Global Semantic Serializability: An Approach to Increase Concurrency in Multidatabase Systems

Angelo Brayner[1] and Theo Härder[2]

[1] University of Fortaleza - UNIFOR, Dept. of Computer Science
60811-341 Fortaleza - Brazil
brayner@unifor.br

[2] University of Kaiserslautern, Dept. of Computer Science
D-67653 Kaiserslautern - Germany
haerder@informatik.uni-kl.de

Abstract. In this work, we present a new approach to control concurrency in multidatabase systems. The proposed approach is based on the use of semantic knowledge to relax the notion of absolute transaction atomicity. Supported by this new concept of atomicity, we propose a new correctness criterion, denoted global semantic serializability, for the execution of concurrent transactions, which provides *a high degree of inter-transaction parallelism*, *ensures consistency of the local databases* and *preserves autonomy of local databases*. Our proposal can also be used to increase concurrency in systems for integrating web data sources based on a mediator mechanism. Two concurrency control protocols we have developed are described.

1 Introduction

A multidatabase consists of a collection of autonomous databases, called local databases (LDBs). A key characteristic of local databases is that *they were created independently and in an uncoordinated way without considering the possibility that they might be somehow integrated in the future*. Systems used to manage multidatabases are called multidatabase systems (MDBSs). An MDBS should provide full database functionality and is built on top of multiple DBSs, called local DBSs (LDBSs), each of which operates autonomously. *Local autonomy* is a key feature of MDBS technology. The *multidatabase approach* provides inter-operability among multiple autonomous databases without attempting to integrate them by means of a single global schema [12].

A global application can access and update objects located in multiple databases by means of global transactions. In order to avoid inconsistencies, while allowing concurrent updates across multiple databases, MDBSs should provide a mechanism to control globally the concurrent access to local data.

Since the early seventies the concurrency control problem in a multiuser environment has been widely explored. In 1976, Eswaran *et al.* [8] proposed a model which introduces the concept of transaction. This model adopts a correctness criterion for the execution of concurrent transactions, called *serializability*, based on *transaction atomicity*. Serializability gives the illusion that the execution of a transaction is carried out in an isolated fashion without interference or interleaving from steps of other transactions. In other words, serializability gives the illusion that each transaction is executed as an atomic action. For that reason, we say that a transaction represents an atomic unit.

However, the nature and requirements of transaction processing in a multidatabase environment are quite different from those in conventional applications. Multidatabase transactions involve operations on multiple and autonomous local databases. Accordingly, they are relatively long-living. In addition, two different types of transactions may be executed in an MDBS context, global and local transactions. Although global and local transactions coexist, MDBSs do not have any information about the existence and execution order of local transactions due to local autonomy requirements. On the other hand, serializability requires knowledge of the execution order of all active transactions.

Therefore, the conventional concurrency control model is unsuited to MDBSs. To provide higher degree of inter-transaction parallelism in a multidatabase environment, new transaction models are needed, especially models that exploit multidatabase application semantics in order to relax serializability as a correctness criterion. Several researchers have started to extend serializability. However, the published solutions (as we will show in Section 3) either sacrifice local autonomy or support a low degree of parallelism.

The main motivation of this work is to provide an efficient solution to the concurrency control problem in multidatabase systems. We propose a new transaction model, denoted $\mathcal{GS}$-serializability, for synchronizing transactions in an MDBS environment. The proposed model *supports a high degree of parallelism among global transactions*, *ensures consistency of the local databases* and *preserves local autonomy of the local DBMSs*.

This work is structured as follows. Section 2 describes a model and reference architecture of MDBSs. Moreover, a running example which we use to illustrate definitions is presented. Some of the most important models for transaction processing in MDBSs are surveyed in Section 3. The new transaction model is presented in Section 4. Some realization aspects and the use of our approach in mediator based-systems are also discussed. In

Section 5, two concurrency control protocols, each of which implementing a different approach to ensure $\mathcal{GS}$-serializability, are outlined. Section 6 concludes this paper.

2 The Multidatabase System Model

An MDBS integrates a set of pre-existing and autonomous local DBSs. In turn, each local DBS consists of a local DBMS and a database. Users interact with the local DBMS by means of transactions. Two classes of transactions are supported in a multidatabase environment:

- Local transactions which are transactions executed by a local DBMS outside the control of the MDBS and
- Global transactions which comprise transactions submitted by the MDBS to local DBMSs. A global transaction G_i consists of a set of subsequences $\{\text{SUB}_{i,1}, \text{SUB}_{i,2}, \text{SUB}_{i,3}, \ldots, \text{SUB}_{i,m}\}$ where each $\text{SUB}_{i,k}$ is executed at LDBS_k as an ordinary (local) transaction.

Observe that the notion of global transaction reflects the fact that subsequences are executed at different sites, which usually do not have direct communication.

Formally, An MDBS consists of:

1. a set $\mathcal{LD}$=$\{\text{LDBS}_1, \text{LDBS}_2, \ldots, \text{LDBS}_m\}$ of local database systems where $m > 1$;
2. a set $\mathcal{L}$=$\{\text{L}_1, \text{L}_2, \ldots, \text{L}_m\}$ of local transactions where each L_k represents the set of local transactions executed at the local system LDBS_k, with $0 < k \leq m$; and
3. a set $\mathcal{G}$=$\{\text{G}_1, \text{G}_2, \ldots, \text{G}_n\}$ of global transactions.

Operations belonging to global transactions are executed by local DBMSs. Local transactions result from the execution of local applications. Henceforth, a *global transaction* will be denoted by G and a *local transaction* by L.

A *local schedule* S_k models the execution of several interleaved operations belonging to local and global transactions performed at a particular local system LDBS_k. A *global schedule* S^G, on the other hand, models the execution of all operations executed by global and local transactions on the multidatabase.

The architecture of an MDBS basically consists of the Global Transaction Manager (GTM), a set of Interface Servers (servers, for short), and multiple local DBSs. To each local DBS, there is an associated server. A local DBS consists of a DBMS and at least one database. The GTM comprises three modules: Global Transaction Interface (GTI), Global Scheduler (GS), and Global Recovery Manager (GRM). An MDBS architecture is depicted in Figure 1.

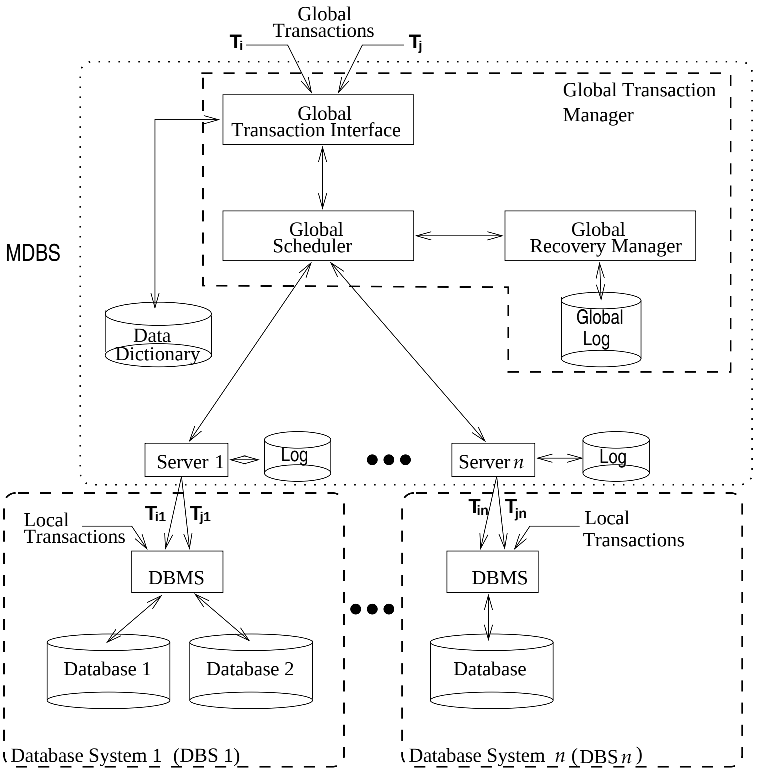


Fig. 1. A model for MDBS.

3 Related Work

As already said, the conventional concurrency control model is unsuited to the MDBS technology. For that reason, several researchers started to extend the conventional transaction model or the concurrency control protocols based on serializability.

Du and Elmagarmid propose in [7] the quasi serializability model for the transaction processing in multidatabase environments. This model is based on the assumption that *update actions executed by global transactions on objects of a particular local database never depend in any way on the values of objects stored in other databases, which were previously read by the same transaction.*

A global schedule S^G is said to be *quasi serial* if *(i)* all local schedules are serializable *and (ii)* global transactions in S^G are executed serially such that for any two global transactions G_i and G_j the following is valid: if G_i precedes G_j in S^G, then all G_i's operations precede G_j's operations in all local schedules in which both transactions appear. In [7], it is shown that quasi serial schedules preserve multidatabase consistency.

The class of correct schedules is broadened by the notion of *quasi serializable* schedules. A global schedule is *quasi serializable* if it is conflict equivalent to a quasi serial schedule. In order to identify quasi serializable schedules, a graph-based method is proposed. The key idea of this method is to construct a directed graph, denoted *quasi serialization graph* (QSG), for a global schedule. In a QSG for a global schedule S^G ($QSG(S^G)$) the nodes represent the global transactions in S^G. The edges of a QSG reflect direct and indirect conflicts among global transactions. A global schedule S^G is quasi serializable if $QSG(S^G)$ is acyclic and all local schedules are conflict serializable.

This model relaxes global serializability. However, it still requires serializable execution of global transactions. Quasi serializability suffers additionally from the following problem. As seen, information about indirect conflicts among global transactions is needed in order to construct quasi serialization graphs. Indirect conflicts are provoked by the execution of local transactions. Only local systems have knowledge about the existence of local transactions. Consequently, information about indirect conflicts can only be provided by local systems. Hence, a GTM implementing quasi serializability presumes that local systems will provide information about local transactions. Clearly, *such information flow (local system to global system) violates local autonomy.* Recall that local autonomy is a key property in MDBS technology.

Mehrotra et al. [13] propose the *two level serializability* (2LSR) model which , according to the authors, relaxes global serializability. This model is based on the following assumptions:

- At each local database there are two types of stored data: *Local data* and *global data*.
- Local transactions may not modify global data. Hence, local transactions are restricted to execute write operations only on local data.

Considering the assumptions above, Mehrotra et al. define that a global schedule S^G is 2LSR if all local schedules are conflict serializable and the execution of global transactions in S^G is serializable.

In [2] we show that the 2LSR model represents, in fact, the application of the notion of predicatewise serializability [10, 11] to multidatabase systems.

Since 2LSR is based on the notion of predicatewise serializability it inherits a serious shortcoming from the later model. 2LSR schedules may violate constraints. In [5] and [14] some examples are shown to illustrate this fact. Additionally, the 2LSR model presents the following two shortcomings. First, 2LSR assumes that objects in local databases are divided in local and global objects. Such an assumption represents a violation of local design autonomy since local database schemes should be modified in order to reflect the database division in local and global objects. Second, 2LSR requires that local transactions do not modify global objects. This strong restriction violates local execution autonomy.

The key problem for controlling concurrency in MDBS stems from the fact that global systems can not identify the serialization order of multidatabase transactions executed by local DBMSs. Georgakopoulos et al. [9] propose a strategy, denoted *ticket method*, to determine this order with the advantage that local systems do not need to give any information about the serialization order of transactions executing locally.

The basic idea of the ticket method is to force conflicts among multidatabase transactions. This is realized by the use of a special database object called *ticket*. Only one ticket is required per local system, and tickets may be accessed only by global transactions. Moreover, each subsequence of a global transaction executing at a local system must read the ticket value ($r(t)$), increment it ($t \leftarrow t+1$), and update the new value into the local database ($w(t)$). The ticket method presumes that all local DBMSs ensure serializability and support *prepare-to-commit* operations.

Note that global transactions conflict when they try to access tickets. Such conflicts make it possible to determine the relative serialization order of subsequences of multidatabase transactions at each LDBS.

The ticket method requires that all global transactions access tickets. This may create a "hot spot" at the local database. Moreover, local database schemes should be altered in order to represent tickets. Some mechanism should be implemented at the local systems in order to ensure that only global transactions access tickets. Such requirements violate local autonomy.

4 The $\mathcal{GS}$-serializability Model

4.1 Basic Concepts

An MDBS integrates a collection of "pre-existing" local databases. Such local databases were created independently and in an uncoordinated way without considering that they will be integrated sometime in the future. For that reason, it is reasonable to see a multidatabase as a collection of disjoint sets of objects, each of which representing a single local database. We call those disjoint sets of objects *semantic units*. It is also reasonable to assume that the result of an update action executed by a global transaction on an object belonging to a particular semantic unit does not depend on the values of objects belonging to other semantic units which are previously read by the same transaction. Based on this semantic knowledge, we relax the notion of absolute transaction atomicity in order to provide a high degree of transaction concurrency in an MDBS environment. In our approach, a global transaction may consist of more than one atomic unit.

Before formalizing the notion of semantic units, we need to specify an additional concept denoted *depends-on*. This concept stems from the *dependence relation* between the (final) result of an updating operation on an object x and the value of another object y. We say that an object x depends-on an object y, if and only if the result of at least one update operation on x in any program (that accesses x and y) is a function of (i.e. is depending on) a value of y read in the same program. The set of all objects on which x *depends* is called depends-on-set(x).

Definition 1. *Let $\mathcal{DB}$ be a database. We say that SU_i, $0 < i \leq n$, are* semantic units *of $\mathcal{DB}$, iff*
(i) $\mathcal{DB} = \bigcup_{i=1}^{n} SU_i$,
(ii) $\forall 1 \leq i, j \leq n, i \neq j : SU_i \bigcap SU_j = \emptyset$, *and*
(iii) $(\forall x \in SU_i, y \in SU_j, i \neq j) \Rightarrow (x \notin \textit{depends-on-set}(y)) \wedge (y \notin \textit{depends-on-set}(x))$ ◇

Intuitively, condition (iii) of Definition 1 reflects the idea that updates on objects of a semantic unit only depend on values of objects of the same semantic unit.

A *transaction* (global or local) is modeled as a finite sequence of *read* and *write* operations on database objects, where each object belongs to a particular semantic unit. We use $r_i(x)$ ($w_i(x)$) to represent a read (write) operation executed by a transaction T_i on a database object x. The set of operations executed by T_i is represented by $OP(T_i)$. In turn, $OP_{SU_a}(T_i)$ represents the set of operations belonging to T_i which are executed on objects of the semantic unit SU_a. Note that $OP_{SU_a}(T_i) \subseteq OP(T_i)$. It is assumed that *the execution of a transaction preserves database consistency if it runs isolated from other transactions.*

A transaction T is denoted module-structured if its operations are grouped into subsequences, called *modules*, such that each module represents an atomic unit of T. Intuitively, a *module-structured* transaction represents a sequence of modules where each module encompasses operations on objects of only one semantic unit. Further, the operations on objects of a semantic unit appear in only one module. For example, the following transaction may be characterized as being module-structured:

$$T_{Alice} = \underbrace{r_{Alice}(E)w_{Alice}(F)}_{module}\overbrace{r_{Alice}(P)w_{Alice}(U)w_{Alice}(V)}^{module}$$

Observe that the modules of a *module-structured* transaction are in fact atomic units. Here it is important to note that the notion of *module-structured* transactions reflects a transaction property. In other words, by means of this notion, we want to capture the fact that some transactions present a serial execution of atomic units without interleavings of operations belonging to different atomic units. It does not mean that our model requires that transactions should be partitioned into smaller pieces as proposed in [16].

Two schedules S_1 and S_2 over the set $\mathcal{T} = \{T_1, T_2, \cdots, T_n\}$ of transactions are said to be equivalent, denoted $S_1 \approx S_2$, if for any conflicting operations $p \in OP(T_i)$ and $q \in OP(T_j)$, the following condition holds: if $p <_{S_1} q$, then $p <_{S_2} q$. Observe that equivalent schedules produce the same effect on the database if they are executed on the same initial state.

Next, we define the concept of projection of a schedule on a set of transactions. Let S be a schedule over a set $\mathcal{G} \cup \mathcal{L}$ of transactions where $\mathcal{G}$ and $\mathcal{L}$ are disjoint sets of transactions. A projection $\mathcal{P}$ of S on the set $\mathcal{G}$ is a schedule for which the following conditions must hold:

(1) $\mathcal{P}$ only contains operations of transactions belonging to set $\mathcal{G}$
(2) $\forall p, q \in OP(\mathcal{P}) \Rightarrow p, q \in OP(S)$
(3) $\forall p, q \in OP(S'), p <_{\mathcal{P}} q \Leftrightarrow p <_S q$.

4.2 Correct Execution of Concurrent Transactions in MDBSs

In this section, we characterize correct schedules in our model. First, we define a standard for schedule correctness, denoted *global semantically serial schedules* ($\mathcal{GS}$-serial schedules, for short). Thereafter, we characterize schedules which produce the same effect on the database as a semantically serial one.

4.2.1 $\mathcal{GS}$-serial Schedules

Definition 2. *Let $S^G = \cup_{k=1}^{m} S_k$ be a global schedule over a set $\mathcal{T} = \mathcal{G} \cup \mathcal{L}$ of global and local transactions and $\mathcal{P}$ the projection of S^G on $\mathcal{G}$. The global schedule S^G is said to be $\mathcal{GS}$-serial if:*

(1) *each local schedule S_k is serializable and*
(2) *for each G_i in $\mathcal{P}$, G_i is module-structured and there is no interleaving within a module of G_i, for all modules in G_i (i.e., interleavings are only allowed between two modules of a transaction).*

We use $GS_eSerial$ to denote the class of all $\mathcal{GS}$-*serial* schedules over a given set of transactions. $\diamond$

Intuitively, the latter condition of Definition 2 enforces that in a $\mathcal{GS}$-serial schedule the projection of S^G on $\mathcal{G}$ represents, in fact, a serial execution of modules belonging to multiple global transactions. This implies, the interleaving granularity for global transactions in a $\mathcal{GS}$-serial schedule is a module.

Theorem 1. *A $\mathcal{GS}$-serial schedule preserves multidatabase consistency.*
Proof. Let S^G be a $\mathcal{GS}$-serial schedule whose operations are performed on objects of a multidatabase $\mathcal{MDB}$.
Case 1. Inconsistencies are caused by the execution of local schedules.
Without loss of generality, suppose that inconsistencies are produced by the local schedule S_k at local database LDB$_k$. This is impossible, since, by Definition 2, each local schedule is serializable. Hence, the execution of every local schedule preserves database consistency, as was to be proved.
Case 2. Inconsistencies are caused by the execution of $\mathcal{P}$.
Without loss of generality, consider that the inconsistency results from operations executed on objects of semantic unit $SU_a \subseteq \mathcal{MDB}$. By assumption, the execution of $\mathcal{P}$ represents a serial execution of modules (second item of Definition 2). Hence, the execution of operations on objects of SU_a in $\mathcal{P}$ represents a serial execution of modules belonging to different transactions. Consequently, inconsistencies on objects of SU_a must have been produced by some module of a transaction in (recall that there is no interleaving within a module). Thus, the inconsistency must have been caused by the execution of some global transaction in $\mathcal{P}$. This is a contradiction because, by assumption, a transaction preserves database consistency. So, $\mathcal{P}$ cannot produce an inconsistent state. That is, $\mathcal{P}$ preserves database consistency, as was to be proved. $\diamond$

4.2.2 $\mathcal{GS}$-serializable Schedules

So far, we have considered only $\mathcal{GS}$-serial schedules as being "safe". However, there are schedules which are not $\mathcal{GS}$-serial, but yield the same effect on the multidatabase as a $\mathcal{GS}$-serial one. That means, such schedules ensure multidatabase consistency and thereby may be considered as being "safe", too. Hence, we can broaden the class of safe schedules in our model and include these schedules.

Definition 3. *A global schedule $S^G = \cup_{k=1}^{m} S_k$ over a set $\mathcal{T}= \mathcal{G} \cup \mathcal{L}$ of global and local transactions is said to be $\mathcal{GS}$-serializable if and only if*

(1) *each local schedule S_k is serializable and*
(2) *the projection $\mathcal{P}$ of S^G on $\mathcal{G}$ is equivalent to the projection $\mathcal{P}_{SS}$ of a $\mathcal{GS}$-serial schedule S_{SS} over $\mathcal{T}$*

We denote the class of all $\mathcal{GS}$-serializable as GS_eSR. ◇

Intuitively, Definition 3 ensures that a global schedule S^G over a set $\mathcal{T}$ is $\mathcal{GS}$-serializable if and only if it is equivalent to a $\mathcal{GS}$-serial schedule over $\mathcal{T}$.

Another important benefit of $\mathcal{GS}$-serializability is that we can determine whether a global schedule is $\mathcal{GS}$-serializable by verifying the acyclicity of a directed graph, called *semantic serialization graph* (S_eSG).

Definition 4. *Let $S^G = \cup_{k=1}^{m} S_k$ be a global schedule over a set $\mathcal{T}= \mathcal{G} \cup \mathcal{L}$ of global and local transactions and $\mathcal{P}$ the projection of S^G on $\mathcal{G}$. The* Semantic Serialization Graph *for $\mathcal{P}$ is a directed graph $S_eSG(\mathcal{P}) = (N, E)$. The set N of nodes represents the transactions in $\mathcal{G}$, i.e., $N =\mathcal{G}$. The set E represents labeled edges of the form $G_i \xrightarrow{SU_k} G_j$, where:*

- *$G_i, G_j \in N$ and*
- *there are two operations $p \in OP(G_i), q \in OP(G_j)$, $p <_P q$, on an object of the semantic unit SU_k, which are in conflict.*

◇

Lemma 1. *If a schedule $S^G = \cup_{k=1}^{m} S_k$ is $\mathcal{GS}$-serial, then the* Semantic Serialization Graph *for the projection $\mathcal{P}$ of S^G on $\mathcal{G}$ is acyclic.*

Proof. Let $S^G \in \mathrm{GS}_e\mathrm{Serial}$ be a schedule over the set $\mathcal{T}= \mathcal{G} \cup \mathcal{L}$ of global and local transactions, where $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$. By condition 2 of Definition 2, the projection $\mathcal{P}$ of S^G over $\mathcal{G}$ represents a serial execution of modules of each transaction $G_i \in \mathcal{G}$, $0 < i \leq n$. Suppose, by way of contradiction, that $S_eSG(\mathcal{P})$ is cyclic and, without loss of generality, the cycle has the following form: $G_i \xrightarrow{SU_n} G_j \xrightarrow{SU_n} \cdots \xrightarrow{SU_n} G_i$. It follows from this that the *module* of G_i which represents the operations of G_i on objects of the semantic unit SU_n is interleaved by some operations of G_j, a contradiction, by assumption, S satisfies condition 2 of Definition 2. Therefore, $S_eSG(\mathcal{P})$ is acyclic, as was to be proved. ◇

Theorem 2. *A global schedule $S^G = \cup_{k=1}^{m} S_k$ over a set $\mathcal{T}= \mathcal{G} \cup \mathcal{L}$ of global and local transactions is gs-serializable if and only if:*

1. *the serialization graph (see [1]) for each local schedule S_k, $0 < k < m$, is acyclic, and*
2. *the semantic serialization graph for the projection of S^G on set $\mathcal{G}$ of global transactions is acyclic.*

Sketch of Proof.

Condition 1. By Definition 3, a schedule is gs-serializable if each local schedule S_k is serializable. In [1], it is shown that a schedule is serializable *if and only if* its serialization graph is acyclic. Hence, the *serialization graph* for each local schedule S_k is acyclic.

Condition 2. ("$\longrightarrow$") In a $\mathcal{GS}$-serializable schedule, the projection of S^G on $\mathcal{G}$ is equivalent to a $\mathcal{GS}$-serial schedule. By Lemma 1, the semantic serialization graph for the projection of a $\mathcal{GS}$-serial schedule Se^G on set $\mathcal{G}$ is acyclic. Since S^G is equivalent to Se^G (a $\mathcal{GS}$-serial schedule), the semantic serialization graph for the projection of S^G on set $\mathcal{G}$ is acyclic, too.

("$\longleftarrow$") Consider that the semantic serialization graph $S_eSG(\mathcal{P})$ for the projection of S^G on set $\mathcal{G}$ contains edges with label SU_n and it is acyclic. Thus, we may topologically sort it by edges with label SU_n. However, for this topological sort, instead of considering a transaction G_i as a node, we consider only the module of G_i, which contains the operations of G_i over objects of the semantic unit SU_n. As a result of this "modified" topological sort, we obtain a serial execution of modules of transactions in $\mathcal{G}$, where this serial execution contains operations on objects of the semantic unit SU_n. If we repeat this process "recursively" for each label in $S_eSG(\mathcal{P})$, we obtain a serial execution of modules of all global transactions in $\mathcal{G}$. Moreover, there is no interleaving within each module. We have, thus, a $\mathcal{GS}$-serial schedule (by Def. 2) which we call S_{SS} over the set $\mathcal{G}$ of global transactions. Since the projections $\mathcal{P}$ of S^G on set $\mathcal{G}$ and $\mathcal{P}_{SS}$ of S_{SS} on $\mathcal{G}$ have the same set of operations and order the conflict operations in the same way, they are equivalent. Therefore, by Definition 3, S^G is $\mathcal{GS}$-serializable, as was to be proved. ⋄

Theorem 2 has a very important practical impact. If we assume that all participating local DBMSs enforce *syntactic* serializability, we only need to verify the acyclicity of the semantic serialization graph for the execution of global transactions. This is a quite reasonable assumption, since all existing database systems implement serializability. Hence, we can apply this strategy to control concurrency in a multidatabase system.

Remark 1. Let $\mathcal{M}$ be a multidatabase system, where:

(i) all participating local DBMSs enforce serializability as local correctness criterion for schedules and

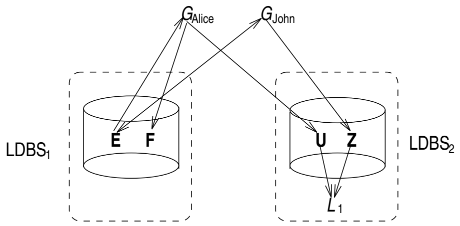
(ii) the GTM of $\mathcal{M}$ implements semantic serializability to synchronize global transactions.

Consider a schedule S^G over a set $\mathcal{G} \cup \mathcal{L}$ of global and local transactions. Furthermore, the operations of S^G are executed by $\mathcal{M}$. The schedule

S_{GTM} represents the projection of S^G on set $\mathcal{G}$. *The acyclicity of the graph* $S_eSG(S_{GTM})$ *is the necessary and sufficient condition to determine whether or not* S^G *is gs-serializable (correct).* $\diamond$

Remark 1 ensures that the transaction manager component of an MDBS can determine the correctness of global schedules without receiving any kind of information from the local systems. That means, a GTM implementing gs-serializability does preserve local autonomy. Recall that some of the proposals examined in Section 3 violate local autonomy.

Example 1. Consider a global schedule S^G which is executed in a multidatabase application. The execution scenario for S^G is depicted in Figure 2.



$G_{Alice} = r_{G_{Alice}}(E)w_{G_{Alice}}(U)w_{G_{Alice}}(F)$
$G_{John} = r_{G_{John}}(E)w_{G_{John}}(E)w_{G_{John}}(Z)$
$L_1 = r_{L_1}(U)r_{L_1}(Z)$
$S_{LDBS_1} = r_{G_{John}}(E)w_{G_{John}}(E)r_{G_{Alice}}(E)w_{G_{Alice}}(F)$
$S_{LDBS_2} = w_{G_{Alice}}(U)r_{L_1}(U)r_{L_1}(Z)w_{G_{John}}(Z)$
$S^G = r_{G_{John}}(E)w_{G_{John}}(E)r_{G_{Alice}}(E)w_{G_{Alice}}(U)r_{L_1}(U)w_{G_{Alice}}(F)r_{L_1}(Z)w_{G_{John}}(Z)$
$S_{GTM} = r_{G_{John}}(E)w_{G_{John}}(E)r_{G_{Alice}}(E)w_{G_{Alice}}(U)w_{G_{Alice}}(F)w_{G_{John}}(Z)$

Fig. 2. Execution scenario of Example 1.

By Definition, $S^G = S_{LDBS_1} \cup S_{LDBS_2}$. The projection of S^G on the set of global transactions is represented in Figure 2 by the schedule S_{GTM}. By Theorem 2, the global scheduler S^G is correct, since the semantic serialization graph for S_{GTM} is acyclic (Figure 3) and the serialization graphs for the local schedules S_{LDBS_1} and S_{LDBS_2} contain no cycles.

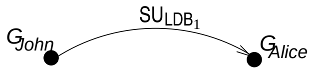


Fig. 3. Semantic serialization graph for the schedule S_{GTM}.

However, if we had assumed that the two local systems of the multidatabase application of Figure 2 enforce serializability, we could have considered Remark 1. In this case, only the acyclicity of $S_eSG(S_{GTM})$

had to be verified. This procedure had already indicated that the global schedule S^G is correct, without violating local autonomy. Observe that S^G is not quasi serializable.

Schedule S^G could also not be produced by concurrency control mechanisms implementing the *ticket method* (TM) or *altruistic locking* (AL) [15] protocols. That is because S^G is not executed in a serializable fashion. $\diamond$

Figure 4 depicts the relationship between the class of $\mathcal{GS}$-serializable schedules and some classes of schedules.

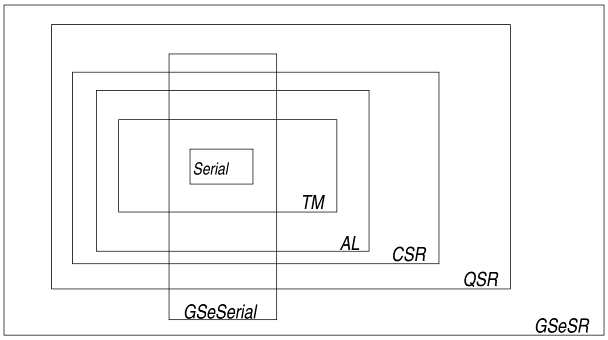


Fig. 4. Relationship among different classes of schedules.

In order to show that the class of QSR schedules is a subset of GS_eSR, consider the schedule S^G depicted in Figure 2. As seen in Example 1, S^G does not belong to the class of QSR schedules. However, S^G is an element of GS_eSR. This implies, $GS_eSR \setminus QSR \neq \emptyset$. Recall that a schedule S belongs to the class QSR if the global transactions in S are executed in a serializable fashion. Schedules belonging to GS_eSR do not necessarily present a serializable execution of their global transactions, since gs-serializability relaxes classical serializability. Therefore, $QSR \subset GS_eSR$.

4.3 Realization Aspects

In order to implement $\mathcal{GS}$-serializability for controlling concurrency in MDBSs, each participating local database should be defined as a semantic unit. Hence, the acquisition of information about the precise locality of database objects is a key question for using $\mathcal{GS}$-serializability in the multidatabase technology. However, such a problem is already addressed by MDBSs, since global systems must identify where global operations are to be performed.

Alternatively, we propose a mechanism which enables the GTM to automatically identify the location of database objects and thereby to identify the correct specification of semantic units. The basic principle of this mechanism is to use the component *Data Dictionary* of the MDBS architecture depicted in Figure 1. By doing this, sufficient information about local databases can be stored in the data dictionary. This information may be used by the GTM, more specifically by the GTI, in order to determine where each global operation should be executed. The GTI may forward this information to the GS. Such an information flow will give the GS the sufficient and necessary support to correctly determine the semantic units and their corresponding database objects.

Note that the process for identifying semantic units is realized without intervention of users, all is performed automatically.

4.4 Extending the Notion of Semantic Units in MDBSs

The notion of semantic unit is flexible enough to allow that two or more local databases can be logically grouped in order to represent a single semantic unit, without violating local autonomy of each local system. By "logically", we mean that only the GTM should be aware of such a representation. It is not necessary to physically join neither the databases nor the database systems.

Specifying more than one local database as a single semantic unit enables our transaction-processing model to synchronize transactions in MDBSs which have the following characteristics:

(i) Data replication. Some MDBSs may contain objects replicated in more than one local database. In this case, the local databases containing replicated data should be grouped in a semantic unit;

(ii) Global constraints. Constraints which span more than one local database are called global constraints. In this case, local databases containing objects referred in a global constraint should determine a semantic unit.

4.5 Increasing Concurrency in Mediator-based Systems

Mediator-based systems have been widely used to integrate heterogeneous web data sources. In such systems, wrappers are responsible for converting local data into a common model. In turn, a mediator provides an integrated view over the data exported by wrappers.

The integrated view can be either virtual or materialized. In the virtual approach, queries submitted to the mediator are decomposed into sub-queries, which are executed on the local web data sources. In other words, queries submitted to the mediator represent global transactions. A global transaction consist of a set of subsequences, where each subsequence corresponds to a subquery executed at a local web source as an ordinary (local) transaction.

In order to implement $\mathcal{GS}$-serializability to control concurrency in mediator-based systems (for integrating web data sources), each web source can be defined as a semantic unit. Since the mediator provides an integrated view of web data, the mediator can automatically identify the location of database objects and thereby identify the correct specification of semantic units.

5 Concurrency Control Protocols

5.1 Semantic Locking ($\mathrm{S}_e\mathrm{L}$)

The $\mathrm{S}_e\mathrm{L}$ protocol associates a lock to each database object. Three types of locks are supported: read-only, write and update locks. A transaction accesses an object if and only if a lock can be associated to the object on behalf of the transaction. Another key characteristic presented by the $\mathrm{S}_e\mathrm{L}$ protocol is to implement the two-phase property of the conventional 2PL protocol. However, it uses another granularity for realizing the two-phase property. In the 2PL protocol, this granularity is a transaction, since once a transaction has released a lock it may not obtain another lock. In the $\mathrm{S}_e\mathrm{L}$ protocol, the granularity is represented by the subsequence of operations on objects of one local database. This implies, locks held by a global transaction G on objects of local database LDB_k may be released after the last operation of G on objects of LDB_k.

Therefore, the $\mathrm{S}_e\mathrm{L}$ protocol guarantees that locks may be released by a global transaction before they complete their executions. This property increases the concurrency among global transactions. Moreover, local DBMSs do not need to hold locks on local resources on behalf of global (and remote) transactions for a long period of time. Additionally, the $\mathrm{S}_e\mathrm{L}$ protocol presents the following benefits. First, it reduces the frequency of deadlocks caused by lock conversions. Second, it implements a variable granularity locking strategy. Multiple lockable units support that concurrency may be enhanced by fine granularity, or locking overhead may be reduced by coarse granularity.

5.2 The S_eSG Checking Protocol

The protocol, denoted $\mathrm{S}_e\mathrm{SGC}$, is based on a similar strategy which is used by the conventional serialization graph testing protocol [6]: *the dynamic* *monitoring and management of an always acyclic conflict graph*. In contrast to the classical serialization graph testing, an S_eSGC protocol exploits semantic knowledge provided by the notion of semantic units.

The graph maintained by the S_eSGC protocol is called semantic conflict graph ($\mathcal{SC}$-graph). It is constructed according to the same rules used to construct a semantic serialization graph (Definition 4). Hence, nodes of the $\mathcal{SC}$-graph represent transactions and edges reflect conflicts between transactions. Notwithstanding, a $\mathcal{SC}$-graph differs from semantic serialization graphs in two aspects. First, not all committed transactions must be represented. Second, not all conflicts must be represented as an edge of the $\mathcal{SC}$-graph. That is because, in some cases, nodes and edges may be "safely" removed from the $\mathcal{SC}$-graph. Later we will show how this can be done.

The protocol works as follows. When a global scheduler (GS) using the S_eSGC protocol starts running, the $\mathcal{SC}$-graph is created as an empty graph. As soon as the scheduler receives the first operation of a new transaction (*begin-transaction*) G_i, a node representing this transaction is inserted in $\mathcal{SC}$-graph. For each operation $p_i(x) \in OP(G_i)$ which the GS receives, it checks if there is a conflicting operation $q_j(x) \in OP(G_j)$ which has already been scheduled. If an operation $q_j(x)$ has already been scheduled, the scheduler inserts an edge of the form $G_j \stackrel{SU_{LDB_k}}{\longrightarrow} G_i$, where x is an object belonging to the semantic unit SU_{LDB_k}. In fact, x is an object of the local database LDB_k.

Thereafter, the GS verifies if the new edge introduces a cycle in the $\mathcal{SC}$-graph. In the affirmative case, the GS rejects the operation $p_i(x)$, undoes the effect of operations of the subsequence $SUB_{i,k}$ and removes the edge $G_j \stackrel{SU_{LDB_k}}{\longrightarrow} G_i$ from $\mathcal{SC}$-graph. Otherwise, $p_i(x)$ is accepted and submitted to the corresponding server.

If the global scheduler identifies a cycle in the $\mathcal{SC}$-graph, only operations belonging to the atomic unit whose operation provokes the cycle are to be rolled back. It is not necessary to abort the entire global transaction.

6 Conclusions

In order to fulfil the requirements of transaction processing in MDBSs we have introduced a new transaction processing model. The key principle behind the proposed model is the use of semantic knowledge which is captured by means of the notion of *semantic units*. By means of the concept of semantic units, absolute transaction atomicity can be relaxed. Supported by this new notion of atomicity we have proposed a new correctness criterion, denoted $\mathcal{GS}$-serializability, for the execution of concurrent transactions in MDBSs. We have shown that $\mathcal{GS}$-serializability enforces multidatabase consistency, provides a high degree of inter-transaction parallelism, while preserving local autonomy (since it does not require any information about the execution of global transactions at the local systems). The notion of semantic units can be extended to allow two or more local databases to determine a single semantic unit.

Finally, two concurrency control protocols based on $\mathcal{GS}$-serializability were described. Although we have already developed a recovery mechanism for MDBSs using $\mathcal{GS}$-serializability (we refer the reader to [3]), we are aware that we have to investigate further on this direction. We are now working on the problem of global deadlock detection and resolution.

References

1. Bernstein, P. A., Hadzilacos, V. and Goodman, N. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.
2. Brayner, A. *Transaction Management in Multidatabase Systems.* Shaker-Verlag, 1999.
3. Brayner, A. and Härder, T. Recovery in multidatabase systems. In *Procedings of XIV Brazilian Symposium on Databases (SBBD 99)*, 1999.
4. Brayner, A., Härder, T. and Ritter, N. Semantic Serializability: A Correctness Criterion for Processing Transactions in Advanced Database Applications. *Data & Knowledge Engineering*, 31(1):1–24, 1999.
5. Breitbart, Y., Garcia-Molina, H., Silberschatz, A. Overview of multidatabase transaction management. *The VLDB Journal*, (2):181–239, 1992.
6. Casanova, M. A. The Concurrency Problem of Database Systems. In *Lectures Notes in Computer Science*, number 116. Springer-Verlag, 1981.
7. Du, W. and Elmagarmid, A. K. Quasi Serializability: a Correctness Criterion for Global Concurrency Control in InterBase. In *Proceedings of the 15th International Conference on VLDB*, pages 347–355, Amsterdam, 1989.
8. Eswaran, K.P., Gray, J.N., Lorie, R.A. and Traiger, I.L. The Notions of Consistency and Predicate Locks in a Database System. *Communications of the ACM*, 19(11):624–633, November 1976.
9. Georgakopoulos, D., Rusinkiewicz, M. and Sheth, A. Using Tickets to Enforce the Serializability of Multidatabase Transactions. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):1–15, February 1993.
10. Korth, H. F. and Speegle, G. D. Formal Model of Correctness Without Serializability. In *Proceedings of ACM SIGMOD Conference*, pages 379–386, 1988.
11. Korth, H. F. and Speegle, G. D. Formal Aspects of Concurrency Control in Long-Duration Transaction Systems Using The NT/PV Model. *ACM Transactions on Database Systems*, 19(3):492–535, September 1994.
12. Litwin, W., Mark, L. and Roussopoulos, N. Interoperability of Multiple Autonomous Databases. *Computing Surveys*, 22(3):267–293, 1990.
13. Mehrotra, S., Rastogi, R., S., Korth, H. and Silberschatz, A. Non-serializable Executions in heterogeneous distributed database systems. In *Proceedings of the*

First International Conference on Parallel and Distributed Information Systems, 1991.
14. Rastogi, R., Mehrotra, S., Breitbart, Y., Korth, H. and Silberschatz, A. On Correctness of Non-serializable Executions. In *Proceedings of the SIGMOD PODS*, pages 97–108, 1993.
15. Salem, K., Garcia-Molina, H. and Shands, J. Altruistic Locking. *ACM Transactions on Database Systems*, 19(1):117–165, March 1994.
16. Shasha, D., Simon, E. and Valduirez, P. Simple Rational Guidance for Chopping Up Transactions. In *Proceedings of 1992 ACM SIGMOD Conference*, pages 298–307, 1992.