Concurrency control in hierarchical multidatabase systems

Sharad Mehrotra[1], **Henry F. Korth**[2], **Avi Silberschatz**[2]

[1] Department of Computer Science, University of Illinois at Urbana Champaign 1304 W. Springfield Avenue, Urbana, IL 61801, USA
[2] Lucent Technologies, 600 Mountain Ave, Murray Hill, NJ 07974, USA

Abstract. Over the past decade, significant research has been done towards developing transaction management algorithms for multidatabase systems. Most of this work assumes a monolithic architecture of the multidatabase system with a single software module that follows a single transaction management algorithm to ensure the consistency of data stored in the local databases. This monolithic architecture is not appropriate in a multidatabase environment where the system spans multiple different organizations that are distributed over various geographically distant locations. In this paper, we propose an alternative multidatabase transaction management architecture, where the system is hierarchical in nature. Hierarchical architecture has consequences on the design of transaction management algorithms. An implication of the architecture is that the transaction management algorithms followed by a multidatabase system must be *composable* – that is, it must be possible to incorporate individual multidatabase systems as elements in a larger multidatabase system. We present a hierarchical architecture for a multidatabase environment and develop techniques for concurrency control in such systems.

Key words: Database management – Concurrency control – Distributed databases – Multidatabase management

1 Introduction

A multidatabase system (MDBS) is a facility, developed on top of local database management systems (DBMSs), that provides users of a DBMS access to data located in other heterogeneous data sources. Early prototype MDBSs (Templeton et al. 1983; Breitbart and Tiemann 1985; Landers and Rosenberg 1982) ignored the transaction management problem and did not support any scheme to coordinate the execution of the global transactions. These systems were designed to only provide read accesses to remote data. However, even if each global transaction is read-only, it can be shown (Mehrotra et al. 1992a), that the resulting schedule

may be non-serializable and the read-only global queries may retrieve inconsistent data. Transaction management issues, and the difficulty in supporting updates in MDBS environments were subsequently discussed in Gligor and Luckenbaugh (1984), Gligor and Popescu-Zeletin (1985, 1986), Breitbart and Silberschatz (1988), Breitbart et al. (1990), Mehrotra et al. (1990), and Elmagarmid and Du (1989). The difficulty arises due to the following two characteristics of MDBS environments:

– **Heterogeneity.** Each local DBMS may follow different concurrency control protocols and recovery algorithms.
– **Autonomy.** The participation of a local DBMS in the MDBS must not result in a loss of control by the local DBMS over its data and its local transactions.

Over the past decade, significant research has been done to identify mechanisms for effectively dealing with the problems that arise due to the heterogeneity and the autonomy of the local systems, e.g., Pu (1988), Breitbart (1988, 1990), Wolski and Veijalainen (1990), Mehrotra et al. (1992a, b), Elmagarmid and Du (1990), Batra et al. (1992). This research has resulted in transaction management algorithms that ensure correctness without sacrificing the autonomy of the individual systems. A large number of these proposed solutions have, however, considered the MDBS as a centralized software module. Clearly, if the local DBMSs are geographically distributed over different nodes of a worldwide computer network, then having a centralized MDBS will result in numerous undesirable consequences. For example, under high global transaction load, the site at which the MDBS software resides will become a bottleneck, resulting in the degradation of the system performance. More importantly, a failure of the site at which the MDBS resides will result in the MDBS being unavailable for processing global transactions, even though the transactions were to execute at only the sites that are operational.

Some of the above problems will be alleviated if the MDBS follows a distributed transaction management algorithm for concurrency control. A distributed mechanism for concurrency control in MDBSs have been suggested in Batra et al. (1992). However, developing the MDBS as a monolithic system in which the MDBS uses a single transaction management algorithm, whether distributed or centralized,

may still be undesirable. To see this, let us consider a typical MDBS environment in which users wish to execute transactions that span database systems belonging to multiple branches of an organization. Additionally, users also wish to execute transactions that span different autonomous organizations. One way to provide such a service is to develop a single monolithic MDBS system which integrates all the branches of all the organizations. However, depending upon the nature of transactions that execute within an organization, the computing resources available, and the reliability of the network, different organizations may prefer different MDBS transaction management algorithms for processing transactions local within the organization. For example, if a high degree of concurrency is critical for good performance in a certain organization, that organization may prefer a centralized MDBS transaction management algorithm for processing transactions local within the organization. On the other hand, if databases belonging to various branches of another organization are geographically distant and the network is not reliable, the organization may prefer a fully decentralized MDBS transaction management algorithm for processing transactions that execute within its different branches. Thus, it is preferable to develop the MDBS as a hierarchical system – each organization (or a set of organizations) has its own MDBS to control the execution of transactions within the organization. Furthermore, an inter-organization MDBS controls the execution of transactions that access data belonging to branches of different organizations. Note that using a single monolithic MDBS system, whether distributed or centralized, will adversely impact the performance of transactions that execute within an organization. In contrast, in a hierarchical MDBS, each organization can use a specialized transaction management algorithm suited to their environment.

The above scenario illustrates why it is desirable for the MDBS architecture to be hierarchical. If the architecture of the MDBS is hierarchical, different component MDBSs may follow different transaction management algorithms for ensuring consistency of the data they integrate. However, the transaction management algorithms followed by individual MDBSs must be such that it is feasible to compose the MDBSs into a larger MDBS. In this paper, we present a hierarchical architecture for MDBSs. We adopt serializability as the correctness criterion and study how existing techniques for ensuring global serializability in MDBS environments can be extended to ensure serializability in hierarchical MDBSs.

The rest of the paper is organized as follows. In Sect. 2, we discuss the motivation behind the transaction management problem in MDBSs and provide a summary of the progress that has been made over the last decade in this area. In Sect. 3, we formally define our MDBS architecture. In Sects. 4–6, we describe our mechanism for concurrency control in hierarchical MDBSs. Section 7 is on related work. Finally, in Sect. 8, we offer concluding remarks and present directions for future work. Proofs of the theorems developed in the paper are included in the appendix.

2 Transaction management in MDBSs

The transaction management problem in MDBSs consists of developing a software module to facilitate the execution of transactions that may span multiple heterogeneous and autonomous local DBMSs. If each local DBMS follows the *two-phase locking* protocol (Bernstein et al. 1987), is capable of participating in a *two-phase commit* protocol (Bernstein et al. 1987), and conforms to the X/Open DTP standard (Gray and Reuter 1993), then, from the perspective of transaction management, the local DBMSs can be integrated using existing transaction processing monitors (e.g., Encina by Transarc; Gray and Reuter 1993). There are three major reasons why this approach is unacceptable. These reasons collectively have motivated the research on transaction management in MDBSs.

First, the local DBMSs may be pre-existing legacy systems that may have been developed independently, without any regard to the possibility that these systems will be integrated into a MDBS at a later date. Legacy DBMSs may not adhere to current standards and may not even support an interface for the execution of the two-phase commit protocol. Requiring that the data from these pre-existing systems be migrated to a new system that is capable of interoperation may not be a feasible cost-effective solution to integration. Second, it is possible that the local DBMSs are highly specialized data management systems (as opposed to general-purpose systems) which have been developed for a specific application domain and they use special-purpose concurrency control and recovery algorithms. For example, a local DBMS may be a full-text database system used within an organization for storage and retrieval of office documents. Such a system may use a special-purpose transaction-processing scheme to preserve consistency of the document index. It may not be possible to integrate such specialized "home-brewed" local DBMSs into a MDBS using existing transaction-processing monitors. Another compelling reason why existing transaction-processing software does not suffice for the task of MDBS integration is that the usage of standard transaction management protocols (viz., the two-phase commit protocol) results in the violation of the local autonomy (Breitbart et al. 1990, 1992a, b; Veijalainen and Wokki et al. 1990; Mehrotra et al. 1992b). This is due to the fact that a two-phase commit protocol requires transactions to hold onto their locks (even at remote sites) for an unbounded period of time under certain adverse conditions (Bernstein et al. 1987; Gray and Reuter 1993). This can be viewed as a violation of the local autonomy, since it results in a local system losing control over its data and its applications.

Most of the approaches developed for transaction management in MDBSs treat local DBMSs as "black boxes" that cannot be modified for the sake of integration. Furthermore, in keeping with the autonomy requirement, which dictates that the applications local to a DBMS execute completely under its control, transactions are classified into two classes: *local transactions* that execute at a single DBMS, and *global transactions* that accesses more than one DBMS. While global transactions execute under the control of the MDBS software, local transactions execute outside its control. Each local DBMS is assumed to support an interface us-

ing operations belonging to subtransactions of global transactions which can be submitted for execution to the local DBMS. The nature of the interface supported affects the transaction management mechanism, including the mechanism developed in this paper, and we will discuss its impact after we have developed our scheme. Furthermore, it is assumed that each local DBMS ensures ACID properties of (sub)transactions that access data at the DBMS. That is, each local DBMS ensures serializability of the schedule local to it, and atomicity of the local transactions and the subtransactions of the global transactions that access data at its site.

Research on transaction management in MDBSs has been done along two complementary directions. A significant amount of work has been done to develop correctness criteria that are weaker than serializability, but nonetheless, can be implemented relatively efficiently in an environment where local DBMSs may follow heterogeneous transaction management protocols. These approaches are based on the assumption that the data integrity constraints in an MDBS environment are of a restricted nature. For example, it may be reasonable to assume that there are no data integrity constraints between data residing at two autonomous local DBMSs. Such a restriction on the nature of data integrity constraints can be exploited to develop correctness criteria, weaker than serializability, that preserve the constraints. Two examples of this approach are the notion of *quasi-serializability* (QSR; Du and Elmagarmid 1989) and *two-level serializability* (2LSR; Mehrotra et al. 1991). In Mehrotra et al. (1991), besides developing the correctness criterion 2LSR, a spectrum of MDBS models for which 2LSR ensures data integrity constraints is explored. Protocols for ensuring 2LSR have been developed in Mehrotra et al. (1992c) and Ouzzani et al. (1995).

The limitation of the above mentioned approaches lies not only in their inapplicability in domains where the restrictions on data integrity constraints are not valid, but as argued in Mehrotra et al. (1992c), preservation of the data integrity constraint may itself not be a sufficient consistency guarantee, – that is, executions that preserve all data integrity constraints may still be incorrect from the perspective of the user. To see this, consider an MDBS consisting of two banking databases located at sites s_1 and s_2. Further, let A_1 and A_2 be two accounts belonging to banking databases at sites s_1 and s_2 respectively such that there is no data integrity constraint that relates the two accounts. In such a case, if a transaction that transfers money from one account to the other executes concurrently with a transaction that reads both the accounts, then it is possible that the transaction that reads both the accounts sees a sum that differs from the true balance of the two accounts which may be unacceptable. Thus, even though each transaction sees a consistent state (that is, a state in which no data integrity constraints are violated) and the final state of the database is consistent, the execution is still undesirable.

The reason why preservation of data integrity constraint may not be sufficient consistency guarantee is that it is impossible to capture all the consistency requirements of the executions using integrity constraints over the data. This is a surprising observation, since most standard text on databases and concurrency control (Bernstein et al. 1987;

Papadimitriou 1986; Gray and Reuter 1993) motivate the need for serializability using the preservation of data integrity constraints as the theoretical basis of correctness.

Another significant body of research exists on mechanisms to ensure serializability in MDBS environments (Breitbart and Silberschatz (1988); Breitbart et al. (1990); Wolski and Veijalainen 1990; Pu 1988; Mehrotra et al. 1992a; Elmagarmid ad Du et al. 1990). One of the first significant approaches developed was in Pu (1988) where a notion of *o-element* was introduced. An o-element corresponding to a transaction is one of its operations that satisfies the following property— if a transaction T_1 is serialized before transaction T_2, then the o-element of T_1 occurs before the o-element of T_2. Using the notion of o-elements, the authors developed a validation based protocol that ensures serializability in an MDBS environment. Similarly, in Elmagarmid and Du (1990), the authors developed a scheme based on conservative timestamp ordering using the notion of o-elements (they refer to the o-element as the the *serialization event*). In Mehrotra et al. (1992a), it was shown that a notion similar to o-elements can be used to reduce the problem of ensuring serializability in MDBSs to that of ensuring serializability in traditional DBMSs. Using the reduction, any of the concurrency control schemes developed for traditional DBMSs can be employed to ensure serializability in MDBSs. This is a significant step in understanding the concurrency control problem in MDBSs, since it effectively overcomes the problems resulting from heterogeneity without jeopardizing the autonomy of the local DBMSs. It provides a framework for design and development of the concurrency control schemes for MDBSs, and facilitates comparison between previously published schemes that were developed in an ad-hoc fashion.

Much of the previous work on MDBS transaction management discussed above has not considered the impact of the MDBS architecture on the design of the transaction management software. As discussed in the introduction, there are compelling reasons for MDBSs to be developed as hierarchical systems. In the remainder of the paper, we describe a hierarchical transaction management architecture for MDBSs and study how existing techniques for ensuring serializability in MDBSs can be extended to hierarchical MDBSs. Concurrency control techniques for hierarchical MDBSs have previously been studied in Pu (1988) in the context of the *superdatabase* architecture. However, the developed technique does not provide the complete benefits of the hierarchical architecture. We will provide a detailed comparison of our scheme with the superdatabase approach in Sect. 7.

Notice that, in this paper, we will only study how approaches to ensuring serializability can be extended to ensure serializability in hierarchical MDBSs. Concurrency control schemes and the consistency guarantees that results in hierarchical MDBSs in which different MDBSs in the hierarchy may follow different correctness criteria (e.g., 2LSR, QSR) is not addressed and is an interesting avenue for future work. Furthermore, we did not consider the issue of failure-resilience in this paper. Failure-resilience is complicated, since the requirement of autonomy preservation renders the usage of a *two-phase commit* protocol (Bernstein et al. 1987) unsuitable for MDBS environments. In the absence of a two-phase commit protocol, it is possible that certain subtransactions of a multi-DBMS transaction
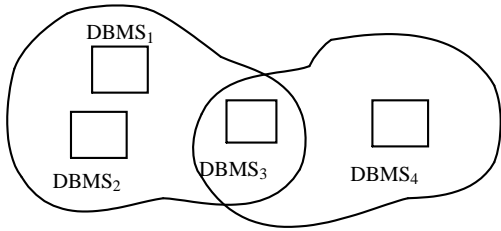
Fig. 1. An example MDBS environment

Fig. 2. Domain ordering for Fig. 1

commit, whereas others abort, thereby violating the atomicity property. The problem of ensuring atomicity in MDBS environments has been studied in Breitbart et al. (1990), Wolksi and Veijalainen (1990), Mehrotra et al. (1992b, d), Zhang et al. (1994). These approaches can be suitably adapted to achieve fault tolerance in hierarchical MDBSs. However, due to space limitations, we do not further address this issue.

3 Hierarchical MDBS architecture

An MDBS is an integrated collection of pre-existing local databases, $DBMS_1$, $DBMS_2$, ..., $DBMS_m$, that permits users to execute transactions that access multiple local DBMSs. Each $DBMS_i$ contains a set of data items that are denoted by DB_i. To describe the architecture of the MDBS, we associate with the MDBS environment a set of *domains* denoted by Δ with an ordering relation $\sqsubset$. A domain $D \in \Delta$ is either

– a set of data items in DB_i, for some $i = 1, 2, \ldots, m$, or
– a union of the set of data items in other domains $D_1, D_2, \ldots, D_n$, denoted by $\bigcup\{D_1, D_2, \ldots, D_n\}$, where $D_i \in \Delta$, $i = 1, 2, \ldots, n$,

The ordering relation $\sqsubset$, referred to as the *domain ordering* relation, is such that $D_i \sqsubset D_j$ iff $D_i \subset D_j$. We use $D_i \sqsubseteq D_j$ to denote that either $D_i \sqsubset D_j$ or $D_i = D_j$. Let D_i and D_j be domains in Δ. We refer to D_i as the child of D_j, denoted by $child(D_i, D_j)$, if $D_i \sqsubset D_j$ and for all $D_k \in \Delta$, either $D_i \not\sqsubset D_k$ or $D_k \not\sqsubset D_j$. We refer to D_j as a parent of D_i, denoted by $parent(D_j, D_i)$, if $child(D_i, D_j)$. We refer to a domain D_j as a *simple domain* if, for all D_i such that $parent(D_j, D_i)$, $D_i = DB_k$ for some local DBMS. That is, a simple domain is simply a collection of local DBMSs. We denote the set of domains $\{D| \text{ for all } D_k \in \Delta, D \not\sqsubset D_k\}$ by TOP.

A transaction $T_i = (O_{T_i}, \prec_{T_i})$, where O_{T_i} is the set of operations and $\prec_{T_i}$ is a partial order over operations in O_{T_i}. We assume that a transaction T_i that executes at a local DBMS (or a set of local DBMSs) consists of a set of **read** (denoted by r_i) and **write** (denoted by w_i) operations. This assumption is not central to the approach, and is made only for pedagogical reasons in the development of the examples. Further, each transaction T_i has **begin** (denoted by b_i), and **commit** (denoted by c_i) or **abort** (denoted by a_i) operations. A transaction that executes at multiple DBMSs may have multiple begin and commit (or abort) operations[1], one
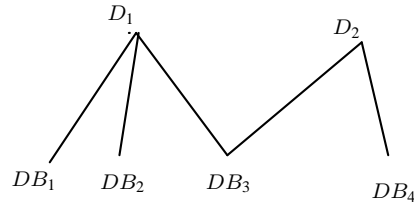
for each DBMS at which it executes. We denote by b_{ik} and c_{ik} (a_{ik}), the begin and commit (abort) operations of a transaction T_i in $DBMS_k$, respectively.

A transaction T_i is said to execute in a domain $D \in \Delta$, if there exists a DB_j, $DB_j \sqsubseteq D$, such that T_i accesses data items in DB_j. A transaction T_i may execute in multiple domains subject to the restriction that, if T_i accesses data items in $DB_1, DB_2, \ldots, DB_k$, then there must exist a domain $D \in \Delta$ such that $DB_j \sqsubseteq D$, $j = 1, 2, \ldots, k$. Such a domain D is denoted by $Dom(T_i)$. Thus, if T_i accesses data items in DB_j, then $DB_j \sqsubseteq Dom(T_i)$. A transaction T_i is said to be *global* with respect to a domain $D \in \Delta$, denoted by $global(T_i, D)$, if T_i executes in D and there exists a domain D', $D' \not\sqsubseteq D$ and $D \not\sqsubseteq D'$ such that T_i executes in D'. A transaction T_i is local with respect to a domain D, denoted by $local(T_i, D)$, if T_i executes in D and $\neg global(T_i, D)$. We illustrate the above-defined notations by the following example.

Example 1. Consider an MDBS environment consisting of four local DBMSs as illustrated in Fig. 1. The set of domains, $\Delta = \{DB_1, DB_2, DB_3, DB_4, D_1, D_2\}$, where domain $D_1 = \bigcup\{DB_1, DB_2, DB_3\}$ and domain $D_2 = \bigcup\{DB_3, DB_4\}$. The domain ordering relation for the MDBS environment depicted in Fig. 1 is illustrated in Fig. 2. Suppose a transaction T_1 accesses data items in domains DB_1 and DB_2. Then, $Dom(T_1) = D_1$, $global(T_1, DB_1)$, $global(T_1, DB_2)$, and $local(T_1, D_1)$. Suppose another transaction T_2 accesses data in domains DB_3 and DB_4. Then, $global(T_2, D_1)$, $global(T_2, D_2)$ and $Dom(T_2) = D_2$. Finally, suppose a transaction T_3 wishes to access data in DB_1 and DB_4. T_3 will not be permitted to execute, since there does not exist any domain $D \in \Delta$ such that $DB_1 \sqsubset D$ as well as $DB_4 \sqsubset D$. However, if there was a domain $D_3 = \bigcup\{D_1, D_2\}$, then the transaction T_3 would be permitted and $Dom(T_3) = D_3$.

Let $S = (\tau_S, \prec_S)$ be a schedule, where τ_S is a set of transactions and $\prec_S$ is a partial order over the operations belonging to transactions in τ_S. The partial order $\prec_S$ satisfies the property that $\prec_{T_i} \subseteq \prec_S$, for each $T_i \in \tau_S$. Let d be a set of data items. S^d denotes the projection of S onto data items in d. Formally, schedule S^d is a *restriction*[2] of the schedule S over the set of data items in d. For notational brevity, we denote the projection of S over the set of data items in DB_k; that is, S^{DB_k}, by S_k.

---

[1] In contrast, the r_i and w_i operations of the transaction on each data item are unique. Since, in this paper, we do not consider the problem

of replica control, we consider different copies of the same data item as independent data items with an equality constraint between them.

[2] A set P_1 with a partial order $\prec_{P_1}$ on its elements is a *restriction* of a set P_2 with a partial order $\prec_{P_2}$ on its elements if $P_1 \subseteq P_2$, and for all $e_1, e_2 \in P_1$, $e_1 \prec_{P_1} e_2$ if and only if $e_1 \prec_{P_2} e_2$.
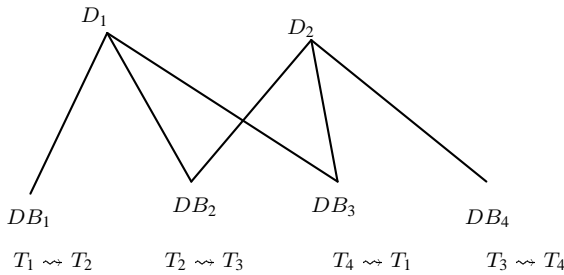
Fig. 3. Example of a non-serializable execution

In a schedule $S = (\tau_S, \prec_S)$, transactions $T_i, T_j \in \tau_S$ are said to *conflict*, denoted by $T_i \rightsquigarrow_S T_j$, if there exist operations o_i in T_i and o_j in T_j such that o_i and o_j *conflict* in S and $o_i \prec_S o_j$. Operations o_i and o_j are said to conflict if they access the same data item and at least one of them is a write operation. We denote the transitive closure of the conflict relation $\rightsquigarrow_S$ among transactions in a schedule S by the relation $\overset{*}{\rightsquigarrow}_S$.

With each domain D_i a *domain manager* $DM(D_i)$ is associated. The domain manager for a domain D_i, along with the domain managers of each domain D_j, $D_j \sqsubset D_i$, controls the concurrent execution of transactions that execute in D_i in such a way that the consistency of data within a domain is preserved. Let D be a domain such that $DB_j \sqsubset D$, $j = 1, 2, \ldots, k$. The domain managers of the domains $D' \sqsubseteq D$, in our architecture, constitute the MDBS software for an MDBS that integrates DBMS$_1$, DBMS$_2$, ..., DBMS$_k$. Note that, if there exists a simple domain $D \in \Delta$ such that for each DB_k, $k = 1, 2, \ldots, m$, $parent(D, DB_k)$, then our MDBS architecture reduces to a single monolithic system. In this case, the existing solutions for transaction management developed for such systems in Mehrotra et al. (1992a), Elmagarmid and Du (1990) and Breitbart et al. (1990) can be used by the domain manager for D to control the concurrent execution of the transactions.

4 Concurrency control in hierarchical MDBSs

In this section, we present a framework for the design of concurrency control mechanisms for hierarchical MDBSs. In a hierarchical MDBS, for the global schedule S to be serializable, the projection of S onto data items in each domain $D \in \Delta$ (that is, S^D) must be serializable. However, as illustrated in the following example, ensuring serializability of S^D, for each $D \in \Delta$, is not sufficient to ensure global serializability.

Example 2. Consider an MDBS environment consisting of local databases: DBMS$_1$ with data item a, DBMS$_2$ with data item b, DBMS$_3$ with data item c, and DBMS$_4$ with data item d. Let the domain ordering relation be as illustrated in Fig. 3. The set of domains $\Delta = \{DB_1, DB_2, DB_3, DB_4, D_1, D_2\}$, where $D_1 = \bigcup\{DB_1, DB_2, DB_3\}$, and $D_2 = \bigcup\{DB_2, DB_3, DB_4\}$. Consider the following transactions T_1, T_2, T_3, and T_4:

$$T_1 : b_{11} \; w_{11}(a) \; b_{13} \; w_{13}(c) \; c_{11} \; c_{13}$$
$$T_2 : b_{22} \; w_{22}(b) \; b_{21} \; w_{21}(a) \; c_{21} \; c_{22}$$
$$T_3 : b_{34} \; w_{34}(d) \; b_{32} \; w_{32}(b) \; c_{32} \; c_{34}$$
$$T_4 : b_{43} \; w_{43}(c) \; b_{44} \; w_{44}(d) \; c_{44} \; c_{43}$$

Note that $Dom(T_1) = D_1$, $Dom(T_2) = D_1$, $Dom(T_3) = D_2$ and $Dom(T_4) = D_2$. Consider a schedule S resulting from the concurrent execution of transactions T_1, T_2, T_3, and T_4 such that the local schedules at DBMS$_1$, DBMS$_2$, DBMS$_3$ and DBMS$_4$ are as follows:

$$S_1 : b_{11} \; w_{11}(a) \; b_{21} \; w_{21}(a) \; c_{11} \; c_{21}$$
$$S_2 : b_{22} \; w_{22}(b) \; b_{32} \; w_{32}(b) \; c_{22} \; c_{32}$$
$$S_3 : b_{43} \; w_{43}(c) \; b_{13} \; w_{13}(c) \; c_{43} \; c_{13}$$
$$S_4 : b_{34} \; w_{34}(d) \; b_{44} \; w_{44}(d) \; c_{34} \; c_{44}$$

Note that the above schedule S is not serializable, even though for, each domain $D \in \Delta$, the schedules S^D are serializable.

To ensure global serializability, besides ensuring serializability of S^D, for each $D \in \Delta$, we need to impose certain restrictions on the domain hierarchy Δ. In Sect. 5, we develop a concurrency control mechanism for ensuring serializability of the schedule S^D. In Sect. 6, we identify the required restrictions on Δ such that the mechanism for ensuring serializability of S^D (developed in Sect. 5) is sufficient to ensure global serializability. The remainder of this section is devoted to developing a design framework on which our mechanism for ensuring serializability of the schedule S^D (in Sect. 5) is based.

Before we discuss how serializability of the schedule S^D can be ensured for an arbitrary domain in the domain hierarchy, let us first consider how it can be ensured for a simple domain. Since a simple domain consists of a collection of local DBMSs, serializability of S^D can be ensured using the techniques developed for ensuring global serializability in monolithic MDBSs. Below, we develop a mechanism for ensuring serializability of S^D for simple domains, which is based on the technique for ensuring global serializability in monolithic MDBSs developed in Mehrotra et al. (1992a). Crucial to the mechanism is the notion of *serialization functions* (Mehrotra et al. 1992a), which is similar to the notion of *o-element* developed in Pu (1988) and that of the *serialization event* introduced in Elmagarmid and Du (1990).

Let $S = (\tau_S, \prec_S)$ be a serializable schedule. Let $\tau' \subseteq \tau_S$. A serialization function of a transaction $T_i \in \tau'$ in a schedule S with respect to the set of transactions τ', denoted by $ser_{S,\tau'}(T_i)$ is a function that maps $T_i \in \tau'$ to some operation in T_i such that the following holds:

For all $T_i, T_j \in \tau'$, if $T_i \overset{*}{\rightsquigarrow}_S T_j$, then $ser_{S,\tau'}(T_i) \prec_S ser_{S,\tau'}(T_j)$

In the remainder of the paper, we will denote the function $ser_{S,\tau'}$ by ser_S. The set of transactions τ' will be clear from the context. For numerous concurrency control protocols that generate serializable schedules, it is possible to associate a serialization function with transactions T_i in the schedule S such that the above property is satisfied.

For example, if the *timestamp ordering* (TO) concurrency control protocol is used to ensure serializability of S and the scheduler assigns timestamps to transactions when they begin execution, then the function that maps every transaction

$T_i \in \tau_S$ to T_i's begin operation is a serialization function for transaction T_i in S with respect to the set of transactions τ_S.

For a schedule S, there may be multiple serialization functions. For example, if S is generated by a *two-phase locking* (2PL) protocol, then a possible serialization function for transactions in S maps every transaction $T_i \in \tau_S$ to the operation that results in T_i obtaining its last lock. Alternatively, the function that maps every transaction $T_i \in \tau_S$ to the operation that results in T_i releasing its first lock is also a serialization function for T_i in S.

It is possible that, for transactions in a schedule generated by certain concurrency control protocols, no serialization function may exist. For example, in a schedule generated by a *serialization-graph-testing* (SGT) scheduler, it may not be possible to associate a serialization function with transactions. However, in such schedules, serialization functions can be introduced by forcing direct conflicts between transactions (Georgakopoulos et al. (1991). Let $\tau' \subseteq \tau$ be a set of transactions in a schedule S. If each transaction in τ' executed a conflicting operation (say a write operation on data item $ticket$) in S, then the functions that maps a transaction $T_i \in \tau'$ to its write operation on $ticket$ is the serialization function for the transactions in S with respect to the set of transactions τ'.

Associating serialization functions with global transactions makes the task of ensuring serializability of S^D relatively simple. Since at each local DBMS the order in which transactions that are global with respect to the local DBMSs are serialized is consistent with the order in which their ser_{S_k} operations execute, serializability of S^D can be ensured by simply controlling the execution order of the ser_{S_k} operations belonging to the transactions global with respect to the local DBMSs. To see how this can be achieved, for a global transaction T_i, let us denote its projection to its serialization function values over the local DBMSs as a transaction $\tilde{T}_i^D$. Formally, $\tilde{T}_i^D$ is defined as follows.

Definition 1. Let T_i be a transaction and D be a simple domain such that $global(T_i, DB_k)$, for some DB_k, where $child(DB_k, D)$, $\tilde{T}_i^D$ is a restriction of T_i consisting of all the operations in the set

$$\{ ser_{S_k}(T_i) \mid T_i \text{ executes in } DB_k,$$
$$\text{and } child(DB_k, D) \}$$

Further, for the global schedule S, we define a schedule $\tilde{S}^D$ to be the restriction of S consisting of the set of operations belonging to transactions $\tilde{T}_i^D$. Thus, $\tilde{S}^D = (\tau_{\tilde{S}^D}, \prec_{\tilde{S}^D})$, where

$$\tau_{\tilde{S}^D} = \{ \tilde{T}_i^D \mid global(T_i, DB_k) \text{ for some } DB_k,$$
$$\text{where } child(DB_k, D) \},$$

and for all operations o_q, o_r in $\tilde{S}^D$, $o_q \prec_{\tilde{S}^D} o_r$, iff $o_q \prec_S o_r$. In the schedule $\tilde{S}^D$ the conflict between operations is defined as follows:

Definition 2. Let S be a global schedule. Operations $ser_{S_k}(T_i)$ and $ser_{S_l}(T_j)$ in schedule $\tilde{S}^D$, $T_i \neq T_j$, are said to conflict if and only if $k = l$.

It is not too difficult to show that the serializability of the schedule S^D can be ensured by ensuring the serializability of the schedule $\tilde{S}^D$. Essentially, ensuring serializability of $\tilde{S}^D$ enforces a total order over global transactions (with respect to the local DBMSs), such that if T_i occurs before T_j in the total order, then ser_{S_k} operation of T_i occurs before ser_{S_k} operation of T_j for all sites s_k at which they execute in common, thereby ensuring serializability of S^D (see Mehrotra et al. 1992a for a detailed explanation).

Notice that operations in the schedule $\tilde{S}^D$ consist of only global transactions. Thus, since global transactions execute under the control of the MDBS software, the MDBS software can control the execution of the operations in $\tilde{S}^D$ to ensure its serializability, thereby ensuring serializability of S^D. How this can be achieved – that is, how the MDBS software can ensure serializability of $\tilde{S}^D$ is a topic of the next section. Recall that the above-described mechanism for ensuring serializability of S^D has been developed under the assumption that D is a simple domain. In the remainder of this section, we extend the mechanism suitably to ensure serializability of the schedule S^D for an arbitrary domain D. One way we can extend the mechanism to arbitrary domains in hierarchical MDBSs is by suitably extending the notion of the serialization function to the set of domains.

Definition 3. Let D be any arbitrary domain in Δ. An extended serialization function is a function $sf(T_i, D)$ that maps a given transaction T_i, and a domain D, to some operation of T_i that executes in D such that the following holds.

For all T_i, T_j, if $global(T_i, D)$, $global(T_j, D)$, and $T_i \overset{*}{\rightsquigarrow}_{S^D} T_j$, then $sf(T_i, D) \prec_{S^D} sf(T_j, D)$.

We refer to $sf(T_i, D)$ as a serialization function of transaction T_i with respect to the domain D. To see how such a serialization function will aid us in ensuring serializability within a domain, consider a domain $D \neq DB_k$, $k = 1, 2, \ldots, m$. To develop the intuition, let us assume that the above-defined serialization function exists for transactions in every child domain of D, that is, for every D_k, where $child(D_k, D)$. If such a serialization function can be associated with the child domains, we can simply use the mechanism developed for simple domains to ensure serializability of S^D. We will, however, have to appropriately extend our definitions of the transaction $\tilde{T}_i^D$, and the schedule $\tilde{S}^D$ with respect to the newly defined serialization function. This is done below.

Definition 4. Let T_i be a transaction and D be a domain such that $global(T_i, D_k)$ for some D_k, where $child(D_k, D)$. $\tilde{T}_i^D$ is a restriction of T_i consisting of all the operations in the set $\{ sf(T_i, D_k) \mid T_i \text{ executes in } D_k, \text{ and } child(D_k, D) \}$.

As before, schedule $\tilde{S}^D$ is simply the schedule consisting of the operations in the transactions $\tilde{T}_i^D$. That is, $\tilde{S}^D = (\tau_{\tilde{S}^D}, \prec_{\tilde{S}^D})$, where

$$\tau_{\tilde{S}^D} = \{ \tilde{T}_i^D \mid global(T_i, D_k)$$
$$\text{for some } D_k, \text{ where } child(D_k, D) \},$$

and for all operations o_q, o_r in $\tilde{S}^D$, $o_q \prec_{\tilde{S}^D} o_r$, iff $o_q \prec_S o_r$. Similar to the case of simple domain, two operations in $\tilde{S}^D$, where D is an arbitrary domain, conflict if they are both serialization function values of different transactions over the same child domain.

Definition 5. Let S be a global schedule. Operations $sf(T_i, D_k)$ and $sf(T_j, D_l)$ in schedule $\tilde{S}^D$, $T_i \neq T_j$, are said to conflict if and only if $k = l$.

It it not difficult to see that similar to the case of simple domains, serializability of S^D can be ensured, where D is an arbitrary domain, by ensuring the serializability of the schedule $\tilde{S}^D$, under the assumption that, for all child domains D_k of D, the schedule S^{D_k} is serializable and further a serialization function sf can be associated with transactions that are global with respect to D_k (see Lemma 1 in the appendix for a formal proof). In fact, this result can be applied recursively over the domain hierarchy to ensure serializability of the schedules S^D for arbitrary domains D in hierarchical MDBSs. To see this, consider a hierarchical MDBS shown in Fig. 4. To ensure serializability of S^{D_3}, it suffices to ensure serializability of the schedule $\tilde{S}^{D_3}$, under the assumption that S^{D_1} and S^{D_2} are serializable and further that an appropriate serialization function sf can be associated with transactions that are global with respect to D_1 and D_2. In turn, serializability of S^{D_1} (S^{D_2}) can be ensured by ensuring that the schedule $\tilde{S}^{D_1}$ ($\tilde{S}^{D_2}$) is serializable, under the assumption that S^{DB_1} and S^{DB_2} (S^{DB_3} and S^{DB_4}) are serializable and further that an appropriate serialization function sf can be associated with transactions that are global with respect to DB_1 and DB_2 (DB_3 and DB_4). The recursion ends when D is a simple domain, since the child domains are local DBMSs and by assumption the schedule at each local DBMS is serializable. Thus, if we can associate an appropriate serialization function sf with transactions in each domain $D \in \Delta$, we can ensure serializability of S^D, by ensuring serializability of $\tilde{S}^D$ for all domains $D \in \Delta$. Note that, for a domain $D = DB_k$, the function sf is simply the function ser_{S_k} introduced earlier. We now define the function sf for an arbitrary domain $D \in \Delta$, which is done recursively over the domain ordering relation.

Definition 6. Let D be a domain and T_i be a transaction such that $global(T_i, D)$. The serialization function for transaction T_i in domain D is defined as follows:

$$sf(T_i, D) = \begin{cases} ser_{S_k}(T_i), & \text{if for some } DB_k, \ D = DB_k. \\ ser_{\tilde{S}^D}(\tilde{T}_i^D), & \text{if for all } DB_k, \ D \neq DB_k \end{cases}$$

Let us illustrate the above definition of the serialization function using the following example.

Example 3. Consider an MDBS environment consisting of local databases: DBMS$_1$ with data item a, DBMS$_2$ with data item b, DBMS$_3$ with data item c, and DBMS$_4$ with data item d. Let the domain ordering relation be as illustrated in Fig. 4. The set of domains:

$$\Delta = \{DB_1, DB_2, DB_3, DB_4, D_1, D_2, D_3\},$$
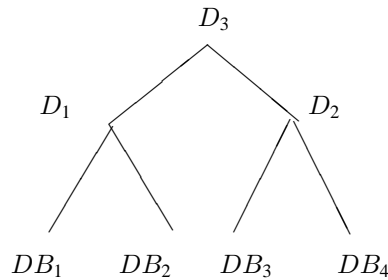

Fig. 4. Domain ordering for Example 3

where $D_1 = \bigcup\{DB_1, DB_2\}$, $D_2 = \bigcup\{DB_3, DB_4\}$, and $D_3 = \bigcup\{D_1, D_2\}$. Consider the following transactions T_1, T_2, and T_3 that execute:

$$T_1 : b_{11} \ w_{11}(a) \ b_{13} \ b_{14} \quad w_{13}(c) \ w_{14}(d) \ c_{11} \ c_{13} \ c_{14}$$
$$T_2 : b_{22} \ w_{22}(b) \ b_{24} \ b_{23} \quad w_{23}(c) \ w_{24}(d) \ c_{22} \ c_{24} \ c_{23}$$
$$T_3 : b_{31} \ w_{31}(a) \ b_{32} \ w_{32}(b) \ c_{31} \quad c_{32}$$

Note that $Dom(T_1) = D_3$, $Dom(T_2) = D_3$, and $Dom(T_3) = D_1$. Further, $global(T_1, D_1)$, $global(T_2, D_1)$ and $local(T_3, D_1)$. Similarly, $global(T_1, D_2)$, $global(T_2, D_2)$. Consider the global schedule S resulting from the concurrent execution of transaction T_1, T_2 and T_3 such that the local schedules at DBMS$_1$, DBMS$_2$, DBMS$_3$ and DBMS$_4$ are as follows:

$$S_1 : b_{11} \ w_{11}(a) \ b_{31} \quad w_{31}(a) \ c_{11} \quad c_{31}$$
$$S_2 : b_{32} \ w_{32}(b) \ b_{22} \quad w_{22}(b) \ c_{32} \quad c_{22}$$
$$S_3 : b_{13} \ b_{23} \quad w_{13}(c) \ c_{13} \quad w_{23}(c) \ c_{23}$$
$$S_4 : b_{24} \ b_{14} \quad w_{14}(d) \ w_{24}(d) \ c_{14} \quad c_{24}$$

Let the functions ser_{S_i}, $i = 1, 2, 3, 4$ be defined. Let ser_{S_1} and ser_{S_2} be functions that map transactions to their begin operation. Furthermore, let ser_{S_3} and ser_{S_4} be functions that map transactions to their commit operations. By Definition 6, $sf(T_1, DB_1) = b_{11}$, $sf(T_2, DB_2) = b_{22}$, $sf(T_3, DB_1) = b_{31}$, $sf(T_3, DB_2) = b_{32}$, $sf(T_1, DB_3) = c_{13}$, $sf(T_1, DB_4) = c_{14}$. $sf(T_2, DB_3) = c_{23}$, and $sf(T_2, DB_4) = c_{24}$. As a result, transactions $\tilde{T}_i^{D_1}$, and $\tilde{T}_i^{D_2}$, $i = 1, 2, 3$, are as follows.

$$\tilde{T}_1^{D_1} : b_{11} \qquad \tilde{T}_2^{D_1} : b_{22} \qquad \tilde{T}_1^{D_2} : c_{13} \ c_{14}$$
$$\tilde{T}_2^{D_2} : c_{24} \ c_{23} \qquad \tilde{T}_3^{D_1} : b_{31} \ b_{32}$$

The schedules $\tilde{S}^{D_1}$ and $\tilde{S}^{D_2}$ are as follows:

$$\tilde{S}^{D_1} : b_{11} \ b_{31} \quad b_{32} \ b_{22} \qquad \tilde{S}^{D_2} : c_{13} \ c_{14} \ c_{24} \ c_{23}$$

Consider the schedule $\tilde{S}^{D_2}$ that contains transactions $\tilde{T}_1^{D_2}$ and $\tilde{T}_2^{D_2}$. Let $ser_{\tilde{S}^{D_2}}$ be a function that maps the transaction $\tilde{T}_i^{D_2}$ to its first operation in $\tilde{S}^{D_2}$. That is, $ser_{\tilde{S}^{D_2}}(\tilde{T}_1^{D_2}) = c_{13}$ and $ser_{\tilde{S}^{D_2}}(\tilde{T}_2^{D_2}) = c_{24}$. The function $ser_{\tilde{S}^{D_2}}$ satisfies the requirement of the serialization function for the schedule $\tilde{S}^{D_2}$. To see this, note that, in $\tilde{S}^{D_2}$, operation c_{13} conflicts with c_{23} and c_{14} conflicts with c_{24}. As a result, transaction $\tilde{T}_1^{D_2}$ is serialized before $\tilde{T}_2^{D_2}$ in $\tilde{S}^{D_2}$. Since $ser_{\tilde{S}^{D_2}}(\tilde{T}_1^{D_2}) = c_{13}$ occurs before $ser_{\tilde{S}^{D_2}}(\tilde{T}_2^{D_2}) = c_{24}$, the function $ser_{\tilde{S}^{D_2}}$ satisfies the requirement of the serialization function for $\tilde{S}^{D_2}$. Thus, by Definition 6, $sf(T_1, D_2) = c_{13}$ and $sf(T_2, D_2) = c_{24}$. Finally, consider the schedule $\tilde{S}^{D_1}$ that contains transactions

$\tilde{T}_1^{D_1}$ and $\tilde{T}_2^{D_1}$ and $\tilde{T}_3^{D_1}$. Let $ser_{\tilde{S}^{D_1}}$ be a function that maps the transaction $\tilde{T}_i^{D_1}$ to its first operation in $\tilde{S}^{D_1}$. That is, $ser_{\tilde{S}^{D_1}}(\tilde{T}_1^{D_1}) = b_{11}$ and $ser_{\tilde{S}^{D_1}}(\tilde{T}_2^{D_1}) = b_{22}$. Note that the function $ser_{\tilde{S}^{D_1}}$ satisfies the requirements of the serialization function for the schedule $\tilde{S}^{D_1}$. Hence, by Definition 6, $sf(T_1, D_1) = b_{11}$ and $sf(T_2, D_1) = b_{22}$.

Using the above-described function sf, serializability of the the schedule S^D, $D \in \Delta$ can be ensured by simply ensuring the serializability of the schedules $\tilde{S}^D$. We state this formally in the following theorem.

Theorem 1. Consider an MDBS environment with the set Δ of domains. Let S be a global schedule and D be an arbitrary domain in Δ. Schedule S^D is serializable if the following three conditions hold:

- For each DB_k such that $DB_k \sqsubset D$, S_k is serializable and further there exists a function ser_{S_k} such that, for all transactions T_i, T_j, if $global(T_i, DB_k)$, $global(T_j, DB_k)$, and $T_i \overset{*}{\leadsto}_{S_k} T_j$, then $ser_{S_k}(T_i) \prec_S ser_{S_k}(T_j)$.
- For all domains $D' \in \Delta$ such that $D' \sqsubset D$ and $D' \neq DB_k$, $k = 1, 2, \dots, m$, $\tilde{S}^{D'}$ is serializable and further there exists a function $ser_{\tilde{S}^{D'}}$ such that, for all transactions T_i, T_j, if $global(T_i, D')$, $global(T_j, D')$, and $\tilde{T}_i^{D'} \overset{*}{\leadsto}_{\tilde{S}^{D'}} \tilde{T}_j^{D'}$, then $ser_{\tilde{S}^{D'}}(\tilde{T}_i^{D'}) \prec_S ser_{\tilde{S}^{D'}}(\tilde{T}_j^{D'})$.
- $\tilde{S}^D$ is serializable.

To see the implication of Theorem 1 consider again the execution in Example 3. In Example 3, the schedules $\tilde{S}^{D_1}$, $\tilde{S}^{D_2}$ and $\tilde{S}^{D_3}$ are as follows:

$$\tilde{S}^{D_1} : b_{11} \quad b_{31} \quad b_{32} \quad b_{22} \qquad \tilde{S}^{D_2} : c_{13} \quad c_{24} \quad c_{14} \quad c_{23}$$
$$\tilde{S}^{D_3} : b_{11} \quad b_{22} \quad c_{13} \quad c_{24}$$

Theorem 1 states that serializability of the schedule S^{D_3} can be ensured if the domain managers of domains D_1, D_2 and D_3 ensure the serializability of the schedules $\tilde{S}^{D_1}$, $\tilde{S}^{D_2}$ and $\tilde{S}^{D_3}$, respectively. Thus, our task of ensuring serializability of the schedule S^D reduces to that of developing a mechanism using which the domain manager $DM(D)$ can ensure serializability of the schedule $\tilde{S}^D$. We develop such a mechanism in the following section.

5 Ensuring serializability of S^D

Before we describe how a domain manager $DM(D)$ for an arbitrary domain $D \in \Delta$ ensures serializability of the schedule $\tilde{S}^D$, let us first discuss how the domain manager for a simple domain can ensure serializability of $\tilde{S}^D$. Later we will extend the described mechanism to the domain managers for an arbitrary domain.

5.1 Ensuring serializability in simple domains

Recall that, since the operations in $\tilde{S}^D$ belong only to global transactions, the domain managers do not need access to operations belonging to local transactions (which execute outside the control of the MDBS software) in order to ensure

serializability of $\tilde{S}^D$. The concurrency control mechanism works as follows.

Before submitting an operation belonging to a global transaction T_i for execution at the local DBMS, the domain manager for DB_k (that is, $DM(DB_k)$) determines whether or not the operation is a $ser_{S_k}(T_i)$ operation for some transaction T_i. If it is a $ser_{S_k}(T_i)$ operation, $DM(DB_K)$ forwards the operation to $DM(D)$, else, it submits the operation for execution to the local DBMS. On receipt of the $ser_{S_k}(T_i)$ operation, $DM(D)$ submits the operation to the local DBMS at site s_k for execution (via $DM(DB_k)$). $DM(D)$ controls the order in which ser_{S_k} operations corresponding to global transactions execute at the local DMBSs by controlling the order in which it submits these operations for execution to the local DBMSs. In particular, it ensures serializability of $\tilde{S}^D$ (which consists of the ser_{S_k} operations) by using a concurrency control protocol (e.g., TO, 2PL, SGT) to control the order in which it submits the operations to the local DBMS for execution.

To see how the approach works consider the execution in Example 3. Let us assume that the domain manager $DM(D_1)$ follows the 2PL protocol to control the submission order of the ser_{S_k} operations to the local DBMS for execution. Assume that T_1 requests execution b_{11} operation first. Recall that the begin operations of transactions are the ser_{S_k} operations at DBMS$_1$ and DBMS$_2$. Thus, the domain manager $DM(DB_1)$ forwards the operation b_{11} to $DM(D_1)$. Since no other transaction holds a a conflicting lock, $DM(D_1)$ submits the operations b_{11} for execution to the local DBMS for execution (via $DM(DB_1)$). Let us assume that next T_3 requests a b_{31} operation. Since b_{31} operation is the $ser_{S_1}(T_3)$ operation, the domain manager $DB(DB_1)$ forwards the operation to $DM(D_1)$. The submission of the operation will be delayed since $\tilde{T}_1^D$ holds a conflicting lock. Once $\tilde{T}_1^D$ releases the lock (according to the 2PL protocol), $DM(D_1)$ may submit b_{31} for execution.

5.2 Assumptions behind the approach

The above description of the concurrency control mechanism to ensure serializability of the schedule $\tilde{S}^D$ implicitly makes the following two assumptions:

1. The $ser_{S_k}(T_i)$ operations can be associated with each global transaction T_i for all local DBMSs.
2. The interface supported by the local DBMSs for the global transactions is such that the MDBS software submits each database operation, including the $ser_{S_k}(T_i)$ operations, explicitly for execution to the local DBMSs, and the local DBMSs acknowledge the execution of the submitted operation. We refer to such an interface as the *operation* interface.

Below, we argue that the assumption 1 is reasonable in practice. Furthermore, we argue that if assumption 2 does not hold for a particular MDBS, it remains possible to use our approach with only a minor changes, and some loss of concurrency.

The basis of the first assumption has been discussed earlier. Depending upon the concurrency control protocol followed by the local DBMS, it may or may not be possible to

associate a serialization function with the transactions. If the concurrency control scheme followed by the local DBMS is such that serialization function cannot be associated with transactions, serialization functions can be artificially introduced for global transactions by forcing every two global transactions that execute at some common sites to conflict directly at those sites. This can be accomplished by augmenting global transactions to execute a write operation on a common data item *ticket* at the site. It should always be possible to add a data item to the local DBMS, but in the case that neither the concurrency control protocol used by the local DBMS supports a serialization function, and nor does the local DBMS provide a mechanism for defining new data items, the scheme developed in this paper, as well as other approaches to concurrency control in MDBSs developed previously, will not be usable to ensure global serializability. Such a situation is extremely unlikely to occur in practice and, thus, the first assumption is reasonable from the practical standpoint.

Unfortunately, the second assumption may not be valid, since some existing local DBMSs do not support an operation interface. Instead, a local DBMS may support a *service* interface (Breitbart et al. 1992a) in which the local DBMS only permits $DM(DB_k)$ to submit a request for execution of an existing local application on behalf of the global transaction (and not the read and write operations that constitute the application). Alternatively, a local DBMS may support an *SQL* interface, that permits $DM(DB_k)$ to request multiple SQL statements (or expressions in the local data manipulation language) as part of the global subtransaction, the execution of each being acknowledged by the local DBMS. The submitted SQL query (or the service request in the case of the service interface) may result in multiple read and write operations over the data and the index structures (e.g., B-trees) maintained by the local DBMS. The domain manager $DM(DB_k)$ may be unaware of these resulting operations, as well as of the mechanisms used by the local DBMS for processing the SQL queries (e.g., protocol for B-tree traversal: Mohan and Levine 1989; key range locking for phantom protection: Lomet 1993).

If the local DBMSs do not support an operation interface, $DM(DB_k)$ does not have direct control over when $ser_{S_k}(T_i)$ operations execute at the local DBMSs. However, the relative order in which ser_{S_k} operations execute can still be controlled by controlling the submission of operations that *cause* the execution of the $ser_{S_k}(T_i)$ operation at the local DBMS. To see this, consider a local DBMS at site s_k that supports an SQL interface. Furthermore, assume that the local DBMS at s_k follows a TO protocol that assigns timestamps to transactions when they begin execution. That is, $ser_{S_k}(T_i)$ is the first database operation belonging to T_i at site s_k. $DM(D)$ can control the relative order in which $ser_{S_k}(T_i)$ operations execute at s_k by controlling the order in which it submits the first SQL query for each global transaction T_i to the local DBMS at site s_k (via the domain manager $DM(DB_k)$). This is possible since $ser_{S_k}(T_i)$ for a global transaction T_i executes only after $DM(D)$ submits the first SQL query of T_i for execution to, and before receiving an acknowledgment from, the local DBMS at s_k (via $DM(DB_k)$). Thus, if local DBMSs do not support an operation interface, our scheme can still be used

to ensure serializability for $\tilde{S}^D$ with the following modification: $DM(DB_k)$ forwards the operation that will cause the execution of $ser_{S_k}(T_i)$ at the local DBMS to $DM(D)$ for processing. $DM(D)$, in turn, forwards the operation for execution to the local DBMS (via $DM(DB_k)$). As before, $DM(D)$ uses a concurrency control protocol (e.g., TO, 2PL, SGT) to control the order in which it submits the operations to the local DBMS for execution, thereby ensuring serializability of $\tilde{S}^D$.

Notice that the nature of the interface supported by the local DBMS affects the degree of concurrency afforded by the developed approach. For example, in the case of a service interface, the entire service or the subtransaction is considered as a single operation by $DM(DB_k)$, and it forwards the request for service invocation to $DM(D)$ for execution. Since $DM(D)$ uses a concurrency control protocol (e.g., 2PL) to control the order in which it forwards the service request to the local DBMS for execution, the service request at the local DBMS causes the execution of the ser_{S_k} operation for the transaction, and the ser_{S_k} operations of two different transactions at the same site conflict, only a single service request is allowed to execute at the same DBMS at a given time. Thus, the scheme essentially results in global transactions executing sequentially at each local DBMS. In contrast, in the case of the operation interface, multiple global transactions may execute concurrently at a given time at each local DBMS as long as the concurrently executing operations are not the $ser_{S_k}(T_i)$ operations.

For the remainder of the paper, we will assume that the local DBMSs support an operation interface. This assumption is made only for the sake of simplicity of the presentation and does not compromise the generality of our solution as explained above.

5.3 Ensuring serializability in arbitrary domains

Recall that our discussion so far has considered only the design of the domain manager $DM(D)$, for simple domains. We now turn our attention to the design of the domain manager for an arbitrary domain $D \in \Delta$ such that $D \notin TOP$ and $D \neq DB_k$, for all $k = 1, 2, \ldots, m$. $DM(D)$ consists of the following three components $DM_1(D)$, $DM_2(D)$ and $DM_3(D)$ as illustrated in Fig. 5:

– $DM_1(D)$: The component $DM_1(D)$ receives the $sf(T_i, D_k)$ operations belonging to transactions $\tilde{T}_i^D$ from the domain manager of each domain D_k at which T_i executes, where $child(D_k, D)$. $DM_1(D)$, on receipt of the operation $sf(T_i, D_k)$, determines if the transaction T_i is local to D. If T_i is local to D (that is, $local(T_i, D)$), then $DM_1(D)$ forwards the operation $sf(T_i, D_k)$ to $DM_3(D)$ for processing. Else, if T_i is global to D (that is, $global(T_i, D)$) and further if the operation $o = sf(T_i, D_k)$ is also the serialization function value of T_i with respect to the domain D, that is, $o = sf(T_i, D)$, then $DM_1(D)$ submits the operation to the domain managers of every domain D' such that $parent(D', D)$ for processing[3]. Else, if $o \neq sf(T_i, D)$, then it submits the

---

[3] Recall that a domain D, in our MDBS architecture, may have multiple domains D' such that $parent(D', D)$.
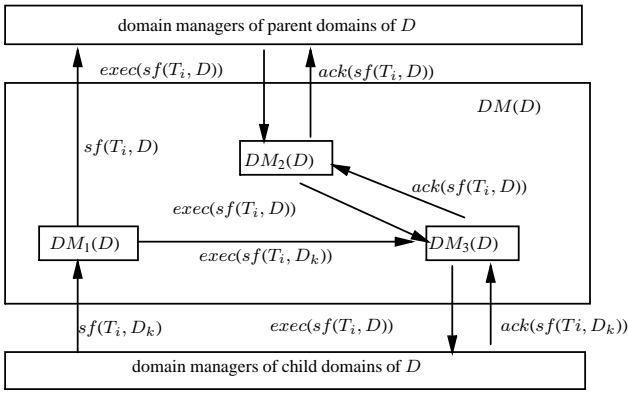
Fig. 5. Components of a domain manager

operation to $DM_3(D)$. The component $DM_3(D)$ is responsible for ensuring serializability of $\tilde{S}^D$.

- $DM_2(D)$**:** The component $DM_2(D)$ receives requests for the execution of the operations $o = sf(T_i, D)$ (that is, $exec(sf(T_i, D))$) requests) from the domain managers of the domains D', where $parent(D', D)$. In case there are multiple domains D' such that $parent(D', D)$, $DM_2(D)$ waits until it receives requests $exec(sf(T_i, D))$ from each domain D', where $parent(D', D)$. On receipt of the request from *each* of the parent domains, it submits the operation for execution to the component $DM_3(D)$. On receipt of the acknowledgement for the successful execution of the operation $sf(T_i, D)$ (denoted by $ack(sf(T_i, D))$) from $DM_3(D)$, $DM_2(D)$, in turn, forwards the acknowledgement to the domain managers of each of the domains D', where $parent(D', D)$.

- $DM_3(D)$**:** The component $DM_3(D)$ is responsible for scheduling the operations of the transactions $\tilde{T}_i^D$ to the local DBMSs for execution (via the domain managers of the child domains of D) in such a fashion that the schedule $\tilde{S}^D$ is serializable. $DM_3(D)$ receives request for the execution of operations $o = sf(T_i, D_k)$, where $child(D_k, D)$ from $DM_1(D)$ (if either o belongs to a transaction T_i such that $local(T_i, D)$, or if $o \neq sf(T_i, D)$) and from the component $DM_2(D)$ (if $global(T_i, D)$ and furthermore the operation $o = sf(T_i, D_k)$ is also the operation $sf(T_i, D)$). $DM_3(D)$, in turn, submits the request for the execution of the operation $sf(T_i, D_k)$, to the domain manager of the domain D_k, where $child(D_k, D)$. Further, on receipt of the acknowledgement for the operation $o = sf(T_i, D_k)$ (that is, $ack(sf(T_i, D_k))$) from the domain manager of the domain D_k, in case the operation is also the serialization function of T_i with respect to D (that is, $sf(T_i, D)$), $DM_3(D)$ forwards the acknowledgement to the component $DM_2(D)$ which, as mentioned previously, acknowledges the execution of the operation to the domain managers of each of the parent domains of D. $DM_3(D)$ controls the submission order of the operations $sf(T_i, D_k)$ to the domain managers of the domains D_k, where $child(D_k, D)$, in such a fashion that the schedule $\tilde{S}^D$ is serializable.

Above, we have described the components of the domain manager for a domain D, where $D \notin TOP$ and

further $D \neq DB_k$, $k = 1, 2, \ldots, m$. The domain manager for the domain $D \in TOP$ differs from the above in that it does not contain the component $DM_2(D)$. Note that if $D \in TOP$, then there does not exist a domain D' such that $parent(D', D)$. Thus, the component $DM_1(D)$ of the domain manager for a domain $D \in TOP$, on receipt of the any operations $o = sf(T_i, D_k)$, where $child(D_k, D)$, submits a request for the execution of $sf(T_i, D_k)$ (that is, $exec(sf(T_i, D_k))$) to the component $DM_3(D)$ directly.

Finally, we consider the domain manager for a domain $D = DB_k$, for some $k = 1, 2, \ldots, m$. The domain manager for a domain $D = DB_k$ is responsible for forwarding operation $sf(T_i, DB_k)$ (that is, the operations $ser_{S_k}(T_i)$), where $global(T_i, DB_k)$ to the domain manager for the parent domains of DB_k. Furthermore, $DM(DB_k)$, on request for the execution of the $sf(T_i, DB_k)$ operations from the parent domains of DB_k submits the operation for execution to the local DBMS. Finally, on receipt of an acknowledgement from the local DBMS for the execution of the $sf(T_i, DB_k)$ operation, it forwards the acknowledgement to the domain managers of the parent domains of D. Thus, the domain manager for a domain $D = DB_k$ differs from the domain manager defined above in that it does not contain a component $DM_3(D)$.

In our design of the domain manager for a domain D, the operation $o = sf(T_i, D_k)$, where $child(D_k, D)$, does not execute in S until the component $DM_3(D)$ of the domain manager for domain D submits a request for the execution of the operation $sf(T_i, D_k)$, that is, $exec(sf(T_i, D_k))$ to the domain manager of domain D_k. Note that this is true, since the component $DM_2(D_k)$ of the domain manager for the child domain D_k waits to receive a request for the execution of the operation $sf(T_i, D_k)$ from each parent domain of D_k. Furthermore, for each operation $sf(T_i, D_k)$, the component $DM_3(D)$ of the domain manager for the domain D receives the acknowledgement for the execution of $sf(T_i, D_k)$, where $child(D_k, D)$, sometime after the execution of $sf(T_i, D_k)$ in S. This is true, since we assume that each DBMS$_j$ acknowledges the execution of the operations belonging to the transactions that are global with respect to DB_j to the domain manager of $D = DB_j$, and the domain manager for each domain D, in turn, acknowledges the execution of the operation $sf(T_i, D)$, to the domain managers of each of its parent domains. Thus, the operation $sf(T_i, D_k)$ executes in S after $DM_3(D)$ submits $sf(T_i, D_k)$ for execution to the domain manager of D_k, and before $DM_3(D)$ receives the acknowledgement for the execution of $sf(T_i, D_k)$ from the domain manager of D_k. Hence, to ensure that the schedule $\tilde{S}^D$ is serializable, the component $DM_3(D)$ can use any concurrency control protocol that ensures serializability (e.g., 2PL, TO, SGT) to schedule the submission of the operations belonging to transactions $\tilde{T}_i^D$ to the domain managers of the child domains. Note that, since the schedule $\tilde{S}^D$ is distributed over the domains $D_1, D_2, \ldots, D_k$, where $child(D_j, D)$, $j = 1, 2, \ldots, k$, $DM_3(D)$ can follow any distributed or centralized concurrency control protocol to ensure serializability of $\tilde{S}^D$.
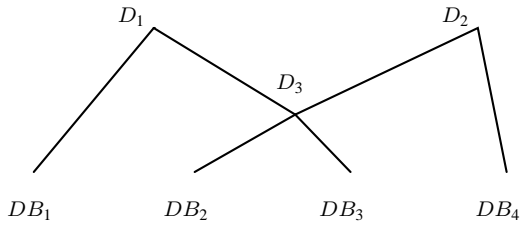
Fig. 6. Example of Δ that satisfies **R1**

6 Ensuring global serializability

In the previous section, we developed a mechanism that the domain managers can use to ensure that the projection of the schedule to their domains is serializable. However, as we mentioned at the beginning of Sect. 4, ensuring serializability of the schedules S^D, $D \in \Delta$ alone may not guarantee global serializability in a hierarchical MDBS (see Example 2). To ensure global serializability, the set of domains Δ must be restricted appropriately. In the remainder of this section, we consider a restriction on Δ such that, if the mechanism developed in the previous section for ensuring serializability of S^D, $D \in \Delta$ is used, then the resulting global schedule is serializable. That is, we identify the required restrictions on Δ such that, if each domain manager ensures serializability of $\tilde{S}^D$, then the resulting global schedule S is serializable.

To identify the appropriate restriction on Δ, let us reexamine the non-serializable execution in Example 2. Let us assume that each $DBMS_i$, $i = 1, 2, 3, 4$, follows a timestamp scheme for concurrency control in which a timestamp is assigned to a transaction when it begins execution. Since each local DBMS follows the timestamp scheme and the timestamp is assigned to a transaction when it begins execution, the serialization function for a transaction with respect to DB_i, $i = 1, 2, 3, 4$, is the transaction's begin operation at the local DBMSs. Thus, the transactions $\tilde{T}_i$ for the transactions T_1, T_2, T_3, T_4 with respect to each of the domains D_1 and D_2 are as follows:

$\tilde{T}_1^{D_1} : b_{11}\ b_{13}$ $\tilde{T}_2^{D_1} : b_{22}\ b_{21}$ $\tilde{T}_3^{D_1} : b_{32}$ $\tilde{T}_4^{D_1} : b_{43}$
$\tilde{T}_1^{D_2} : b_{13}$ $\tilde{T}_2^{D_2} : b_{22}$ $\tilde{T}_3^{D_2} : b_{34}\ b_{32}$ $\tilde{T}_4^{D_2} : b_{43}\ b_{44}$

The schedules $\tilde{S}^{D_1}$ and $\tilde{S}^{D_2}$ for the schedule S in Example 2 are as follows:

$\tilde{S}^{D_1} : b_{22}\ b_{43}\ b_{11}\ b_{32}\ b_{21}\ b_{13}$
$\tilde{S}^{D_2} : b_{22}\ b_{43}\ b_{34}\ b_{32}\ b_{13}\ b_{44}$

In schedule $\tilde{S}^{D_1}$ operations b_{11}, b_{21}, operations b_{22}, b_{32}, and operations b_{43}, b_{13} conflict. Thus, $\tilde{S}^{D_1}$ is serializable in the order $\tilde{T}_4^{D_1}$, $\tilde{T}_1^{D_1}$, $\tilde{T}_2^{D_1}$, $\tilde{T}_3^{D_1}$. Similarly, in the schedule $\tilde{S}^{D_2}$ operations b_{22}, b_{32}, operations b_{43}, b_{13}, and operations b_{34}, b_{44} conflict. Thus, $\tilde{S}^{D_2}$ is serializable in the order $\tilde{T}_2^{D_2}$, $\tilde{T}_3^{D_2}$, $\tilde{T}_4^{D_2}$, $\tilde{T}_1^{D_2}$. Since both $\tilde{S}^{D_1}$ and $\tilde{S}^{D_2}$ are serializable, the execution in Example 2 could have been generated (if, for example, the domain managers of D_1 and D_2 were following the SGT scheme to ensure serializability of $\tilde{S}^{D_1}$ and $\tilde{S}^{D_2}$, respectively). Note that, in the execution, the domain manager of D_1 serialized the transaction $\tilde{T}_1^{D_1}$ before $\tilde{T}_3^{D_1}$. In contrast, the domain manager of D_2 serializes $\tilde{T}_3^{D_2}$ before transaction $\tilde{T}_1^{D_2}$, thereby resulting in the loss of serializability. If, however, there existed a domain $D_3 = \bigcup \{DB_2, DB_3\}$ (illus-

trated in Fig. 6), then the order in which the domain manager for domain D_1 serializes transaction $\tilde{T}_i^{D_1}$ and $\tilde{T}_j^{D_1}$, and the order in which the domain manager of D_2 serializes transactions $\tilde{T}_i^{D_2}$ and $\tilde{T}_j^{D_2}$ must be the same (identical to the order in which the domain manager D_3 serializes the transactions). Hence, if there existed a domain $D_3 = \bigcup \{DB_2, DB_3\}$, then the non-serializable execution in Example 2 would not result. We therefore consider the following restriction on the set Δ of domains:

R1. *For all domains* $D_i, D_j \in TOP$, *there exists a* $D_k \in \Delta$, *such that* $D_k = D_i \cap D_j$.

In the domain ordering relation illustrated in Fig. 3, since $DB_2 \sqsubset D_1$, $DB_2 \sqsubset D_2$, and $DB_3 \sqsubset D_1$, $DB_3 \sqsubset D_2$, the domain $D_1 \cap D_2$ does not exist. Thus, the corresponding set Δ does not satisfy **R1**. In contrast, in the domain ordering relation illustrated in Fig. 6, the domain $D_3 = D_1 \cap D_2$. Thus, the corresponding set Δ satisfies the restriction **R1**.

Unfortunately, even if the set of domain Δ satisfies the restriction **R1**, and each domain manager ensures serializability of the schedule $\tilde{S}^D$, the resulting global schedule may still not be serializable. To see this let us consider the following example.

Example 4. Consider an MDBS environment consisting of local databases: $DBMS_1$ with data item a, $DBMS_2$ with data item b, and $DBMS_3$ with data item c. Let the domain ordering be as illustrated in Fig. 7. The set of domains $\Delta = \{DB_1, DB_2, DB_3, D_1, D_2, D_3\}$, where $D_1 = \bigcup \{DB_1, DB_2\}$, $D_2 = \bigcup \{DB_2, DB_3\}$, and $D_3 = \bigcup \{DB_1, DB_3\}$. Further, the set $TOP = \{ D_1, D_2, D_3 \}$, $D_1 \cap D_2 = DB_2$, $D_2 \cap D_3 = DB_3$, and $D_1 \cap D_3 = DB_1$. Hence, Δ satisfies the restriction **R1**. Consider the following transactions T_1, T_2, and T_3 that execute:

$T_1 : b_{11}\ w_{11}(a)\ b_{13}\ w_{13}(c)\ c_{11}\ c_{13}$
$T_2 : b_{21}\ w_{21}(a)\ b_{22}\ w_{22}(b)\ c_{21}\ c_{22}$
$T_3 : b_{32}\ w_{32}(b)\ b_{33}\ w_{33}(c)\ c_{32}\ c_{34}$

Note that $Dom(T_1) = D_3$, $Dom(T_2) = D_1$, and $Dom(T_3) = D_2$. Suppose that each local DBMS follows a timestamp scheme for concurrency control in which a timestamp is assigned to a transaction when it begins execution. Then, the serialization function for a transaction with respect to DB_i, $i = 1, 2, 3$, is the transactions' begin operation at the local DBMSs. Thus, the transactions $\tilde{T}_i$ for the transactions T_1, T_2, T_3 with respect to each of the domains D_1, D_2 and D_3 are as follows:

$\tilde{T}_1^{D_1} : b_{11}$ $\tilde{T}_2^{D_1} : b_{21}\ b_{22}$ $\tilde{T}_3^{D_1} : b_{32}$
$\tilde{T}_1^{D_2} : b_{13}$ $\tilde{T}_2^{D_2} : b_{22}$ $\tilde{T}_3^{D_2} : b_{32}\ b_{33}$
$\tilde{T}_1^{D_3} : b_{11}\ b_{13}$ $\tilde{T}_2^{D_3} : b_{21}$ $\tilde{T}_3^{D_3} : b_{33}$

Consider a schedule S resulting from the concurrent execution of transactions T_1, T_2, and T_3 such that the local schedules at $DBMS_1$, $DBMS_2$, $DBMS_3$ are as follows:

$S_1 : b_{11}\ w_{11}(a)\ b_{21}\ w_{21}(a)\ c_{11}\ c_{21}$
$S_2 : b_{22}\ w_{22}(b)\ b_{32}\ w_{32}(b)\ c_{22}\ c_{32}$
$S_3 : b_{33}\ w_{33}(c)\ b_{13}\ w_{13}(c)\ c_{33}\ c_{13}$

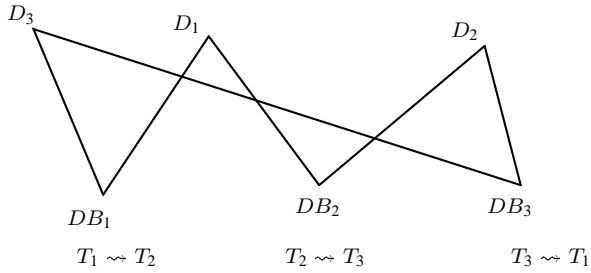Furthermore let the schedules $\tilde{S}^{D_1}$, $\tilde{S}^{D_2}$ and $\tilde{S}^{D_3}$ be as follows:

Fig. 7. A domain ordering with a cyclic DG

Fig. 8. A domain ordering such that LDG contains no undesirable cycles

$$\tilde{S}^{D_1} : b_{11}\ b_{21}\ b_{22}\ b_{32}$$
$$\tilde{S}^{D_2} : b_{22}\ b_{32}\ b_{33}\ b_{13}$$
$$\tilde{S}^{D_3} : b_{11}\ b_{21}\ b_{33}\ b_{13}$$

In schedule $\tilde{S}^{D_1}$ operations b_{11}, b_{21}, and operations b_{22}, b_{32}, conflict. Thus, $\tilde{S}^{D_1}$ is serializable in the order $\tilde{T}_1^{D_1}$, $\tilde{T}_2^{D_1}$, $\tilde{T}_3^{D_1}$. In the schedule $\tilde{S}^{D_2}$ operations b_{22}, b_{32}, and operations b_{33}, b_{13} conflict. Thus, $\tilde{S}^{D_2}$ is serializable in the order $\tilde{T}_2^{D_2}$, $\tilde{T}_3^{D_2}$, $\tilde{T}_1^{D_2}$. Similarly, in the schedule $\tilde{S}^{D_3}$ operations b_{11}, b_{21}, and operations b_{33}, b_{13} conflict. Thus, $\tilde{S}^{D_3}$ is serializable in the order $\tilde{T}_3^{D_3}$, $\tilde{T}_1^{D_3}$, $\tilde{T}_2^{D_3}$. Thus, each schedule $\tilde{S}^{D_1}$, $\tilde{S}^{D_2}$ and $\tilde{S}^{D_3}$ is serializable. However, the global schedule S is not serializable.

The above example illustrates that, even if Δ satisfies the restriction **R1**, ensuring serializability of $\tilde{S}^D$ for each domain D may not ensure global serializability. To identify conditions under which global serializability is ensured, we need to introduce the notion of a *domain graph*. A domain graph (DG) for a set of domains Δ, is an undirected graph whose nodes correspond to the set of domains $D \in TOP$. Let D_i and D_j be two nodes in DG. There is an edge (D_i, D_j) in DG if there exists a domain $D_k \in \Delta$ such that $D_k \sqsubset D_i$ and $D_k \sqsubset D_j$.

Theorem 2. Consider an MDBS environment with the set Δ of domains. Let S be a global schedule. Further, let each of the following three hold:

- For each DB_k such that $DB_k \sqsubset D$, S_k is serializable and further there exists a function ser_{S_k} such that for all transactions T_i, T_j, $global(T_i, DB_k)$, $global(T_j, DB_k)$, and $T_i \overset{*}{\leadsto}_{S_k} T_j$, then $ser_{S_k}(T_i) \prec_S ser_{S_k}(T_j)$.
- For all domains $D \in \Delta$ such that $D \notin TOP$, $\tilde{S}^D$ is serializable and further there exists a function $ser_{\tilde{S}^D}$ such that for all transactions T_i, T_j, if $global(T_i, D)$, $global(T_j, D)$, and $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$, then $ser_{\tilde{S}^D}(\tilde{T}_i^D) \prec_S ser_{\tilde{S}^D}(\tilde{T}_j^D)$.
- For all domains $D \in \Delta$ such that $D \in TOP$, $\tilde{S}^D$ is serializable.

If Δ satisfies **R1** and the DG is acyclic, then S is serializable.

Note that Theorem 2 states that, under the hypothesis of Theorem 1, global serializability is ensured if the domain hierarchy satisfies the restriction **R1** and the domain gr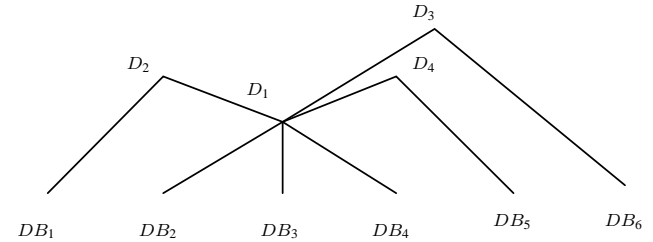aph DG does not contain any cycles. The DG for the set of domains Δ corresponding to the domain ordering relation illustrated in Fig. 6 contains nodes D_1 and D_2 and an edge (D_1, D_2). Since this DG is acyclic and the set of domains Δ satisfies **R1**, it follows that, in order to ensure global serializability, it suffices to ensure that the schedules $\tilde{S}^D$, for each domain $D \in \Delta$, is serializable. In contrast, the DG for the set of domains corresponding to the domain ordering relation illustrated in Fig. 7 contains a cycle (D_1, D_2), (D_2, D_3) and (D_3, D_1). Hence, even if for each domain $D \in \Delta$, the schedule $\tilde{S}^D$ is serializable and the set of domains Δ satisfies restriction **R1**, loss of global serializability may result.

The restriction imposed on the domain hierarchy in Theorem 2 can be relaxed since not every cycle in the domain graph DG would result in a potential loss of serializability. Consider, for example, DG for the set of domains corresponding to the domain ordering relation illustrated in Fig. 8. Note that DG contains a cycle (D_2, D_3), (D_3, D_4), (D_4, D_2). However, for the set of domains corresponding to the domain ordering relation illustrated in Fig. 8, if for each $D \in \Delta$, the domain manager for D ensures that the schedule $\tilde{S}^D$ is serializable, then it can be shown that the resulting global schedule S is serializable. Thus, certain cycles in DG do not result in a potentially non-serializable global schedule. Below we formalize the nature of the cycles that can be permitted in DG. To do so, we first introduce the notion of the *labeled domain graph* (LDG).

An LDG is a domain graph in which each edge (D_i, D_j) has a label, referred to as $label(D_i, D_j)$, where $label(D_i, D_j) = D_i \cap D_j$. Let $(D_1, D_2), (D_2, D_3), \ldots, (D_{r-1}, D_r), (D_r, D_1)$ be a cycle in the LDG. We refer to the cycle in the LDG as a *undesirable cycle* if and only if for all k, l, $k = 1, 2, \ldots, r$, $l = 1, 2, \ldots, r$, if $k \neq l$, then $label(D_k, D_{(k+1)mod\ r}) \neq label(D_l, D_{(l+1)mod\ r})$. Note that the LDG for the set of domains corresponding to the domain ordering relation illustrated in Fig. 8, has edges (D_2, D_3), (D_3, D_4) and (D_4, D_2), where $label(D_2, D_3) = label(D_3, D_4) = label(D_4, D_2) = D_1$. Thus, LDG does not contain any undesirable cycles. In contrast, the LDG for the set of domains corresponding to the domain ordering illustrated in Fig. 7 contains a cycle (D_1, D_2), (D_2, D_3), (D_3, D_1), where $label(D_1, D_2) = DB_2$, $label(D_2, D_3) = DB_3$, $label(D_3, D_1) = DB_1$. Hence, LDG contains an undesirable cycle. If the LDG for the set of domains Δ does not contain any undesirable cycles, then ensuring that $\tilde{S}^D$, for each domain $D \in \Delta$ would ensure global serializability as is stated in the following theorem.

Theorem 3. Consider an MDBS environment with the set Δ of domains. Let S be a global schedule. Further, let each of the following three hold:

– For each DB_k such that $DB_k \sqsubset D$, S_k is serializable and further there exists a function ser_{S_k} such that, for all transactions T_i, T_j, $global(T_i, DB_k)$, $global(T_j, DB_k)$, and $T_i \overset{*}{\leadsto}_{S_k} T_j$, then $ser_{S_k}(T_i) \prec_S ser_{S_k}(T_j)$.

– For all domains $D \in \Delta$, such that $D \notin TOP$, $\tilde{S}^D$ is serializable and further there exists a function $ser_{\tilde{S}^D}$ such that, for all transactions T_i, T_j, if $global(T_i, D)$, $global(T_j, D)$, and $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$, then $ser_{\tilde{S}^D}(\tilde{T}_i^D) \prec_S ser_{\tilde{S}^D}(\tilde{T}_j^D)$.

– For all domains $D \in \Delta$ such that $D \in TOP$, $\tilde{S}^D$ is serializable.

If Δ satisfies **R1** and LDG contains no undesirable cycles, then S is serializable.

Since the domain graph DG contains a cycle if and only if the labeled domain graph LDG contains a cycle, Theorem 3 relaxes the requirement in Theorem 2 for DG to be acyclic to the requirement that LDG does not contain any undesirable cycles for our scheme to ensure global serializability.

7 Related work

Relevant related work on transaction management in MDBSs was discussed in Sect. 2. As we mentioned there, hierarchical transaction management architecture for MDBSs has previously been studied in Pu (1988), where the author proposed a *superdatabase* architecture for MDBSs. In this section, we compare our approach with the concurrency control scheme developed in Pu (1988) for the superdatabase architecture. Furthermore, we discuss the relationship of our work with the hierarchical concurrency control schemes that have been studied in the context of multilevel transactions (Weikum and Schek 1984, 1991; Beeri et al. 1988).

A superdatabase can be seen as a hierarchical MDBS with the following restriction on Δ:

For all domains $D_i, D_j \in \Delta$, if $child(D_i, D_j)$, then, for all $D_k \neq D_j$, $\neg child(D_i, D_k)$.

An example of a superdatabase architecture is the domain ordering illustrated in Fig. 4. It is easy to see that a superdatabase is a special instance of a hierarchical MDBS for which the set of domains Δ satisfies the restriction **R1** and, further, the domain graph corresponding to Δ is acyclic. Thus, from Theorem 2, it follows that a concurrency control scheme based on ensuring the serializability of $\tilde{S}^D$ for each domain $D \in \Delta$ can be used in superdatabases to ensure global serializability.

The concurrency control scheme for superdatabases developed in Pu (1988) is very different from our approach. In contrast to our scheme, where, for each domain D, the domain manager $DM(D)$ ensures serializability of the schedule $\tilde{S}^D$, Pu (1988) developed a protocol, referred to as the *hierarchical validation*, in order to ensure global serializability. In the hierarchical validation protocol, the domain manager for a domain D, $D \notin TOP$, for each transaction T_i such that $\neg local(T_i, DB_k)$, $k = 1, 2, \ldots, m$, (that is, transactions that are not local to any local DBMS), submits the operations

$\{ser_{S_k}(T_i) \mid DB_k \sqsubseteq D$ and T_i accesses data item in $DB_k\}$

to the domain manager of the domain D', where $parent(D', D)$[4]. Further, the domain manager $DM(D)$ of a domain D considers any two operations $ser_{S_k}(T_i)$ and $ser_{S_l}(T_j)$, belonging to transactions T_i and T_j, to conflict if $k = l$, and uses SGT certification (Bernstein et al. 1987) to ensure the serializability of the projection of the schedule S to the operations

$$\{ser_{S_k}(T_i) \mid DB_k \sqsubseteq D \text{ and } T_i \text{ accesses data item in } DB_k\}.$$

Notice that, unlike our approach in which different domain managers could follow different concurrency control protocols (centralized or distributed), in hierarchical validation, each domain manager follows the SGT certification protocol. However, this is not the only difference between the two approaches. A more subtle and a very important difference is that in the case of the hierarchical validation, the domain manager D submits the $ser_{S_k}(T_i)$ operations to the parent domain manager irrespective of whether or not the transaction is local to D. For example, if hierarchical validation is used to control execution in Example 3, even though the transaction T_3 executes only at DB_1 and DB_2 and is local to the domain D_1, its serialization operations $ser_{S_1}(T_3)$ and $ser_{S_2}(T_3)$ will be forwarded to the domain manager $DM(D_3)$. Thus, the execution of transactions that are local to a domain D_1 will not only be controlled by the concurrency control protocols followed by the domain manager of domain D_1 (and the domain managers of the descendent domains of D_1 at which T_3 executes), but also by the concurrency control protocols followed by the domain managers of all the ancestor domains of D_1. In particular, the domain manager of the root domain in a superdatabase will control the concurrent execution of all the transactions that are global with respect to any local DBMS.

If, in the hierarchical validation protocol, $DM(D)$ does not submit the operations $ser_{S_k}(T_i)$, where $local(T_i, D)$, to the parent domain of D, then the protocol may not ensure global serializability. We illustrate this in the following example.

Example 5. Consider an MDBS environment consisting of local databases: DBMS$_1$ with data item a, DBMS$_2$ with data item b, DBMS$_3$ with data item c, and DBMS$_4$ with data item d. Let the domain ordering relation be as illustrated in Fig. 4. The set of domains:

$\Delta = \{DB_1, DB_2, DB_3, DB_4, D_1, D_2, D_3\}$,

where $D_1 = \bigcup\{DB_1, DB_2\}$, $D_2 = \bigcup\{DB_3, DB_4\}$, and $D_3 = \bigcup\{D_1, D_2\}$. Note that the set of domains Δ conforms to the superdatabase architecture. Consider the following transactions T_1, T_2, T_3 and T_4 that execute:

$T_1 : b_{11}\ w_{11}(a)\ b_{13}\ w_{13}(c)\ c_{11}\ c_{13}$
$T_2 : b_{22}\ w_{22}(b)\ b_{24}\ w_{24}(d)\ c_{22}\ c_{24}$
$T_3 : b_{31}\ w_{31}(a)\ b_{32}\ w_{32}(b)\ c_{31}\ c_{32}$
$T_4 : b_{43}\ w_{43}(c)\ b_{44}\ w_{44}(d)\ c_{43}\ c_{44}$

---

[4] Note that in the superdatabases each domain may have at most one *parent*.

Note that $Dom(T_1) = D_3$, $Dom(T_2) = D_3$, $Dom(T_3) = D_1$ and $Dom(T_4) = D_2$. Further, $global(T_1, D_1)$, $global(T_2, D_1)$ and $local(T_3, D_1)$. Similarly, $global(T_1, D_2)$, $global(T_2, D_2)$ and $local(T_4, D_2)$. Suppose that each local DBMS follows a timestamp scheme for concurrency control in which a timestamp is assigned to a transaction when it begins execution. Consider a schedule S resulting from the concurrent execution of transactions T_1, T_2, T_3, and T_4 such that the local schedules at DBMS$_1$, DBMS$_2$, DBMS$_3$ and DBMS$_4$ are as follows:

$S_1 : b_{11}\ w_{11}(a)\ b_{31}\ w_{31}(a)\ c_{11}\ c_{31}$
$S_2 : b_{32}\ w_{32}(b)\ b_{22}\ w_{22}(b)\ c_{32}\ c_{22}$
$S_3 : b_{43}\ w_{43}(c)\ b_{13}\ w_{13}(c)\ c_{43}\ c_{13}$
$S_4 : b_{24}\ w_{24}(d)\ b_{44}\ w_{44}(d)\ c_{24}\ c_{44}$

Note that the above schedule S is not serializable. However, it could have been generated even if a hierarchical validation protocol is used for concurrency control in case the domain manager $DM(D_1)$ does not submit operations belonging to the transaction T_3 to $DM(D_3)$, and $DM(D_2)$ does not submit operations belonging to the transaction T_4 to $DM(D_3)$ since $local(T_3, D_1)$ and $local(T_4, D_2)$. To see this, consider that each domain manager follows an SGT certification protocol. Since $DM(D_1)$ does not submit the operations of T_3 (that is, $ser_{S_1}(T_3)$ and $ser_{S_2}(T_3)$ operations), and $DM(D_2)$ does not submit the operations of T_4 (that is, $ser_{S_3}(T_4)$ and $ser_{S_4}(T_4)$ operations) to $DM(D_3)$ the schedules at the domain managers $DM(D_1)$, $DM(D_2)$, and $DM(D_3)$, denoted by S^1, S^2, and S^3, respectively, are as follows:

$S^1 : b_{11}\ b_{31}\ b_{32}\ b_{22}$
$S^2 : b_{24}\ b_{43}\ b_{44}\ b_{13}$
$S^3 : b_{11}\ b_{22}\ b_{24}\ b_{13}$

In the above schedules note that operations b_{11}, b_{31}, operations b_{32}, b_{22}, operations b_{24}, b_{44}, and operations b_{43}, b_{13} conflict. Thus, each of the schedules S^1, S^2 and S^3 are serializable. Hence, the schedule S would be permitted by the hierarchical validation protocol even though it is not serializable.

In contrast, in our approach, for a transaction T_i and a domain D such that $local(T_i, D)$, the domain manager of D does not submit any information to the parent domain of D and the execution of the operations of T_i is controlled by only the domain manager for D (and its descendents on which T_i executes). For example, in the execution in Example 5, the execution of the operation b_{31} and b_{32} (which are the serialization function values of T_3) will be controlled by only the domain manager of D_1 (and its descendents) and not by the domain manager of D_3, since T_3 is local to the D_1. Similarly, since T_4 is local to D_2, the execution of the operations b_{43} and b_{44} is controlled by only the domain manager of D_2 (and its descendents). Not only does this increase scalability of our approach, but it also preserves the autonomy of the individual MDBSs, since the transactions local to a domain D are controlled by only the domain manager of D (and its descendents). Furthermore, since, in our approach, only transactions global with respect to a domain D pay the overhead of the concurrency control at the parent domain of D, our approach will have better performance as compared to the hierarchical validation protocol in which every transaction that is global with respect to some local DBMS pays the concurrency control overhead at multiple levels.

Hierarchical concurrency control schemes have also been previously studied in the context of multilevel transactions (Weikum and Schek 1984, 1991; Beeri et al. 1988). A multilevel transaction is a special type of a nested transaction (Moss 1987; Gray and Reuter 1993) in which levels of the transaction represent a hierarchy of abstract data types. Operations at a given level i are implemented completely using operations at the next lower level $i - 1$. The recursion stops at level 0, the operations at which are assumed to be atomic and indivisible. In multilevel systems, concurrency control is done hierarchically at each level. The concurrency control scheme at level i, ensures isolation of the level i operation under the assumption that the level $i - 1$ operations are atomic, which are made atomic by the concurrency control scheme of level $i - 1$, and so on. The objective is that viewing transactions in the multilevel framework allows for the exploitation of the application semantics to enhance concurrency. Two transactions, even though they result in operations that conflict at the lower abstraction level may not conflict at the higher level. To see this, consider a two-level system in which transactions are implemented as a sequence of operations on tuples, and each tuple-level operation is implemented as a sequence of page-level operations. Two transactions that access/modify different tuples on the same page, even though they execute conflicting page-level operations, do not conflict at the level of the tuples. Multilevel concurrency control enables such transactions to execute concurrently, thereby enhancing concurrency.

Besides the fact that both our approach for MDBS environments, as well as the mechanisms developed for multilevel transactions, are hierarchical concurrency control protocols, there is not much similarity between them. For example, in multilevel concurrency control schemes, each transaction is defined over a given abstraction hierarchy, and its execution is controlled by the concurrency control scheme at each level. There is no concept similar to local and global transactions as is the case with transactions in MDBS environments. Furthermore, in the MDBS environments considered in this paper, the hierarchy of MDBSs represents only a structural hierarchy, and there is no implied hierarchy of abstractions as is the case with multilevel transactions.

Note that we are not claiming that the notion of multilevel transactions is orthogonal to the MDBS transaction management problem. In fact, one of the proposals (Sheck et al. 1991; Weikum et al. 1991) for concurrency control in MDBSs is to consider global transactions as two-level transactions in which each subtransaction is considered as a lower level operation. However, such approaches are based on exploiting the semantics of the application domain, and do not ensure global serializability. Hence, such approaches are not directly related to the scheme developed in this paper.

An interesting observation is that similar to our work (Sect. 6), recently, efforts have also been made to develop concurrency control protocols for multilevel systems in which the abstraction hierarchy may not necessarily be a true hierarchy. Specifically, in Muth et al. (1993), the authors develop a technique for concurrency control in multilevel

systems, in which not every transaction has a representation at each abstraction level. For example, consider a two-level system discussed earlier, in which the levels correspond to the tuples and pages. The authors develop a multilevel concurrency control scheme for a system in which not every transaction has a representation at both the tuple as well as the page abstraction level. Instead some transactions may be implemented as operations on the pages directly. The motivation for their work comes from trying to map the applications in object-oriented databases to multilevel transactions for the purpose of concurrency control. One way to do so is to map the method invocation hierarchy to the abstraction hierarchy of the multilevel transaction. However, it is not too difficult to see that the hierarchy for most systems will not be a true hierarchy as is traditionally assumed in the work on multilevel transactions. Similar to our work on identifying limitations that must be imposed on the domain hierarchy for the developed concurrency control approach (that is, ensuring serializability of $\tilde{S}^D$) to ensure global serializability in hierarchical MDBSs, it will be interesting to study limitations on the abstraction hierarchy that might need to be imposed for the scheme developed in Muth et al. (1993) to ensure serializability in multilevel systems.

8 Conclusions

A multidatabase system (MDBS) is a facility, developed on top of pre-existing local DBMSs, that provides users of a DBMS access and update privileges to data located in other heterogeneous data sources. Over the past decade, substantial research has been done to identify mechanisms for effectively dealing with the problems that arise due to the *heterogeneity* and *autonomy* of the local systems. This research has resulted in transaction management algorithms for MDBSs that ensure correctness without sacrificing the autonomy of the individual system. Most of the proposed approaches have, however, considered an MDBS as a single monolithic system which, executing on top of the existing local DBMSs, controls the execution and commitment of the *global transactions* (transactions that execute at multiple local DBMSs) in such a way that consistency of the individual systems is not jeopardized.

In this paper, we have proposed a hierarchical architecture for multidatabase systems and studied how concurrency control can be done in such systems. We believe that a large MDBS, that spans multiple organizations geographically distributed over nodes of a worldwide computer network will not be developed as a single monolithic system. Instead, it will be developed as a hierarchical system in which an MDBS that integrates certain local DBMSs may itself be a part of a larger MDBS. In a hierarchical MDBS, depending upon the nature of the transactions that execute, the computing resources available, and the reliability of the network, different component MDBSs may follow different transaction management schemes to ensure the consistency of the data they integrate. However, the transaction management algorithms followed by the individual MDBSs must be such that it is feasible to compose them as elements of a larger MDBS.

To describe the architecture, with an MDBS environment we associate a set of *domains* Δ with an ordering relation $\sqsubset$. A domain is either a set of data items at some local DBMS, or it may consist of a union of the set of data items in other domains. The execution of the transactions within a domain $D \in \Delta$ is controlled by the *domain manager* of D. We developed a mechanism using which the domain managers can ensure that the concurrent execution of the transactions does not result in a loss of serializability within their domains. More specifically, for a global schedule S and a domain D, we identified a schedule $\tilde{S}^D$ such that, if $\tilde{S}^D$ is serializable, and for all domains $D' \sqsubset D$ $\tilde{S}^{D'}$ is serializable, then the serializability of the projection of S to data items in D (that is S^D) is ensured. We developed a mechanism using which the domain manager of D can control the order in which the operations that belong to $\tilde{S}^D$ execute such that $\tilde{S}^D$ is serializable. In our mechanism, the domain manager may use any concurrency control protocol known for traditional DBMSs (distributed or centralized) to ensure serializability of $\tilde{S}^D$. Finally, we identified restrictions that need to be imposed on the architecture of the hierarchical MDBSs such that our mechanism of ensuring serializability of $\tilde{S}^D$ for each domain $D \in \Delta$ results in global schedules that are serializable.

In this paper, we did not consider the issue of failure-resilience. Failure-resilience in MDBSs is complicated since the requirement of autonomy preservation renders the usage of *atomic commit protocols* (Bernstein et al. 1987) unsuitable for MDBS environments. In the absence of atomic commit protocols, it is possible that certain subtransactions of a multisite transaction commit, whereas others abort, thereby violating the atomicity property. The problem of ensuring atomicity in MDBS environments has been studied in Breitbart et al. (1990), Wolksi and Veijalainen (1990), Mehrotra et al. (1992b, d) and Zhang et al. (1994). We need to further study how these schemes can be adapted for hierarchical MDBSs. Finally, in this paper we concentrated only on developing mechanisms for ensuring global serializability in hierarchical MDBSs. Since ensuring global serializability in an MDBS environment is both complex and expensive, and schemes that ensure serializability may not offer the desired degree of concurrency, substantial research has been done to develop correctness criteria for MDBSs that are weaker than serializability but ensure database consistency under appropriate assumptions about the MDBS environment (Du and Elmagarmid 1989; Mehrotra et al. 1991). It will be interesting to study concurrency control schemes and the consistency guarantee that results in hierarchical MDBSs in which different domains may follow different notions of correctness.

Appendix Proofs of the theorems

In this appendix, we prove Theorems 1–3 stated in the paper. We begin by first proving Theorem 1. To prove the theorem, we first need to develop the following two lemmas.

Lemma 1. Consider an MDBS environment with the set Δ of domains. Let S be a global schedule and D be an arbitrary domain in Δ. Schedule S^D is serializable, if each of the following three conditions hold:

1. For each domain D_k such that $child(D_k, D)$, schedule S^{D_k} is serializable.
2. For each domain D_k, such that $child(D_k, D)$, there exists a serialization function sf such that the following holds:

$$\text{For all transactions } T_i, T_j, \text{ if } global(T_i, D_k),$$
$$global(T_j, D_k), \text{ and } T_i \overset{*}{\leadsto}_{S^{D_k}} T_j, \text{ then}$$
$$sf(T_i, D_k) \prec_S sf(T_j, D_k).$$

3. Schedule $\tilde{S}^D$ is serializable.

Proof. Assume that S^D is not serializable. Since by (1) each S^{D_k} is serializable, there exist transactions $T_1, T_2, \ldots, T_n$ such that $T_1 \overset{*}{\leadsto}_{S^{D_{k_1}}} T_2, T_2 \overset{*}{\leadsto}_{S^{D_{k_2}}} T_3, \ldots, T_{n-1} \overset{*}{\leadsto}_{S^{D_{k_{n-1}}}} T_n, T_n \overset{*}{\leadsto}_{S^{D_{k_n}}} T_1$, where $child(D_{k_i}, D)$, $global(T_i, D_{k_i})$, and $global(T_{(i+1) \bmod n}, D_{k_i})$, $i = 1, 2, \ldots, n$. By (2), $sf(T_1, D_{k_1}) \prec_S sf(T_2, D_{k_1})$, $sf(T_2, D_{k_2}) \prec_S sf(T_3, D_{k_2})$, $\ldots$, $sf(T_{n-1}, D_{k_{n-1}}) \prec_S sf(T_n, D_{k_{n-1}})$, $sf(T_n, D_{k_n}) \prec_S sf(T_1, D_{k_n})$. Thus, by the definition of conflicts in $\tilde{S}^D$, $\tilde{T}_1^D \leadsto_{\tilde{S}^D} \tilde{T}_2^D, \tilde{T}_2^D \leadsto_{\tilde{S}^D} \tilde{T}_3^D, \ldots, \tilde{T}_{n-1}^D \leadsto_{\tilde{S}^D} \tilde{T}_n^D, \tilde{T}_n^D \leadsto_{\tilde{S}^D} \tilde{T}_1^D$. Hence, $\tilde{T}_1^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_1^D$ which is a contradiction since $\tilde{S}^D$ by (3) above is serializable. Hence proved.

We next prove that the function sf defined in the paper meets the requirement of a serialization function for a domain D.

Lemma 2. Consider an MDBS environment with the set Δ of domains. Let S be a global schedule with transactions T_i, and T_j, and let D be an arbitrary domain in Δ. If $global(T_i, D)$, $global(T_j, D)$ and $T_i \overset{*}{\leadsto}_{S^D} T_j$, then $sf(T_i, D) \prec_S sf(T_j, D)$.

In the proof of Lemma 2, we will need the following notion of a level of a domain:

$$level(D) =$$
$$\begin{cases} 1, & \text{if } D = DB_k \text{ for some local database DBMS}_k \\ maximum(level(D_k)) + 1, & \text{where } child(D_k, D) \end{cases}$$

Proof. The proof is by the induction over the level of the domains.
Basis ($level(D) = 1$)**:** If $level(D) = 1$, then for some DB_k, $D = DB_k$. Hence, for all transactions T_i, T_j, if $global(T_i, D)$, $global(T_j, D)$, and $T_i \overset{*}{\leadsto}_{S_k} T_j$, then by definition of ser_{S_k}, $ser_{S_k}(T_i) \prec_S ser_{S_k}(T_j)$. Hence, $sf(T_i, D)$

$\prec_S sf(T_j, D)$.
Induction. Assume that the lemma is true for all domains D such that $level(D) \le p$. Let

$$D = \bigcup \{D_1, D_2, \ldots, D_n\}$$

be an arbitrary domain such that $level(D) = p+1$. Let T_i, T_j be transactions such that $global(T_i, D)$, $global(T_j, D)$, and $T_i \overset{*}{\leadsto}_{S^D} T_j$. There are two cases to consider:

- ($T_i \overset{*}{\leadsto}_{S^{D_k}} T_j$ for some D_k such that $child(D_k, D)$): Since $global(T_i, D)$ and $global(T_j, D)$ and T_i, T_j executes in D_k, $global(T_i, D_k)$ and $global(T_j, D_k)$. Thus, by IH, $sf(T_i, D_k) \prec_S sf(T_j, D_k)$. Hence, by definition of a conflict in $\tilde{S}^D$, $\tilde{T}_i^D \leadsto_{\tilde{S}^D} \tilde{T}_j^D$. As a result, by the definition of $sf(T, D)$, $sf(T_i, D) \prec_S sf(T_j, D)$
- (There exist transactions $T_1, T_2, \ldots, T_n$ such that $T_i \overset{*}{\leadsto}_{S^{D_{k_1}}} T_1, T_1 \overset{*}{\leadsto}_{S^{D_{k_2}}} T_2, \ldots, T_{n-1} \overset{*}{\leadsto}_{S^{D_{k_{n-1}}}} T_n, T_n \overset{*}{\leadsto}_{S^{D_{k_n}}} T_j$, where $child(D_{k_l}, D)$, $l = 1, 2, \ldots, n$): Note that $global(T_l, D_{k_l})$, $l = 1, 2, \ldots, n$, and $global(T_{l+1}, D_{k_l})$, $l = 1, 2, \ldots, n - 1$. Thus, by IH,

$$sf(T_i, D_{k_1}) \prec_S sf(T_1, D_{k_1}),$$
$$sf(T_1, D_{k_2}) \prec_S sf(T_2, D_{k_2}), \ldots,$$
$$sf(T_{n-1}, D_{k_{n-1}}) \prec_S sf(T_n, D_{k_{n-1}}), sf(T_n, D_{k_n}) \prec_S$$
$$sf(T_j, D_{k_n}).$$

Hence, by definition of a conflict in $\tilde{S}^D$, $\tilde{T}_i^D \leadsto_{\tilde{S}^D} \tilde{T}_1^D$, $\tilde{T}_1^D \leadsto_{\tilde{S}^D} \tilde{T}_2^D, \ldots, \tilde{T}_{n-1}^D \leadsto_{\tilde{S}^D} \tilde{T}_n^D, \tilde{T}_n^D \leadsto_{\tilde{S}^D} \tilde{T}_j^D$. Hence, $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$. As a result, by the definition of $sf(T, D)$, $sf(\tilde{T}_i, D) \prec_S sf(T_j, D)$. Hence proved.

Proof of Theorem 1. The proof is by the induction over the level of the domain D.
Basis ($level(D) = 1$)**:** If $level(D) = 1$, then, for some DB_k, $D = DB_k$. Since S_k is serializable, for all $k = 1, 2, \ldots, n$, S^D is serializable.
Induction. Assume that the theorem is true for each D such that $level(D) \le p$. We show it to be true for each domain, D such that $level(D) = p + 1$. Let D be such a domain and further let $D = \bigcup \{D_1, D_2, \ldots, D_n\}$. Since $level(D_k) \le p$, $child(D_k, D)$, by IH, S^{D_k} is serializable. Further, since $child(D_k, D)$, $D_k \notin TOP$. Thus, the function $ser_{\tilde{S}^{D_k}}$ exists. By Lemma 2, $sf(T_i, D_k) = ser_{\tilde{S}^{D_k}}(T_i)$ satisfies the property that for all T_i, T_j, such that $global(T_i, D_k)$, $global(T_j, D_k)$, $T_i \overset{*}{\leadsto}_{S^{D_k}} T_j \Rightarrow$
$sf(T_i, D_k) \prec_S sf(T_j, D_k)$. Thus, by Lemma 1, since $\tilde{S}^D$ is serializable, S^D is serializable. Hence proved.

Proof of Theorems 2 and 3. Note that Theorem 2 directly follows from Theorem 3, since for a given set of domains Δ and a domain ordering $\sqsubset$ if DG is acyclic, then the corresponding LDG does not contain any undesirable cycles, We, thus, restrict ourselves to proving Theorem 3 which is done in the remainder of the appendix. To do so, let us consider a schedule S that is not serializable. Thus, there exist transactions $T_1, T_2, \ldots, T_n$ such that $T_1 \leadsto_{DB_1} T_2, T_2 \leadsto_{DB_2} T_3, \ldots, T_{n-1} \leadsto_{DB_{n-1}} T_n, T_n \leadsto_{DB_n} T_1$.[5] Let $D \in TOP$ such that $DB_1 \sqsubset D$. If, for all DB_i, $i = 1, 2, \ldots, n$, $DB_i \sqsubset D$,

---

[5] For notational brevity, we denote $\leadsto_{S^{DB_i}}$ (or $\leadsto_{S_i}$) by $\leadsto_{DB_i}$.

then $T_1 \overset{*}{\leadsto}_{S^D} T_1$ which is a contradiction, since, as shown in Theorem 1, for each domain $D \in \Delta$, our scheme ensures that S^D is serializable. Thus, to prove that under the hypothesis of Theorem 3, the resulting global schedule S is serializable, we can restrict ourselves to the case in which there exists a $DB_i \not\sqsubseteq D$. Let DB_k be the first such DB_i, that is, for all DB_i, $1 \le i < k$, $DB_i \sqsubseteq D$ and further $DB_k \not\sqsubseteq D$. In this case, since transaction T_k executes on both DB_{k-1} and DB_k, $global(T_k, D)$ and further, $T_k \overset{*}{\leadsto}_S T_k$. We will show that in this case it must be the case that $\tilde{T}_k^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_k^D$ which is a contradiction since $\tilde{S}^D$ is serializable.

Thus, our task of proving Theorem 3 reduces to that of establishing that under the hypothesis of Theorem 3, if there exists transaction T_i, T_j, such that $global(T_i, D)$ and $global(T_j, D)$, and $T_i \overset{*}{\leadsto}_S T_j$, then $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$. The proof of this is a little involved. Let us first consider the case in which the conflict $T_i \overset{*}{\leadsto} T_j$ arises due to a direct conflict between T_i and T_j at some DBMS$_k$; that is, $T_i \leadsto_{DB_k} T_j$. In this case, there are two possibilities– either $DB_k \sqsubseteq D$, or $DB_k \not\sqsubseteq D$. If $DB_k \sqsubseteq D$, then $T_i \overset{*}{\leadsto}_{S^D} T_j$. The following lemma shows that if $T_i \overset{*}{\leadsto}_{S^D} T_j$, then $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$.

Lemma 3. Let T_i and T_j be transactions and D be a domain such that $global(T_i, D)$ and $global(T_j, D)$ and $level(D) \ge 2$. If $T_i \overset{*}{\leadsto}_{S^D} T_j$, then $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$.

Proof. Let $p = level(D)$ The proof is by induction on p.
Basis ($p = 2$)**:** Thus, $D = \{DB_1, DB_2, \ldots, DB_m\}$ for some local database DBMS$_k$, $k = 1, 2, \ldots, m$. Since $T_i \overset{*}{\leadsto}_{S^D} T_j$, there exists transactions $T_1, T_2, \ldots, T_n$ such that $T_i \overset{*}{\leadsto}_{DB_{k_1}} T_1$, $T_1 \overset{*}{\leadsto}_{DB_{k_2}} T_2$, $\ldots$, $T_{n-1} \overset{*}{\leadsto}_{DB_{k_n}} T_n$, $T_n \overset{*}{\leadsto}_{DB_{k_{n+1}}} T_j$, where

$global(T_l, DB_{k_l})$ and $global(T_l, DB_{k_{(l+1)}})$, $l = 1, 2, \ldots, n$.

Hence, by definition of the serialization function sf,

$$sf(T_i, DB_{k_1}) \prec_S sf(T_1, DB_{k_1}),$$
$$sf(T_1, DB_{k_2}) \prec_S sf(T_2, DB_{k_2}), \ldots,$$
$$sf(T_{n-1}, DB_{k_n}) \prec_S sf(T_n, DB_{k_n}), \text{ and}$$
$$sf(T_n, DB_{k_{n+1}}) \prec_S sf(T_j, DB_{k_{n+1}}).$$

Thus, by definition of $\tilde{T}_i$, $\tilde{T}_i^D \leadsto_{\tilde{S}^D} \tilde{T}_1^D$, $\tilde{T}_1^D \leadsto_{\tilde{S}^D} \tilde{T}_2^D$, $\ldots$, $\tilde{T}_{n-1}^D \leadsto_{\tilde{S}^D} \tilde{T}_n^D$, and $\tilde{T}_n^D \leadsto_{\tilde{S}^D} \tilde{T}_j^D$. Hence, $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$.
Induction. Assume that the lemma holds for all domains such that $level(D) \le p$. We show that it holds for domains such that $level(D) = p + 1$. Let $D = \bigcup\{D_1, D_2, \ldots, D_m\}$ be an arbitrary domain such that $level(D) = p + 1$. Since $T_i \overset{*}{\leadsto}_{S^D} T_j$, there exist transactions $T_1, T_2, \ldots, T_n$, $n \ge 0$, such that $T_i \overset{*}{\leadsto}_{S^{D_{k_1}}} T_1$, $T_1 \overset{*}{\leadsto}_{S^{D_{k_2}}} T_2$, $\ldots, T_{n-1} \overset{*}{\leadsto}_{S^{D_{k_n}}} T_n$, and $T_n \overset{*}{\leadsto}_{S^{D_{k_{n+1}}}} T_j$. Since $child(D_{k_l}, D)$, $level(D_{k_l}) < level(D)$. Thus, by IH, $\tilde{T}_i^{D_{k_1}} \overset{*}{\leadsto}_{\tilde{S}^{D_{k_1}}} \tilde{T}_1^{D_{k_1}}$, $\tilde{T}_1^{D_{k_2}} \overset{*}{\leadsto}_{\tilde{S}^{D_{k_2}}} \tilde{T}_2^{D_{k_2}}$, $\ldots$, $\tilde{T}_{n-1}^{D_{k_n}} \overset{*}{\leadsto}_{\tilde{S}^{D_{k_n}}} \tilde{T}_n^{D_{k_n}}$, and $\tilde{T}_n^{D_{k_{n+1}}} \overset{*}{\leadsto}_{\tilde{S}^{D_{k_{n+1}}}} \tilde{T}_j^{D_{k_{n+1}}}$. Hence, by definition of the serialization function sf, $sf(T_i, D_{k_1}) \prec_S sf(T_1, D_{k_1})$, $sf(T_1, D_{k_2}) \prec_S sf(T_2, D_{k_2})$, $\ldots, sf(T_{n-1}, D_{k_n}) \prec_S sf(T_n, D_{k_n})$, and $sf(T_n, D_{k_{n+1}}) \prec_S sf(T_j, D_{k_{n+1}})$. Thus, by definition of $\tilde{T}_i$, $\tilde{T}_i^D \leadsto_{\tilde{S}^D} \tilde{T}_1^D$, $\tilde{T}_1^D \leadsto_{\tilde{S}^D} \tilde{T}_2^D$, $\ldots, \tilde{T}_{n-1}^D \leadsto_{\tilde{S}^D} \tilde{T}_n^D$, $\tilde{T}_n^D \leadsto_{\tilde{S}^D} \tilde{T}_j^D$. Hence,

$\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$. Hence proved.

Recall that we were considering the proof of the fact that if there exist transactions T_i, T_j, such that, $global(T_i, D)$ and $global(T_j, D)$, and $T_i \leadsto_{DB_k} T_j$, then $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$. If in case, $DB_k \sqsubseteq D$, then the above developed lemma shows that $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$. We next consider the case in which $DB_k \not\sqsubseteq D$. Let $DB_k \sqsubseteq D'$, where $D' \in TOP$. Note that, since $global(T_i, D)$ and $global(T_j, D)$, it must be the case that $global(T_i, D')$ and $global(T_j, D')$. Thus, by Lemma 3, we have that $\tilde{T}_i^{D'} \overset{*}{\leadsto}_{\tilde{S}^{D'}} \tilde{T}_j^{D'}$. To prove that $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$, we will show that if for any pair of transactions T_i, T_j and domains $D, D' \in TOP$, where $global(T_i, D)$ and $global(T_j, D)$, if $\tilde{T}_i^{D'} \overset{*}{\leadsto}_{\tilde{S}^{D'}} \tilde{T}_j^{D'}$, then $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$.

Lemma 4. Let the set Δ satisfy restriction **R1** and the LDG be acyclic. Further, let T_i, T_j be transactions and $D, D' \in TOP$ be domains such that $global(T_i, D')$ and $global(T_j, D')$. If $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$, then $\tilde{T}_i^{D'} \overset{*}{\leadsto}_{\tilde{S}^{D'}} \tilde{T}_j^{D'}$.

To prove Lemma 4, we first develop the following two lemmas that relate conflicts between transactions in domains D, D', where $D' \sqsubset D$.

Lemma 5. Let D be a domain and T_i, T_j be transactions such that $global(T_i, D)$ and $global(T_j, D)$. If there exists a $D' \sqsubseteq D$ such that $\tilde{T}_i^{D'} \overset{*}{\leadsto}_{\tilde{S}^{D'}} \tilde{T}_j^{D'}$, then $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$.

Proof. The proof in by induction on the level of the domain D, where $D' \sqsubseteq D$.
Basis ($level(D) = level(D')$)**:** Since $D' \sqsubseteq D$, it must be the case that $D' = D$. Thus, $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$.
Induction. Assume that the lemma is true for all domains $D, D' \sqsubseteq D$ such that $level(D) \le level(D') + p$. We show that the lemma is true for all domains such that $level(D) = level(D') + p + 1$. Let D be such a domain. Since $D' \sqsubseteq D$, there exists a domain D'', $D' \sqsubseteq D''$, where $child(D'', D)$. Further, since $global(T_i, D)$ and $global(T_j, D)$, and since T_i and T_j execute in D'', it must be the case that $global(T_i, D'')$ and $global(T_j, D'')$. Thus, by IH, $\tilde{T}_i^{D''} \overset{*}{\leadsto}_{\tilde{S}^{D''}} \tilde{T}_j^{D''}$. Since $\tilde{T}_i^{D''} \overset{*}{\leadsto}_{\tilde{S}^{D''}} \tilde{T}_j^{D''}$, by definition of sf, $sf(T_i, D'') \prec_S sf(T_j, D'')$. Thus, by definition of $\tilde{T}_i$, $\tilde{T}_i^D \leadsto_{\tilde{S}^D} \tilde{T}_j^D$. Hence, $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$.

Lemma 6. Let T_i, T_j be transactions and let D be a domain such that $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$. For all D', $D' \sqsubset D$, if T_i and T_j execute in D', then $sf(T_i, D') \prec_S sf(T_j, D')$.

Proof. Let there exists a D' such that $sf(T_j, D') \prec_S sf(T_i, D')$. Thus, there exists a domain D'' such that $D'' \sqsubseteq D$, $parent(D'', D')$ such that $\tilde{T}_j^{D''} \leadsto_{\tilde{S}^{D''}} \tilde{T}_i^{D''}$. Hence, by Lemma 5, $\tilde{T}_j^D \leadsto_{\tilde{S}^D} \tilde{T}_i^D$. Thus, $\tilde{S}^D$ is not serializable, which is a contradiction. Hence, such a D' does not exist. Thus, for all D', $D' \sqsubset D$, $sf(T_i, D) \prec_S sf(T_j, D)$.

Proof of Lemma 4. There are two cases to consider.

– $(D \cap D' \neq \emptyset:)$ We first show that both T_i and T_j execute at $D \cap D'$. If T_i does not execute at $D \cap D'$, then, since T_i executes at D, there exists a $DB_1 \sqsubset D$ and a $DB_2 \sqsubset D'$ such that T_i executes at DB_1 and DB_2, where $DB_1 \not\sqsubset D \cap D'$ and $DB_2 \not\sqsubset D \cap D'$. Since T_i executes at DB_1 and DB_2, there exists a domain $D'' \in TOP$, $Dom(T_i) \sqsubset D''$, such that $D'' \neq D$ and $D'' \neq D'$. Consider the labeled domain graph LDG. In LDG, since $DB_1 \sqsubset D$ and $DB_1 \sqsubset D''$, there is an edge (D, D'') such that $DB_1 \sqsubset label(D, D'')$. Further, since $DB_2 \sqsubset D'$ and $DB_2 \sqsubset D''$, there is an edge (D', D'') such that $DB_2 \sqsubset label(D', D'')$. Since $D \cap D' \neq \emptyset$, there exists an edge (D, D') in LDG. Thus, LDG contains a cycle (D, D''), (D'', D'), (D', D). Since $DB_1 \not\sqsubset D'$, $DB_1 \not\sqsubset label(D, D')$. Further, since $DB_2 \not\sqsubset D$, $DB_2 \not\sqsubset label(D, D')$. Hence, the cycle (D, D''), (D'', D'), (D', D) is an undesirable cycle. Thus, it must be the case that T_i executes in $D \cap D'$. Similarly, it is the case that T_j executes in $D \cap D'$. Since $\tilde{T}_i^D \stackrel{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$, by Lemma 6, we have that $sf(T_i, D \cap D') \prec_S sf(T_j, D \cap D')$. Thus, by Lemma 5, since $global(T_i, D')$ and $global(T_j, D')$, we have that $\tilde{T}_i^{D'} \stackrel{*}{\leadsto}_{\tilde{S}^{D'}} \tilde{T}_j^{D'}$.

– $(D \cap D' = \emptyset:)$ Since T_i executes at D as well as D', let T_i executes at DB_1, DB_3, where $DB_1 \sqsubset D$ and $DB_3 \sqsubset D'$. Further, since T_j executes at D as well as D', let T_j executes at DB_2, DB_4, where $DB_2 \sqsubset D$ and $DB_4 \sqsubset D'$. We show that there exists a domain D'' such that $DB_1 \sqsubset D''$, $DB_2 \sqsubset D''$, $DB_3 \sqsubset D''$, $DB_4 \sqsubset D''$. Say, such a domain D'' does not exist. Since T_i executes at DB_1 and DB_3, there exists a domain $D''' \in TOP$, such that $Dom(T_i) \sqsubset D'''$ and thus $DB_1 \sqsubset D'''$ and $DB_3 \sqsubset D'''$. Further, since T_j executes at DB_2 and DB_4, there exists a domain $D'''' \in TOP$, such that $Dom(T_j) \sqsubset D''''$ and thus $DB_2 \sqsubset D''''$ and $DB_4 \sqsubset D''''$. If $D''' = D''''$, then $DB_1 \sqsubset D'''$, $DB_2 \sqsubset D'''$, $DB_3 \sqsubset D'''$, and $DB_4 \sqsubset D'''$. Hence, $D''' \neq D''''$. Thus, $D \neq D' \neq D''' \neq D''''$. Consider the labeled domain graph LDG. In LDG, there is an edge (D, D''') such that $DB_1 \sqsubset label(D, D''')$, there is an edge (D''', D') such that $DB_3 \sqsubset label(D, D''')$, there is an edge (D', D'''') such that $DB_4 \sqsubset label(D', D'''')$, and there is an edge (D'''', D) such that $DB_2 \sqsubset label(D'''', D)$. Thus, LDG contains a cycle (D, D'''), (D''', D'), (D', D''''), (D'''', D). We next show that LDG contains a undesirable cycle. There are two cases to consider:

– $(D''' \cap D'''' = \emptyset :)$ Since $DB_1 \sqsubset label(D, D''')$, $DB_1 \sqsubset D'''$. Since $D''' \cap D'''' = \emptyset$, $DB_1 \not\sqsubset D''''$. Thus, $DB_1 \not\sqsubset label(D'''', D)$ and further $DB_1 \not\sqsubset label(D'''', D')$. Similarly, since $DB_1 \not\sqsubset D'$, $DB_1 \not\sqsubset label(D', D''')$. Hence, $label(D, D''') \neq label(D', D''')$, $label(D, D''') \neq label(D', D'''')$, and $label(D, D''') \neq label(D, D'''')$. Using similar reasoning, we can show that $label(D, D''') \neq label(D''', D') \neq label(D', D'''') \neq label(D'''', D)$. Hence, the cycle (D, D'''), (D''', D'), (D', D''''), (D'''', D) is an undesirable cycle.

– $(D''' \cap D'''' \neq \emptyset :)$ If $D''' \cap D'''' \neq \emptyset$, then LDG contains an edge (D'''', D''') and thus LDG, besides containing the cycle (D, D'''), (D''', D'), (D', D''''), (D'''', D), also contains cycles (D, D'''), (D''', D''''), (D'''', D) and (D', D'''), (D''', D''''), (D'''', D'). Note that, since $D \cap D' = \emptyset$, it must be the case that $DB_1 \not\sqsubset label(D', D''')$ and $DB_1 \not\sqsubset label(D', D'''')$. Further, $DB_2 \not\sqsubset label(D', D''')$ and $DB_2 \not\sqsubset label(D', D'''')$. Similarly, $DB_3 \not\sqsubset label(D', D''')$, $DB_3 \not\sqsubset label(D', D'''')$, $DB_4 \not\sqsubset label(D, D''')$ and $DB_4 \not\sqsubset label(D, D'''')$. Thus, if the cycle (D, D'''), (D''', D'), (D', D''''), (D'''', D) is not an undesirable cycle, then either $label(D, D'''') = label(D, D''')$ or $label(D', D'''') = label(D', D''')$. Note that $label(D, D'''') = label(D, D''')$ and $label(D', D'''') = label(D', D''')$ both cannot hold, since then D_3 would be such that $DB_1 \sqsubset D_3$, $DB_2 \sqsubset D_3$, $DB_3 \sqsubset D_3$, and $DB_4 \sqsubset D_3$. If $label(D, D'''') = label(D, D''')$, and $label(D', D'''') \neq label(D', D''')$, then the cycle (D', D'''), (D''', D''''), (D'''', D') is an undesirable cycle. Else, if
$$label(D', D'''') = label(D', D'''),$$
and
$$label(D, D'''') \neq label(D, D'''),$$
then the cycle (D, D'''), (D''', D''''), (D'''', D) is an undesirable cycle.

Hence, there must exist a domain D'' such that $DB_1 \sqsubset D''$, $DB_2 \sqsubset D''$, $DB_3 \sqsubset D''$, and $DB_4 \sqsubset D''$. Since $global(T_i, D \cap D'')$ and $global(T_j, D \cap D'')$, and $\tilde{T}_i^D \stackrel{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$, by Lemma 6, $sf(T_i, D \cap D'') \prec_S sf(T_j, D \cap D'')$. Hence, by Lemma 5, and the definition of $\tilde{T}_i$, $\tilde{T}_i^{D''} \stackrel{*}{\leadsto}_{\tilde{S}^{D''}} \tilde{T}_j^{D''}$. Since $global(T_i, D' \cap D'')$ and $global(T_j, D' \cap D'')$, and $\tilde{T}_i^{D''} \stackrel{*}{\leadsto}_{\tilde{S}^{D''}} \tilde{T}_j^{D''}$, by Lemma 6, $sf(T_i, D' \cap D'') \prec_S sf(T_j, D' \cap D'')$. Hence, by Lemma 5, and the definition of $\tilde{T}_i$, $\tilde{T}_i^{D'} \stackrel{*}{\leadsto}_{\tilde{S}^{D'}} \tilde{T}_j^{D'}$. Hence proved.

Using Lemmas 3 and 4 we can establish that if there exists transaction T_i, T_j, such that $global(T_i, D)$ and $global(T_j, D)$, and $T_i \leadsto_{DB_k} T_j$, then $\tilde{T}_i^D \stackrel{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$. Recall that to prove Theorem 3 we needed to show that under the hypothesis of Theorem 3, if there exists transaction T_i, T_j, such that $global(T_i, D)$ and $global(T_j, D)$, and $T_i \stackrel{*}{\leadsto}_S T_j$, then $\tilde{T}_i^D \stackrel{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_j^D$. We have already proved the above under the assumption that the conflict $T_i \stackrel{*}{\leadsto} T_j$ arises as a result of a direct conflict between transactions T_i and T_j; that is, there exists a DB_k such that $T_i \leadsto_{DB_k} T_j$. We next relax the assumption and show that, for any arbitrary conflict $T_i \stackrel{*}{\leadsto}_S T_j$, the claim holds. To do so, we will require the following lemma that establishes how the presence/absence of certain edges between the nodes in an LDG impacts the nature of the conflicts between transactions that can occur in the system.

Lemma 7. Let $T_1, T_2, \ldots, T_n$, $n > 2$, be transactions such that $T_1 \leadsto_{DB_1} T_2$, $T_2 \leadsto_{DB_2} T_3$, $\ldots$, $T_{n-1} \leadsto_{DB_{n-1}} T_n$. Let D' and D'', $D' \in TOP$, $D'' \in TOP$, be domains

such that $DB_1 \sqsubseteq D'$ and $DB_{n-1} \sqsubseteq D''$. There exists a path (D', D_1), (D_1, D_2), ..., (D_{r-1}, D_r), (D_r, D'') in LDG such that for all D_i, $i = 1, 2, \ldots, r$, there exists a DB_j, $j = 1, 2, \ldots, n - 1$, such that $DB_j \sqsubset D_i$. Further, let edges (D', D_1), (D_1, D_2), ..., (D_{r-1}, D_r), (D_r, D'') have labels $L_1, L_2, \ldots, L_r$, respectively. For all L_i, $i = 1, 2, \ldots, r$, there exists a DB_j, $j = 1, 2, \ldots, n - 1$, such that $DB_j \sqsubset L_i$.

Proof. The proof is by induction on n.

Basis ($n = 3$): Thus, $T_1 \leadsto_{DB_1} T_2$ and $T_2 \leadsto_{DB_2} T_3$, where $DB_1 \sqsubset D'$ and $DB_2 \sqsubset D''$. Consider the domain $D \in TOP$ such that $Dom(T_2) \sqsubseteq D$. If $D = D'$, then, since $DB_2 \sqsubset D''$ and $DB_1 \sqsubset D'$, there is an edge (D', D'') in LDG. Further, $DB_2 \sqsubset label(D', D'')$. Else, if $D = D''$, then since $DB_1 \sqsubset D'$ and $DB_2 \sqsubset D''$, there is an edge (D', D'') in LDG. Further, $DB_1 \sqsubset label(D', D'')$. Else, if $D \neq D'$ and $D \neq D''$, then there are edges (D', D) and (D, D'') in LDG such that $DB_1 \sqsubset label(D', D)$ and $DB_2 \sqsubset label(D, D'')$.

Induction. Assume that the lemma holds for $n = m - 1$, $m \geq 4$. We show it holds for $n = m$. Thus, we have $T_1 \leadsto_{DB_1} T_2$, $T_2 \leadsto_{DB_2} T_3$, ..., $T_{m-1} \leadsto_{DB_{m-1}} T_m$. Let $DB_{m-2} \sqsubset D'''$, where $D''' \in TOP$. If $D''' = D'$, then by base case, the lemma holds. Else, if $D''' = D''$, then $T_1 \leadsto_{DB_1} T_2$, $T_2 \leadsto_{DB_2} T_3$, ..., $T_{m-2} \leadsto_{DB_{m-2}} T_{m-1}$, where $DB_1 \sqsubseteq D'$ and $DB_{m-1} \sqsubseteq D''$. Hence, by IH, the lemma holds. Else, $D''' \neq D'$ and $D''' \neq D''$. By IH, there exists a path (D', D_1), (D_1, D_2), ..., (D_{r-1}, D_r), (D_r, D''') in LDG such that for all D_i, $i = 1, 2, \ldots, r$, there exists a DB_j, $j = 1, 2, \ldots, m - 2$, such that $DB_j \sqsubset D_i$. Further, for all L_i, $i = 1, 2, \ldots, r$, there exists a DB_j, $j = 1, 2, \ldots, m - 2$, such that $DB_j \sqsubset L_i$. By the base case, since $T_{m-2} \leadsto_{DB_{m-2}} T_{m-1}$ and $T_{m-1} \leadsto_{DB_{m-1}} T_m$, there exists a path (D''', D'_1), (D'_1, D'_2), ..., $(D'_{r'-1}, D'_{r'})$, $(D'_{r'}, D'')$ such that for all D'_i, $i = 1, 2, \ldots, r'$, there exists a DB_j, $j = m - 2, m - 1$, such that $DB_j \sqsubset D'_i$. Further, for all L_i, $i = 1, 2, \ldots, r'$, there exists a DB_j, $j = m - 2, m - 1$, such that $DB_j \sqsubset L_i$. Hence, for some s there exists a path (D', D_1), (D_1, D_2), ..., (D_{s-1}, D_s), (D_s, D'') in LDG such that for all D_i, $i = 1, 2, \ldots, s$, there exists a DB_j, $j = 1, 2, \ldots, m - 1$, such that $DB_j \sqsubset D_i$. Further, for all L_i, $i = 1, 2, \ldots, s$, there exists a DB_j, $j = 1, 2, \ldots, m - 1$, such that $DB_j \sqsubset L_i$.

Lemma 8. Let the set Δ of domains satisfy the restriction **R1** and let LDG be acyclic. Further, let $D \in TOP$ be a domain, $level(D) \geq 2$, and T_1 and T_n be transactions such that $global(T_1, D)$ and $global(T_n, D)$. If $T_1 \leadsto_{DB_1} T_2$ $T_2 \leadsto_{DB_2} T_3$, ..., $T_{n-1} \leadsto_{DB_{n-1}} T_n$, then $\tilde{T}_1^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_n^D$.

Proof. The proof in by induction on n.

Basis ($n = 1$): Thus, $T_1 \leadsto_{DB_1} T_2$. There are two cases to consider.

- ($DB_1 \sqsubseteq D$:) In this case, $T_1 \leadsto_{S^D} T_2$. Thus, by Lemma 3, since $global(T_1, D)$ and $global(T_2, D)$, we have that $\tilde{T}_1^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_2^D$.
- ($DB_1 \not\sqsubseteq D$:) Let $DB_1 \sqsubset D'$, where $D' \in TOP$. By Lemma 3, $\tilde{T}_1^{D'} \overset{*}{\leadsto}_{\tilde{S}^{D'}} \tilde{T}_2^{D'}$. Since $global(T_1, D)$ and $global(T_2, D)$, by Lemma 4, $\tilde{T}_1^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_n^D$.

Induction. Assume that the lemma holds for all $n \leq m - 1$. We show that it holds for $n = m$. Thus, we have $T_1 \leadsto_{DB_1}$

T_2, $T_2 \leadsto_{DB_2} T_3$, ..., $T_{m-1} \leadsto_{DB_{m-1}} T_m$. There are two cases to consider.

- (there exists i, $i = 1, 2, 3, \ldots, m - 1$, $DB_i \sqsubset D$:) Let $DB_k \sqsubset D$, $1 \leq k \leq m - 1$. Further, let DB_{k_1} and DB_{k_2}, $1 \leq k_1 \leq k$, and $k \leq k_2 \leq m - 1$, be such that for all DB_i, $i = k_1, k_1 + 1, \ldots, k, k + 1, \ldots, k_2$, $DB_i \sqsubset D$, $DB_{k_1 - 1} \not\sqsubset D$ and $DB_{k_2 + 1} \not\sqsubset D$. If $k_1 = 1$ and $k_2 = m - 1$, then by Lemma 3, since for all DB_i, $i = 1, 2, \ldots, m - 1$, $DB_i \sqsubset D$, $\tilde{T}_1^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_m^D$. So we only need to consider the case in which either $1 < k_1$ or $k_2 < m - 1$. There are two cases to consider:
  - ($1 < k_1$:) Consider transaction T_{k_1}. Note that $T_{k_1 - 1} \leadsto_{DB_{k_1 - 1}} T_{k_1}$ and further $T_{k_1} \leadsto_{DB_{k_1}} T_{k_1 + 1}$. Since $DB_{k_1 - 1} \not\sqsubseteq D$, $DB_{k_1} \sqsubseteq D$, and transaction T_{k_1} executes on DB_{k_1} and $DB_{k_1 - 1}$, $global(T_{k_1}, D)$. Hence, by IH, $\tilde{T}_1^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_{k_1}^D$ and further $\tilde{T}_{k_1}^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_n^D$. Hence, $\tilde{T}_1^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_n^D$.
  - ($k_2 < m - 1$:) Consider transaction T_{k_2}. Note that $T_{k_2} \leadsto_{DB_{k_2}} T_{k_2 + 1}$ and further $T_{k_2 + 1} \leadsto_{DB_{k_2 + 1}} T_{k_2 + 2}$. Since $DB_{k_2 + 1} \not\sqsubseteq D$, $DB_{k_2} \sqsubseteq D$, and transaction $T_{k_2 + 1}$ executes on DB_{k_2} and $DB_{k_1 + 1}$, $global(T_{k_2}, D)$. Hence, by IH, $\tilde{T}_1^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_{k_2 + 1}^D$ and further $\tilde{T}_{k_2 + 1}^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_n^D$. Hence, $\tilde{T}_1^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_n^D$.
- (for all i, $i = 1, 2, 3, \ldots, m - 1$, $DB_i \not\sqsubset D$:) Let $DB_1 \sqsubset D'$, $D' \neq D$, where $D' \in TOP$ and $Dom(T_1) \sqsubseteq D'$. There are two cases to consider:
  - ($DB_{m-1} \sqsubseteq D'$:) We first show that $\tilde{T}_1^{D'} \overset{*}{\leadsto}_{\tilde{S}^{D'}} \tilde{T}_m^{D'}$. It will follow from Lemma 4 that $\tilde{T}_1^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_m^D$. Since $DB_{m-1} \sqsubseteq D'$, we have that $T_1 \leadsto_{DB_1} T_2$, $T_2 \leadsto_{DB_2} T_3$, ..., $T_{m-1} \leadsto_{DB_{m-1}} T_m$, where $DB_1, DB_{m-1} \sqsubset D'$. If for all DB_i, $i = 1, 2, \ldots, m - 1$, $DB_i \sqsubset D'$, then, since $T_1 \overset{*}{\leadsto}_{S^{D'}} T_m$, by Lemma 3, we have that $\tilde{T}_1^{D'} \overset{*}{\leadsto}_{\tilde{S}^{D'}} \tilde{T}_m^{D'}$. Thus, by Lemma 4, $\tilde{T}_1^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_m^D$. Else, there exists a DB_k, $k = 2, 3, \ldots, m - 2$, such that $DB_k \not\sqsubset D'$. Let k_1 be such that $DB_{k_1} \not\sqsubset D'$ and for all $k = 1, 2, \ldots, k_1 - 1$, $DB_k \sqsubset D'$. Thus, $T_{k_1 - 1} \leadsto_{DB_{k_1 - 1}} T_{k_1}$ and $T_{k_1} \leadsto_{DB_{k_1}} T_{k_1 + 1}$, where $DB_{k_1 - 1} \sqsubset D'$ and $DB_{k_1} \not\sqsubset D'$. Since T_{k_1} executes both on $DB_{k_1 - 1}$ and DB_{k_1}, $global(T_{k_1}, D')$. Hence by IH, $\tilde{T}_1^{D'} \overset{*}{\leadsto}_{\tilde{S}^{D'}} \tilde{T}_{k_1}^{D'}$ and $\tilde{T}_{k_1}^{D'} \overset{*}{\leadsto}_{\tilde{S}^{D'}} \tilde{T}_m^{D'}$. Hence, $\tilde{T}_1^{D'} \overset{*}{\leadsto}_{\tilde{S}^{D'}} \tilde{T}_m^{D'}$. Thus, by Lemma 4, $\tilde{T}_1^D \overset{*}{\leadsto}_{\tilde{S}^D} \tilde{T}_m^D$.
  - ($DB_{m-1} \not\sqsubseteq D'$:) Let $DB_{m-1} \sqsubset D''$, where $D'' \in TOP$ and $Dom(T_m) \sqsubseteq D''$. Note that $D'' \neq D'$ and further $D'' \neq D$. Since T_1 executes in both D and D', and $Dom(T_1) \sqsubseteq D'$, LDG contains an edge (D, D'). Let $label(D, D') = L'$. Similarly, since T_m executes in both D and D'', and $Dom(T_m) \sqsubseteq D''$, LDG contains an edge (D, D''). Let $label(D, D'') = L''$. We first show that it must be the case that $L' = L''$.

    Assume on the contrary that $L' \neq L''$. Since $T_1 \leadsto_{DB_1} T_2$, $T_2 \leadsto_{DB_2} T_3$, ..., $T_{m-1} \leadsto_{DB_{m-1}} T_m$, where $DB_1 \sqsubset D'$ and $DB_{m-1} \sqsubset D''$, by Lemma 7, there exists a path (D', D_1), (D_1, D_2), ..., (D_{r-1}, D_r), (D_r, D'') such that for all D_i, $i = 1, 2, \ldots, r$, there exists a DB_j, $j = 1, 2, \ldots, m - 1$, $DB_j \sqsubset D_i$

and further, for all edges in the path (D_l, D_m), there exists a DB_j, $j = 1, 2, \ldots, m-1$, $DB_j \sqsubseteq label(D_l, D_m)$. Since for all DB_j, $DB_j \not\sqsubseteq D$, the path does not contain D. Hence, LDG contains a cycle (D, D'), (D', D_1), (D_1, D_2), $\ldots$, (D_{r-1}, D_r), (D_r, D''), (D'', D). We next show, using induction on r, that LDG contains an undesirable cycle.

Basis ($r = 0$:) Thus, LDG contains an edge (D', D''). Let

$label(D', D''') = L'''$. Since there exists a DB_j such that $DB_j \sqsubseteq L'''$, and further since $L' \sqsubseteq D$ and $L'' \sqsubseteq D$, it is the case that $L''' \neq L''$ and $L''' \neq L'$. Since by assumption $L' \neq L''$, the cycle, (D, D'), (D', D''), (D'', D) is an undesirable cycle.

Induction. Assume that if there is a cycle (D, D'), (D', D_1), (D_1, D_2), $\ldots$, (D_{r-2}, D_{r-1}), (D_{r-1}, D''), (D'', D), then LDG contains an undesirable cycle. We next show that if there exists a cycle

$$(D, D'), (D', D_1), (D_1, D_2), \ldots, (D_{r-1}, D_r),$$
$$(D_r, D''), (D'', D)$$

in the LDG, then LDG contains an undesirable cycle. Consider the cycle (D, D'), (D', D_1), (D_1, D_2), $\ldots$, (D_{r-1}, D_r), (D_r, D''), (D'', D). Let the labels on the edges (D', D_1), (D_1, D_2), $\ldots$, (D_{r-1}, D_r), (D_r, D'') be $L_1, L_2, \ldots, L_r$, respectively. By assumption $L' \neq L''$. Further, since, for each L_i, there exists a DB_j, $j = 1, 2, \ldots, m-1$ such that $DB_j \sqsubseteq L_i$, and since, $L' \sqsubseteq D$, $L'' \sqsubseteq D$, and since for all DB_j, $j = 1, 2, \ldots, m-1$, $DB_j \not\sqsubseteq D$, it is the case that $L' \neq L_i$ and $L'' \neq L_i$, for all $i = 1, 2, \ldots, r$. Thus, if the cycle (D, D'), (D', D_1), (D_1, D_2), $\ldots$, (D_{r-2}, D_{r-1}), (D_{r-1}, D''), (D'', D), is not a undesirable cycle, then there must exist labels L_{r_1}, L_{r_2} in the cycle such that $L_{r_1} = L_{r_2}$. Consider domains $D_{r_1 - 1}, D_{r_2 + 1}$. Since $L_{r_1} \sqsubseteq D_{r_1 - 1}$ and $L_{r_2} \sqsubseteq D_{r_2 + 1}$, there is an edge between $D_{r_1 - 1}$ and $D_{r_2 + 1}$ with a label L such that $L_{r_1} \sqsubseteq L$. Hence, there exists a cycle in LDG, (D, D'), (D', D_1), (D_1, D_2), $\ldots$, $(D_{r_1 - 1}, D_{r_2 + 1})$, $(D_{r_2 + 1}, D_{r_2 + 2})$, $\ldots$, (D_{r-1}, D_r), (D_r, D''), (D'', D) such that the length of the cycle is less than r. Thus, by IH, there exists an undesirable cycle in LDG.

Hence, it must be the case that $L' = L''$. Since $Dom(T_m) \sqsubseteq D''$, and T_m executes in D, T_m executes in L''. Since $L'' = L'$ and $L' \sqsubseteq D'$, it is the case that T_m executes in D'. Further, since $global(T_m, D)$, it is the case that $global(T_m, D')$. Hence, we have that $T_1 \leadsto_{DB_1} T_2$, $T_2 \leadsto_{DB_2} T_3$, $\ldots$, $T_{m-1} \leadsto_{DB_{m-1}} T_m$, where $global(T_1, D')$, $global(T_m, D')$ and $DB_1 \sqsubseteq D'$ and $DB_{m-1} \sqsubseteq D''$. Since $D' \neq D''$, there exists a k such that for all i, $1 \leq i < k$, $DB_k \sqsubseteq D'$ and $DB_k \not\sqsubseteq D'$. Hence, since $T_{k-1} \leadsto_{DB_{k-1}} T_k$, and $T_k \leadsto_{DB_k} T_{k+1}$, where $DB_{k-1} \sqsubseteq D'$ and $DB_k \not\sqsubseteq D'$, we have that $global(T_k, D')$. Thus, by IH, we have that $\tilde{T}_1^{D'} \overset{*}{\leadsto}_{\tilde{S}D'} \tilde{T}_k^{D'}$ and $\tilde{T}_k^{D'} \overset{*}{\leadsto}_{\tilde{S}D'} \tilde{T}_n^{D'}$. Thus, $\tilde{T}_1^{D'} \overset{*}{\leadsto}_{\tilde{S}D'} \tilde{T}_n^{D'}$. Hence, by Lemma 4, $\tilde{T}_1^D \overset{*}{\leadsto}_{\tilde{S}D} \tilde{T}_n^D$. Hence proved.

Since we had reduced our task of proving Theorem 3 to that of proving that under the hypothesis of Theorem 3, if

there exists transaction T_i, T_j, such that $global(T_i, D)$ and $global(T_j, D)$, and $T_i \overset{*}{\leadsto}_S T_j$, then $\tilde{T}_i^D \overset{*}{\leadsto}_{\tilde{S}D} \tilde{T}_j^D$, the proof of Lemma 8 completes our task of proving Theorem 3. We summarize the proof of Theorem 3 below.

Proof of Theorem 3. If S is not serializable, then there exist transactions $T_1, T_2, \ldots, T_n$ such that $T_1 \leadsto_{DB_1} T_2$, $T_2 \leadsto_{DB_2} T_3$, $\ldots$, $T_{n-1} \leadsto_{DB_{n-1}} T_n$, $T_n \leadsto_{DB_n} T_1$. Let $D \in TOP$ such that $DB_1 \sqsubseteq D$. If for all DB_i, $i = 1, 2, \ldots, n$, $DB_i \sqsubseteq D$, then $T_1 \overset{*}{\leadsto}_{SD} T_1$. Hence, by Lemma 3, $\tilde{T}_1^D \overset{*}{\leadsto}_{\tilde{S}D} \tilde{T}_1^D$, which is a contradiction. Thus, there exists a $DB_k \not\sqsubseteq D$. Let, for all DB_i, $1 \leq i < k$, $DB_i \sqsubseteq D$ and further $DB_k \not\sqsubseteq D$. Hence, since transaction T_k executes on both DB_{k-1} and DB_k, $global(T_k, D)$. Consider the sequence of conflicts $T_k \leadsto_{DB_k} T_{k+1}$, $T_{k+1} \leadsto_{DB_{k+1}} T_{k+2}$, $\ldots$, $T_{n-1} \leadsto_{DB_{n-1}} T_n$, $T_n \leadsto_{DB_n} T_1$, $T_1 \leadsto_{DB_1} T_2$, $\ldots$, $T_{k-1} \leadsto_{DB_{k-1}} T_k$. Since $global(T_k, D)$, by Lemma 8, $\tilde{T}_k^D \overset{*}{\leadsto}_{\tilde{S}D} \tilde{T}_k^D$ which is a contradiction. Hence, the sequence of transactions cannot exist. Thus, S is serializable.

References

Batra RK, Georgakopoulos D, Rusinkiewicz M (1992) A decentralized deadlock-free concurrency control method for multidatabase transactions. In *Proceedings of the Twelfth International Conference on Distributed Computing Systems, Yokohoma, Japan*

Beeri C (1988) Schek H-J, Weikum G, Multilevel transaction management, theoretical art or practical need? In *International Conference on Extending Database Technology, Lecture Notes on Computer Science*, Vol. 303. Springer, Berlin Heidelberg New York, pp –

Bernstein PA (1987) Hadzilacos V, Goodman N, *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, Mass.

Breitbart Y, Tieman L (1985) Adds – heterogeneous distributed database system. In: Schreiber FA, Litwin W (ed) *Distributed Data Sharing Systems*. North Holland Publishing Company, pp –

Breitbart Y, Silberschatz A (1988) Multidatabase update issues. In *Proceedings of ACM-SIGMOD 1988 International Conference on Management of Data, Chicago*, pp 135–141

Breitbart Y, Silberschatz A, Thompson GR (1990) Reliable transaction management in a multidatabase system. In *Proceedings of ACM-SIGMOD 1990 International Conference on Management of Data, Atlantic City, New Jersey*, pp 215–224

Breitbart Y, Garcia-Molina H, Silberschatz A (1992a) Overview of multidatabase transaction management. *VLDB J*, 1:–

Breitbart Y, Silberschatz A, Thompson GR (1992b) Transaction management in a failure-prone multidatabase environment. *VLDB J*, 1:–

Du W, Elmagarmid AK (1989) Quasi serializability: a correctness criterion for global concurrency control in InterBase. In *Proceedings of the Fifteenth International Conference on Very Large Databases, Amsterdam*, pp 347–355

Elmagarmid AK, Du W (1989) Supporting value dependencies for nested transactions in interbase. Technical Report CSD-TR-885, Purdue University, Computer Sciences Department

Elmagarmid AK, Du W (1990) A paradigm for concurrency control in heterogeneous distributed database systems. In *Proceedings of the Sixth International Conference on Data Engineering, Los Angeles*

Georgakopoulos D, Rusinkiewicz M, Sheth A (1991) On serializability of multidatabase transactions through forced local conflicts. In *Proceedings of the Seventh International Conference on Data Engineering, Kobe, Japan*

Gligor V, Luckenbaugh GL (1984) Interconnecting heterogeneous database management systems. *IEEE Computer* :33–43

Gligor V, Popescu-Zeletin R (1985) Concurrency conctrol issues in distributed heterogeneous database management systems. In Schreiber FA, Litwin W (ed) *Distributed Data Sharing Systems*, North Holland Publishing Company, pp 43–56

Gligor V, Popescu-Zeletin R (1986) Transaction management in distributed heterogeneous database management systems. *Inform Syst* :287–297

Gray J, Reuter A (1993) *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Mateo, Calif.

Landers T, Rosenberg RL (1982) An overview of multibase. In: Schneider HJ (editor) *Distributed Data Bases*. North Holland Publishing Company

Lomet D, Salzberg B (1992) Access method concurrency and recovery. In *Proceedings of ACM-SIGMOD 1992 International Conference on Management of Data, San Diego, California*

Lomet D (1993) Key range locking. In *Proceedings of the Nineteenth International Conference on Very Large Databases, Dublin*

Mehrotra S, Rastogi R, Korth HF, Silberschatz A (1991) Non-serializable executions in heterogeneous distributed database systems. In *Proceedings of the First International Conference on Parallel and Distibuted Information Systems, Miami Beach, Florida*

Mehrotra S, Rastogi R, Breitbart Y, Korth HF, Silberschatz A (1992a) The concurrency control problem in multidatabases: Characteristics and solutions. In *Proceedings of ACM-SIGMOD 1992 International Conference on Management of Data, San Diego, California*

Mehrotra S, Rastogi R, Breitbart Y, Korth HF, Silberschatz A (1992b) Ensuring transaction atomicity in multidatabase systems. In *Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, California*

Mehrotra S, Rastogi R, Korth HF, Silberschatz A (1992c) Relaxing serializability in multidatabase systems. In *Proceedings of the Second International Workshop on Research Issues on Data Engineering: Transaction and Query Processing, Mission Palms, Arizona*

Mehrotra S, Rastogi R, Korth HF, Silberschatz A (1992d) A transaction model for heterogeneous distributed database systems. In *Proceedings of the Twelfth International Conference on Distributed Computing Systems, Yokohoma, Japan*

Mohan C (1989) ARIES/KVL: A key-value locking method for concurrency control of multiaction transactions operating on B-trees indexes. Technical Report RJ7008, IBM Research

Mohan C, Levine F (1989) ARIES/IM: An efficient and high concurrency index management method using write-ahead logging. Technical Report RJ6846, IBM Research

Moss JEB (1987) Nested transactions: An introduction. In: Bhargava B (ed) *Concurrency Control and Reliability in Distributed Systems*. Van Nostrand Reinhold, pp 395–425

Muth P, Rakow TC, Weikum G, Brossler P, Hasse C (1993) Semantic concurrency control in object-oriented database systems. In *Proceedings of the Ninth International Conference on Data Engineering, Vienna, Austria*, pp 233–242

Ouzzani M, Atroun MA, Belkhodja NL (1995) A top-down approach to two-level serializability. In *Proceedings of the 20th International Conference on Very Large Databases, Santiago, Chile*

Papadimitriou C (1986) *The Theory of Database Concurrency Control*. Computer Science Press, Rockville, Md.

Pu C (1988) Superdatabases for composition of heterogeneous databases. In *Proceedings of the Fourth International Conference on Data Engineering, Los Angeles*

Schek H-J, Weikum G, Schaad W (1991) A multilevel transaction approach to federated dbms transaction management. In *International Workshop on Interoperability in Multidatabase Systems*

Templeton M, Brill D, Hwang A, Kameny I, Lund E (1983) An overview of the mermaid system – a frontend to heterogeneous databases. In *Proceedings EASCON*, pp 387–402

Veijalainen J, Wolski A (1990) The 2PC agent method and its correctness. Technical Report Research Notes 1192, Technical research Centre of Finland

Weikum G, Schek H (1984) Architectural issues of transaction management in multi-layered systems. In: *Proceedings of the Tenth International Conference on Very Large Databases, Singapore*, pp 454–465

Weikum G, Schek H-J (1991) Concepts and applications of multilevel transactions and open nested transactions. In Elmagarmid AK (ed) *Advanced Transaction Models for new applications*. Morgan-Kaufmann, San Mateo, Calif.

Wolski A, Veijalainen J (1990) 2PC agent method: Achieving serializability in presence of failures in a heterogeneous multidatabase. In *Proceedings of the International conference on databases, parallel architectures and their applications*, pp 321–330

Zhang A, Nodine M, Bhargava B, Bukhres O (1994) Ensuring relaxed atomicity for flexible transactions in multidatabase systems. In *Proceedings of ACM-SIGMOD 1994 International Conference on Management of Data, Minneapolis, Minnesota*