all related sites. When the local serialization orders of transactions are not consistent with their ticket numbers, they are aborted. In the global execution portion $G = (\rightarrow^G, O^G, T^G)$, $\rightarrow^G$ is a total order consistent with the alphabetical ordering of $TNO(a) = TNO(parent(a)) \parallel TN(a)$ for any subtransaction $a \in O^G$ where $TNO(parent(a)) \parallel TN(a)$ denotes the concatenation of the $TNO(parent(a))$ and the ticket of a as illustrated in Figure 6. Note that the alphabetical order for OMNI is $TNO(OMNI) = 0$. **2)**Since L^k is serializable, $\rightarrow^{Lk}$ is a partial order for $k = 1, .., n$ from Theorem 1. For any $a, b \in O^G$ if $parent(a) = parent(b)$ and $a, b \in O^k$, then a and b conflict on a common ticket item at site k and these siblings are enforced to be ordered in $\rightarrow^{Lk}$ in the order of their ticket numbers otherwise they are aborted. Therefore H is EGOL. **3)** Furthermore for every sibling $a, b \in O^G$, if $a, b \in O^k$ then they are enforced to be ordered in $\rightarrow^{Lk}$. Since $a \rightarrow^{Gk} b$ implies $a \rightarrow^{Lk} b$, and since $\rightarrow^{Lk}$ is a partial order, it is not possible to have $b \rightarrow^{Lk} a$. Hence H is ELOT.

Therefore due to Theorem 2 we have $H = ((\cup_k \rightarrow^{Lk}) \cup \rightarrow^G)^*, O, T)$ serializable.□

5 Conclusions

In this paper we have presented a theory for the serializability of nested transactions in multidatabases and then developed a technique called NTNT that provides for the correctness of nested transactions in multidatabases. To the best of our knowledge NTNT is the first technique to provide serializability of nested transactions in multidatabases. The correctness of the NTNT technique is proved by using the developed theory. Note that the theory developed is general enough to be applicable to correctness of future techniques.

References

[BBG 89] C. Beeri, P. A. Bernstein, and N. Goodman. A Model for Concurrency in Nested Transaction Systems. Journal of the ACM, 36(2), 1989.

[BGS 92] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of Multidatabase Transaction Management. VLDB Journal, 1(2), 1992.

[BHG 87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. Concurrency Control and Recovery in Database Systems. Addison Wesley, Reading, MA, 1987.

[BOH 92] A. Buchman, M. T. Ozsu, M. Hornick, D. Georgakopulos, and F. A. Manola. A Transaction Model for Active Distributed Object Systems. In A. K. Elmagarmid (Ed.), Database Transaction Models for Advanced Applications, Morgan Kaufmann, San Mateo, CA., 1992.

[CR 91] P. K. Chrysanthis, and K. Ramamritham. A Formalism for Extended Transaction Models. In Proc. of the 17th Int. Conf. on VLDB, Barcelona, 1991.

[DAO 95] A. Dogac, M. Altinel, C. Ozkan, B. Arpinar, I. Durusoy, and I. Altintas. METU Object-Oriented DBMS Kernel. In Proc. of Intl. Conf. on Database and Expert Systems Applications, London, Sept. 1995, Lecture Notes in Computer Science, Springer-Verlag.

[DDK 96] A. Dogac, C. Dengi, E. Kilic, G. Ozhan, F. Ozcan, S. Nural, C. Evrendilek, U. Halici, B. Arpinar, P. Koksal, and S. Mancuhan. METU Interoperable Database System. Demo Description, In Proc. of ACM Sigmod Intl. Conf. on Man. of Data, Montreal, June 1996.

[DEO 96] A. Dogac, C. Dengi, and T. Ozsu. Building Interoperable Databases on Distributed Object Management Platforms. Communications of the ACM (to appear).

[ELLR 90] A.K. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A Multidatabase Transaction Model for Interbase. In Proc. of the 16th VLDB Conf., Brisbane, Australia, 1990.

[GR 93] J. Gray, and A. Reuter. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993.

[GRS 94] D. Georgakopoulos, M. Rusinkiewicz, and A. P. Sheth. Using Tickets to Enforce the Serializability of Multidatabase Transactions. IEEE Transactions on Knowledge and Data Engineering, 6(1), 1994.

[HD 89] U. Halici, and A. Dogac. Concurrency Control in Distributed Databases Through Time Intervals and Short Term Locks. IEEE Transactions on Software Engineering, 15(8), August 1989.

[HD 91] U. Halici, and A. Dogac. An Optimistic Locking Technique for Concurrency Control in Distributed Databases. IEEE Transactions on Software Engineering, 17(7), July 1991.

[HR 93] T. Harder, and K. Rothermel. Concurrency Control Issues in Nested Transactions. VLDB Journal, 2(1), 1993.

[HFBK 94] G. Huck, P. Fankhauser, R. Busse, and W. Klas. IRO-DB: An Object-Oriented Approach towards Federated and Interoperable DBMS. In Proc. of ADBIS'94, Moscow, May 1994.

[M 85] J. E. B. Moss. An Approach to Reliable Distributed Computing. MIT Press, 1985.

[OMG 91] Object Management Group. The Common Object Request Broker: Architecture and Specification. OMG Document, December 1991.

[OMG 94] Object Transaction Service. OMG Document, 1994.

[SWS 91] H.-J. Schek, G. Weikum, and W. Schaad, A Multi-Level Transaction Approach to Federated DBS Transaction Management. In Proc. of Int. Workshop on Interoperability in Multidatabase Systems, Kyoto, 1991.

[W 91] G. Weikum. Principles and Realization Strategies of Multilevel Transaction Management. ACM TODS, 16(1), 1991.

[ZE 93] A. Zhang, and A. K. Elmagarmid. Theory of Global Concurrency Control in Multidatabase Systems. VLDB Journal 2(3), 1993.