1989

An Extended Transaction Model for Multidatabase Systems

Yungho Leu

Ahmed K. Elmagarmid
Purdue University, ake@cs.purdue.edu

Marek Rusinkiewicz

Report Number:
89-925

AN EXTENDED TRANSACTION MODEL FOR MULTIDATABASE SYSTEMS

Yungho Leu
Ahmed Elmagarmid
Marek Rusinkiewicz

CSD-TR 925
October 1989
(Revised April 1990)

An Extended Transaction Model for Multidatabase Systems[1]

Yungho Leu, Ahmed K. Elmagarmid
and Marek Rusinkiewicz[2]
Computer Sciences Department
Purdue University
West Lafayette, IN 47907

[1]This work is supported by a PYI Award from NSF under grant IRI-8857952 and grants from AT&T Foundation, SERC, Tektronix, and Mobil Oil.

[2]On leave from the University of Houston 1989-1990

Abstract

Transaction management in multidatabase systems presents numerous challenges due to the heterogeneity of the underlying databases and the need to preserve autonomy of local systems. On the other hand, the existence of multiple databases frequently makes it possible to accomplish the objective of a transaction by performing an action in one of several functionally equivalent databases. This property of multidatabase systems will be referred to as *function replication*. In this paper, we propose two extensions to the traditional transaction model for a multidatabase environment. They are: (1) allowing the composition of flexible global transactions by taking advantage of the function replication and (2) providing explicit control of the execution of the subtransactions of a global transaction. In the proposed model, a global transaction can tolerate a site failure by submitting a substitute subtransaction to another functioning site. We formally define the new transaction model and discuss the scheduling, concurrency control and commitment of the extended multidatabase transactions.

1 Introduction

A Multidatabase System(MDBS) is a facility that allows access to data stored in multiple autonomous and possibly heterogeneous database systems. Although transaction management in such systems has been a subject of extensive research, many problems remain unresolved because of the complexity caused by data distribution, heterogeneity and the need to preserve autonomy of the member database systems. The research has been concentrated mostly on developing concurrency control and commitment protocols suitable for these new environments [EH88][Pu88][AGMS87][BST87][BS88]. In another direction, the basic notion of multidatabase consistency has been reexamined, leading to the emergence of the new paradigms that reject serializability as the correctness criterion for multidatabase systems [LT88][DE89].

In this paper, we will concentrate on examining the semantics of multidatabase transactions and extending the existing transaction model to take into account the characteristics of this new environment. The proposed model seems to be more suitable for the multidatabase systems, since it allows the user to compose flexible global transactions which can tolerate failures of the local database systems.

The rest of this paper is organized as follows. In section 2, we examine the characteristics of the multidatabase environment and the problems of using the traditional transaction model in this environment. In section 3, we define the execution dependency among the set of subtransactions of a global transaction. The execution dependency is a fundamental concept of the extended transaction model. In section 4, we formally define the extended transaction model. In section 5, we discuss the problems of transaction scheduling and concurrency control which must be solved in order to apply the new model in a MDBS environment. Section 6 presents a summary of the paper.

2 Background

2.1 The Characteristics of a MDBS Environment

Although multidatabase transactions are similar to the regular distributed database transactions, their execution in a MDBS is quite different. The

difficulty in executing a global transaction in a MDBS is due to the following characteristics of the MDBS:

Autonomy Requirements. The requirement of autonomy of the local systems states that in building a multidatabase system, the local database system must not be changed in any respect. The following aspects of local autonomy have been identified in the literature: [DELO89][GMK88][EV87].

- *Design Autonomy*: The design autonomy allows heterogeneity of the Local Database Systems (LDBS) in terms of hardware, operating system, communication protocol, data model, transaction management mechanism, etc. The design autonomy reflects the fact that the LDBS were built independently of each other, possibly at different times.
- *Execution Autonomy*: The execution autonomy refers to the ability of a local database system to decide whether to execute a transaction that has been submitted to it, or to reject it. For example, a subtransaction of a multidatabase transaction can be rejected because of a conflict with a local transaction that has not passed through the multidatabase interface.
- *Communication Autonomy*: Communication autonomy states that the local database system may not respond to a query or subtransaction. This may be due to a communication failure, or because the local database system is simply not available at this time.
- *Naming Autonomy*: Naming autonomy states that, in different LDBSs, different names can be used to denote analogical objects.

The effects of these autonomy requirements on the execution of global transactions will be discussed in the next two sections.

Unreliable Environment. The MDBS environment is inherently unreliable in the sense that frequently a global transaction can not be executed successfully due to problems caused by data distribution, heterogeneity or local autonomy. Some of these problems are common to all heterogeneous distributed computing environments (e.g. site and communication failures), but some of them are unique for the multidatabase systems.

For example, due to the *design autonomy*, the pre-existing hardware and/or software in some local sites may be very unreliable. A local database may refuse to communicate with the outside world due to the *communication*

autonomy, which will fail a global transaction if it has a subtransaction submitted to this local database. Due to the *execution autonomy*, a local site may refuse to execute a subtransaction of a global transaction, which will result in the failure of the global transaction.

Potential for long-lived transactions. A multidatabase transaction is potentially long-lived since its execution time may span over a long period of time. There are several factors which may contribute to the long execution time of a global transaction. Frequently, a multidatabase transaction involves sites and LDBSs with vastly different speeds and capabilities. In order to enforce agreement protocols in such environment (such as in the case of deciding whether a MDBS transaction can be committed), the MDBS software must proceed at a rate determined by the slowest member LDBS that may run on a microcomputer and communicate over serial lines. Similarly, due to the execution autonomy, a local database may decide to delay the execution of a subtransaction of a global transaction, which in turn delays the completion of the global transaction. Finally, the communication delay on the heterogeneous communication network may also contribute to the long execution time of a global transaction. These factors may delay the completion of a global transaction even though the global transaction is not long-lived by itself.

2.2 Transaction Model for MDBS

The most widely accepted transaction processing model for a MDBS has been proposed by Gligor and Popescu-Zeletin in [GPZ86]. The following are the main assumptions made in this model:

- A MDBS consists of a set of local database management systems.
- A global transaction is a transaction which accesses more than one local database system, while a local transaction accesses only one database system.
- A global transaction is composed of a set of subtransactions. Each subtransaction accesses one local database system on behalf of the global transaction.
- Each global transaction can have at most one subtransaction per local database system.

- The execution of a global transaction is controlled by a module called the *Global Transaction Manager* (GTM). The execution of the subtransactions and the local transactions are controlled by modules in the local database systems called the *Local Transaction Managers* (LTMs).

- A global transaction, when submitted, is decomposed into a set of subtransactions which, in turn, are submitted to the local database systems for execution. The GTM performs the concurrency control and commitment control on the execution of the global transactions to ensure the serializability and atomicity properties of the global transaction.

The above transaction processing model, as shown in Figure 1, suggests that a global transaction can be viewed as a two-level nested transaction [Mos81]. However, the nested transaction model is not suitable for the MDBS environment for two reasons. Firstly, the nested transaction model assumes that the parent transaction exercises full control over the execution of local transactions, for example by inheriting or releasing their locks. This assumption is clearly not realistic in an environment consisting of heterogeneous and autonomous local systems. Secondly, in a nested transaction, the execution dependency between subtransactions of a global transaction is not clearly defined and is implied in the semantics of the transaction.

2.3 The Limitations of the Existing Transaction Model and Possible Extensions

The existing transaction model can not capture some special characteristics of the transaction processing in a MDBS environment. As discussed above, the long delay and low reliability of the MDBS environment may adversely affect the execution of global transactions. This can be illustrated by the following simple example:

Example 1: Consider a global transaction G_1 which is composed of three subtransactions, as presented in Figure 2. Let us assume that the three subtransactions are submitted to the corresponding local database systems at the same time and that the probability for G_{11}, G_{12} and G_{13} to succeed is the same and is $\frac{1}{2}$. Assuming that the probabilities of success at each site are independent of each other, the probability that G_1 will succeed is only $\frac{1}{8}$. Now let us suppose, that for each subtransaction there are two sites

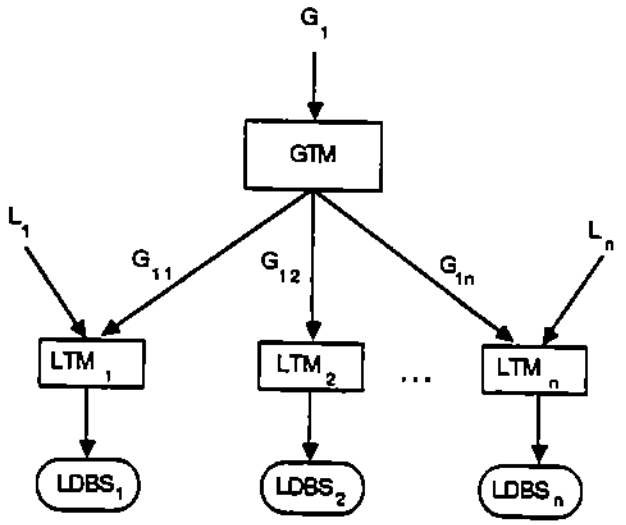

Figure 1: The existing MDBS transaction processing model

capable of implementing it, with the same probability of success. In this case, the probability of success for the global transaction becomes $\frac{27}{64}$, which is significantly higher. □

Another weakness of the existing transaction model is its lack of explicit submission control of the subtransactions of a global transaction. Using the same example, let us suppose that G_{13} has to wait for the result from G_{11} and G_{12} before it can start performing its computations. According to the existing transaction model, G_{13} will be submitted to the local database and wait until G_{11} and G_{12} complete. Although this waiting is not harmful by itself, G_{13} may unnecessarily occupy some of the local resources that can be used by other transactions. Furthermore, G_{13} may fail or be aborted, while it is waiting, thus resulting in the whole global transaction being aborted. It can be argued that a much better policy is to submit G_{13} after G_{11} and G_{12} have finished.

Based on the above observations, we propose the following two extensions to the existing transaction model:

1. Allow the user to compose flexible global transactions by taking advantage of the possibility of executing a subtransaction at one of several

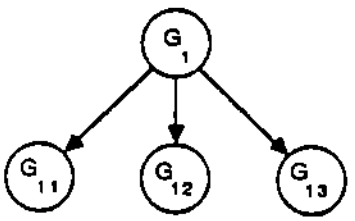

Figure 2: Example 1

functionally equivalent sites. We refer to this property of MDBS as function replication.

2. Provide explicit submission control of the subtransactions of a global transaction.

These extensions require that the user clearly specifies the set of subtransactions which constitute a global transaction, the execution dependency among the subtransactions and the function replication information. This information can be then used to schedule the execution of the subtransactions of the global transaction.

3 Execution Dependency

3.1 Function Replication

In a MDBS environment, frequently there is more than on local database system which can be used to implement a desired function of an application. We will call this property *function replication*. For a specific function t_i, the

local database systems which can be used to implement t_i are said to be *functionally replicated* for t_i. Function replication is an inherent characteristic of a MDBS. We can often take advantage of the function replication to compose a global transaction in such a way that it can achieve its objectives even if one or more of its subtransactions fail. As an example, let us consider a travel agent system [Gra81]; a transaction in this system may consist of the following operations:

1. Customer calls the agent to schedule a trip.
2. Agent negotiates with airlines for flight tickets.
3. Agent negotiates with car rental companies for car reservations.
4. Agent negotiates with hotels to reserve rooms.
5. Agent receives tickets and reservations and then gives them to the customer.

Let us assume that the only applicable airlines are Northwest and United, the only car rental company in the customer's destination city is Hertz and the only hotels in the city are Hilton, Sheraton and Ramada. The travel agent can order a ticket from either Northwest or United, so the Northwest and the United databases are functionally replicated for ordering a ticket. Similarly, the agent can reserve a room for a customer at any of the three hotels, so the databases of Hilton, Sheraton and Ramada are replicated for the hotel reservation function. Based on the function replication, the travel agent may construct a global transaction to accomplish the transaction's objectives as follows:

Subtransaction	*Action*	*Condition*
t_1	Order a ticket at Northwest Airlines	
t_2	Order a ticket at United Airlines	if t_1 fails
t_3	Rent a car at Hertz	
t_4	Reserve a room at Hilton	if t_5 fails
t_5	Reserve a room at Sheraton	
t_6	Reserve a room at Ramada	if t_4 and t_5 fail.

Several dependencies exists in the execution of the subtransactions. One of the dependencies is the relationship between t_1 and t_2, stating that t_2 should

be executed only if t_1 fails. Another dependency is the relationship among t_1, t_2 and t_3. If we assume that the flight information must be available before renting a car, t_3 can not be executed until either t_1 or t_2 are successfully completed. By providing alternatives for implementing a specific function, the global transaction becomes more tolerant to the failures which may prevent some subtransactions from being successfully executed. We believe that the above global transaction corresponds more closely to the actions of the travel agent in real life.

3.2 Execution Dependency among Subtransactions

In order to define an extended distributed transaction, we have to specify the dependency among the subtransactions. A *positive dependency* between subtransactions exists if a subtransaction can not be executed until another subtransaction is successfully completed. This occurs, for example, if a subtransaction must wait for the results from another subtransactions [ED89]. The relationship among subtransactions t_1, t_2 and t_3 of the travel agent transaction is an example. t_3 has to wait until either t_1 or t_2 succeed, before it can be executed. If two subtransactions are functionally replicated, one of them must wait for the failure of the other one, before it can be executed. This is called a *negative dependency*. The relationship between subtransactions t_1 and t_2 is an example of such dependency. All execution dependencies among the subtransactions of the travel agent transaction are shown in the graph in Figure 3. In order to specify the dependency among the subtransactions, we define a transaction failure state as follows.

Definition 1 *A transaction failure state s of a global transaction T with m subtransactions is an m-dimensional array* $(s_1, s_2, ..., s_m)$ *where*

$$s_i = \begin{cases} 0^1 & \textit{if subtransaction } t_i \textit{ has failed} \\ 1 & \textit{otherwise} \end{cases}$$

The boolean variable s_i is called the *failure state variable* of the subtransaction t_i. The set of all possible transaction failure states for a global transaction T is denoted by S. A subtransaction is considered to fail if the site on which the subtransaction is executed has failed before the subtransaction

[1] 0 stands for false and 1 stands for true.

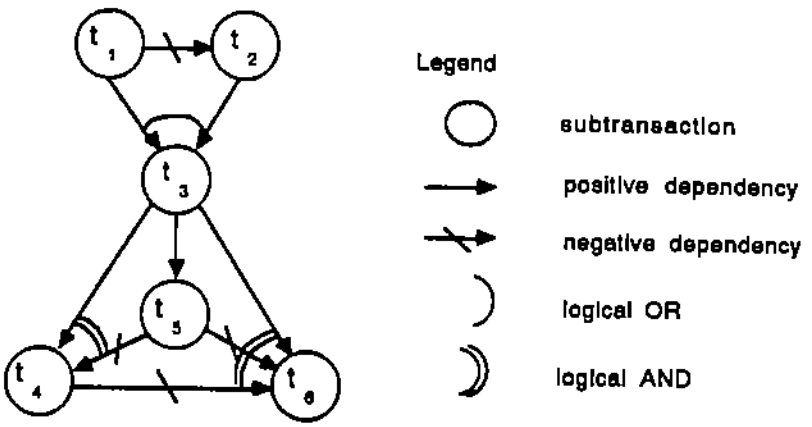

Figure 3: The execution dependency in the travel agent transaction

starts its execution, or the execution fails during the execution of the subtransaction. In order to simplify the specification of the dependencies among the subtransactions, the failure state of all subtransactions is initialized to 1 before they are actually executed.

In order to express the dependency, we associate with each subtransaction t_i a predicate p_i. The predicate is defined on the transaction failure state and can take the value of 1 or 0. We formally define a subtransaction predicate as follows:

Definition 2 *A subtransaction predicate p_i for a subtransaction t_i is a predicate defined on S, where*

$$p_i : S \rightarrow \{1, 0\}$$

We define the *simple positive dependency* as s_i for any subtransaction t_i. If $p_j = s_i$, then t_j has to wait until t_i is successfully executed. We also define the *simple negative dependency* as $\overline{s_i}$ for any subtransaction t_i. If $p_j = \overline{s_i}$, then t_j can not be executed until t_i is executed and fails. It is easy to see that all dependencies can be expressed as the boolean combination of these simple

dependencies. For example, the dependencies of the subtransactions of the travel agent transaction can be expressed by the following set of predicates:

$$P : \begin{cases} p_1 = 1 \\ p_2 = \overline{s_1} \\ p_3 = (s_1 \vee s_2) \\ p_4 = s_3 \wedge \overline{s_5} \\ p_5 = s_3 \\ p_6 = s_3 \wedge \overline{s_4} \ \wedge \overline{s_5} \end{cases}$$

It can be observed from the set of predicates, that t_1 does not depend on any other subtransactions; t_2 depends on the failure of t_1; t_3 depends on the success of either t_1 or t_2; t_4 depends on the failure of t_5 and the success of t_3; t_5 depends on the success of t_3; and finally, t_6 depends on the success of t_3 and the failure of both t_4 and t_5.

4 Extended Transaction Model

The fundamental idea of the extended transaction model is to associate with each subtransaction t_i of a global transaction a predicate p_i. The predicate p_i is defined on the transaction failure state and is used to determine the condition under which the subtransaction can be scheduled[2]. The value of the subtransaction predicate changes with the transaction failure state, as the global transaction executes. We formally define a global transaction as follows.

Definition 3 *A global transaction is a 4-tuple (T, O, P, S) where*

- *T is the set of all subtransactions of the global transaction*
- *O is a partial order on T*
- *P is the set of all predicates which are associated with the subtransactions of T*
- *S is the set of all possible transaction failure states of T*

[2] A subtransaction is said to be scheduled if it is submitted to the local site and executed by the local site, or such an attempt is made and the local site fails before it is submitted.

The partial order O on T is determined by the transaction semantics, which specifies the execution order of the set of subtransactions. The order O can be obtained from the dependency relationship and is used here to specify the execution rules of the global transaction. In terms of our model, the travel agent transaction can be specified as follows:

Example 2:

$$\mathrm{T} = \{t_1, t_2, t_3, t_4, t_5, t_6\}$$
$$\mathrm{O} : t_1 \prec t_3,\ t_2 \prec t_3,\ t_3 \prec t_4,\ t_3 \prec t_5,\ t_3 \prec t_6$$

$$P : \begin{cases} p_1 = 1 \\ p_2 = \overline{s_1} \\ p_3 = (s_1 \vee s_2) \\ p_4 = s_3 \wedge \overline{s_5} \\ p_5 = s_3 \\ p_6 = s_3 \wedge \overline{s_4} \ \wedge \overline{s_5} \end{cases}$$

$$\mathrm{S} = \{\ (1, 1, 1, 1, 1, 1), (0, 1, 1, 1, 1, 1), \cdots\}$$

The above 4-tuple defines the static structure of the global transaction. Before specifying the dynamic structure of a global transaction we will introduce some additional definitions.

Definition 4 *For a subtransaction t_i, its predecessors are those subtransactions which precede t_i in the partial order O. We will use pred(t_i) to denote the set of predecessors of t_i, i.e.*

$$pred(t_i) = \{t_j |\ t_j \in T \text{ and } t_j \prec t_i \text{ in } O\}.$$

For a given failure state s, we define a subtransaction t_i as executable if

1. t_i has not been scheduled yet;
2. $\forall\ t_k \in pred(t_i)$, either t_k is scheduled or $p_k(t_k)$ is false; and
3. $p_i(s)$ is true.

In order to monitor the execution of a global transaction, we define the transaction execution state as follows.

Definition 5 *An execution state, x, of a global transaction with m subtransactions is an m-dimensional array* $(x_1, x_2, ..., x_m)$ *where*

$$x_i = \begin{cases} 1 & \textit{if subtransaction } t_i \textit{ is successfully executed} \\ 0 & \textit{otherwise} \end{cases}$$

The set of all the possible execution states is denoted by X. The x_i's are initialized to 0 before the global transaction starts its execution. The execution state changes as some of the subtransactions are executed successfully. At a certain point of the execution, the objective of the global transaction is achieved, and the global transaction is said to be successfully completed. The execution state in which the objective of the global transaction is achieved is called an *acceptable state*. The set of all acceptable states of a global transaction is called the *acceptable state set* and is denoted by A.

Definition 6 *The acceptable state set, A, of a global transaction is a subset of X, where*

$$A = \{\, x \mid x \in X, \textit{ and in state } x, \textit{ the objective of } T \textit{ is achieved} \,\}$$

For the travel agent transaction, the acceptable state set A is shown below.

$$\begin{aligned} \mathrm{A} = \{\ & (1, 0, 1, 1, 0, 0), \\ & (1, 0, 1, 0, 1, 0), \\ & (1, 0, 1, 0, 0, 1), \\ & (0, 1, 1, 1, 0, 0), \\ & (0, 1, 1, 0, 1, 0), \\ & (0, 1, 1, 0, 0, 1)\ \} \end{aligned}$$

The execution algorithm of a global transaction T can be now formulated as follows:

1. The execution starts from the initial execution state of T;

2. The executable subtransactions of T are scheduled until the termination condition has been reached;

3. When the execution of a subtransaction t_i is completed, s_i is set to 1 if t_i is successfully executed, otherwise set it to 0. x_i is modified in the same way;

4. If the current execution state is acceptable, or none of the subtransactions is executable, the algorithm terminates.

Before the system can schedule a subtransaction of a global transaction for execution, it has to check the transaction failure state and the transaction execution state. The rules allow concurrent scheduling of several subtransactions for execution if they are executable at the same time.

Definition 7 *A transaction state is an ordered pair (s, x), where $s \in S$ and $x \in X$.*

Definition 8 *We define a positive execution step, denoted by e, to be a successful execution of a subtransaction.*

$$e : X \rightarrow X$$

Hence, a positive execution step is a mapping from an execution state into another execution state.

Definition 9 *We define a negative execution step, denoted by f, to be a failure in the execution of a subtransaction.*

$$f : S \rightarrow S$$

A negative execution step maps a failure state into another failure state. We define a transaction execution step in terms of the positive and the negative execution steps as follows:

Definition 10 *A transaction execution step β is defined on the transaction state.*

$$\beta : S \times X \rightarrow S \times X, \text{ where}$$

$$\beta(s,x) = \begin{cases} (s, e(x)) & \textit{if } \beta \textit{ is a positive execution step } e, \\ (f(s), x) & \textit{otherwise (} \beta \textit{ is a negative execution step } f) \end{cases}$$

We define an execution, denoted by E, of a global transaction to be a composition of the transaction execution steps applied to the initial transaction state:

$$\mathrm{E} : \beta_1 \circ \beta_2 \circ \cdots \circ \beta_m$$

where

$$\begin{aligned} \mathrm{E}(s,x) &= \beta_1 \circ \beta_2 \circ \cdots \circ \beta_m(s,x) \\ &= \beta_2 \circ \beta_3 \circ \cdots \circ \beta_m(\beta_1(s,x)) \\ &= \cdots \\ &= \beta_m(\beta_{m-1} \cdots \beta_1(s, \overbrace{x)) \cdots)}^{m} \end{aligned}$$

Definition 11 *An execution of a global transaction T is successful, iff*

- *it has been carried out in accordance with the execution algorithm specified above to the termination of the algorithm, and*
- *the final transaction state of T is in A.*

Otherwise the execution of T is called unsuccessful.

Example 3: Three transaction executions of the travel agent transaction are shown below.

$$\begin{aligned} E_1 : &((1,1,1,1,1,1),(0,0,0,0,0,0)) \rightarrow \\ &((1,1,1,1,1,1),(1,0,0,0,0,0)) \rightarrow \\ &((1,1,1,1,1,1),(1,0,1,0,0,0)) \rightarrow \\ &((1,1,1,1,1,1),(1,0,1,0,1,0)) \ \square \end{aligned}$$

$$E_2 : ((1,1,1,1,1,1),(0,0,0,0,0,0)) \rightarrow$$
$$((1,1,1,1,1,1),(1,0,0,0,0,0)) \rightarrow$$
$$((1,1,1,1,1,1),(1,0,1,0,0,0)) \rightarrow$$
$$((1,1,1,1,0,1),(1,0,1,0,0,0)) \rightarrow$$
$$((1,1,1,1,0,1),(1,0,1,1,0,0)) \Box$$

$$E_3 : ((1,1,1,1,1,1),(0,0,0,0,0,0)) \rightarrow$$
$$((1,1,1,1,1,1),(1,0,0,0,0,0)) \rightarrow$$
$$((1,1,0,1,1,1),(1,0,0,0,0,0)) \Box$$

E_1 is a normal execution without errors. Three subtransactions executed are t_1, t_3 and t_5, which represents the customer getting a ticket from the Northwest Airlines, renting a car from the Hertz, and staying in the Sheraton Hotel. E_2 is an execution in which an execution error occurs in executing t_5. The final result is still acceptable, and represents the customer getting a ticket from the Northwest Airlines, renting a car from Hertz and staying in Hilton hotel. E_3 is an unacceptable execution. An execution error in executing t_3 has prevented the global transaction from being successfully completed.

Since the execution of the global transaction is not deterministic, it requires a special control mechanism, which will be discussed in the next section.

5 Application of the New Model to the MDBS Environment

In order to apply the new model to the MDBS environment, three fundamental problems have to be solved. They are subtransaction scheduling, global concurrency control and the global atomicity control. The purpose of the subtransaction scheduling is to schedule the execution of the subtransactions of a global transaction given the dependency structure of the global transaction and the current failure state of the MDBS. The global concurrency control and the global atomicity control are enforced in order to maintain the basic properties of global transactions, namely *atomicity* and *isolation* [HR83]. The atomicity property refers to the transaction having its complete desired effect, or no effect at all. This property is usually enforced by a commit protocol. The isolation property refers to the transaction having

no interference with other transactions. This property is maintained by the concurrency control protocol. In this section, we briefly discuss these issues.

5.1 Subtransaction Scheduling

Because of the execution dependencies, the requirements for subtransaction scheduling are different in this new model than in other transaction models, such as the *nested transaction model* [Mos81] or *CAD transaction model* [KKB88]. In these models, subtransaction invocation is determined by the application and the execution order of subtransactions is fixed. In contrast, under our model the execution of subtransactions of a global transaction is dynamic and nondeterministic. Hence, a new scheduling algorithm is required for the new model. This algorithm must be able to capture the different dependency structure of the global transaction, to monitor the failures of subtransactions and to evaluate their effects on the execution of the global transaction. This problem is under investigation now.

5.2 Commitment

In the existing distributed transaction model, a global transaction is atomic if either all or none of its subtransactions are completed (or have their effects) on the local sites. This *"all or nothing"* property has to be modified in the new model. For a global transaction in the new model, there is more than one combination of subtransactions, which when completed can achieve the objective of the global transaction. We define the *functional atomicity* of a global transaction as follows.

Definition 12 *An execution of a global transaction is functionally atomic if it terminates and the subtransactions are committed/aborted in accordance with the rules below:*

1. *When an acceptable state has been reached, all subtransactions t_i, such that $x_i = 1$ in x, are committed and all remaining subtransactions in progress (if any) are aborted.*
2. *When the execution algorithm terminates without reaching an acceptable state, all subtransactions t_i are aborted.*

The functional atomicity constitutes a new atomicity criterion for our model. To ensure the functional atomicity property of the global transaction, a new commit protocol is required. The commit protocol has to communicate with the scheduling algorithm in order to know if an acceptable state has been reached or if the global transaction has failed, before it can decide to commit the global transaction or to abort it. This problem is also under investigation now.

5.3 Concurrency Control

A generally accepted approach to the global concurrency control without violating local autonomy [GL84], is to impose a control hierarchy between the global and the local concurrency controller. A class of hierarchical concurrency control protocols in a heterogeneous multidatabase environment is also discussed in [ED90]. In [EL89] we argued that, under the assumption that the local concurrency control algorithms are **static**[3], it is sufficient for the global concurrency controller to maintain the compatibility of the serialization orders of the subtransactions of all committed global transactions. In this section, we adapt the *site queue* algorithm proposed in [EL89] to the requirements of the extended global transactions.

Under the extended global transaction model, the subtransactions of a global transaction are not submitted to the local database systems at the same time. Instead, they are submitted dynamically by the subtransaction scheduling algorithm. The global concurrency control can be carried out as follows (Figure 4):

For every local database system, we create a *server* which maintains a *subtransaction queue* and a *site timestamp*. Each global transaction is assigned a unique timestamp (e.g. a clock value when the global transaction is submitted). All subtransactions of a global transaction receive the timestamp of the global transaction. The subtransactions are then submitted to the servers dynamically. The local site timestamp records the transaction timestamp of the latest subtransaction which has been submitted to the LTM. The server receives the subtransactions from the GTM and submits them to the LTM according to the following rules:

[3] A concurrency control algorithm is static if, for any transaction under its control, the serialization order of the transaction is determined in its lifetime.

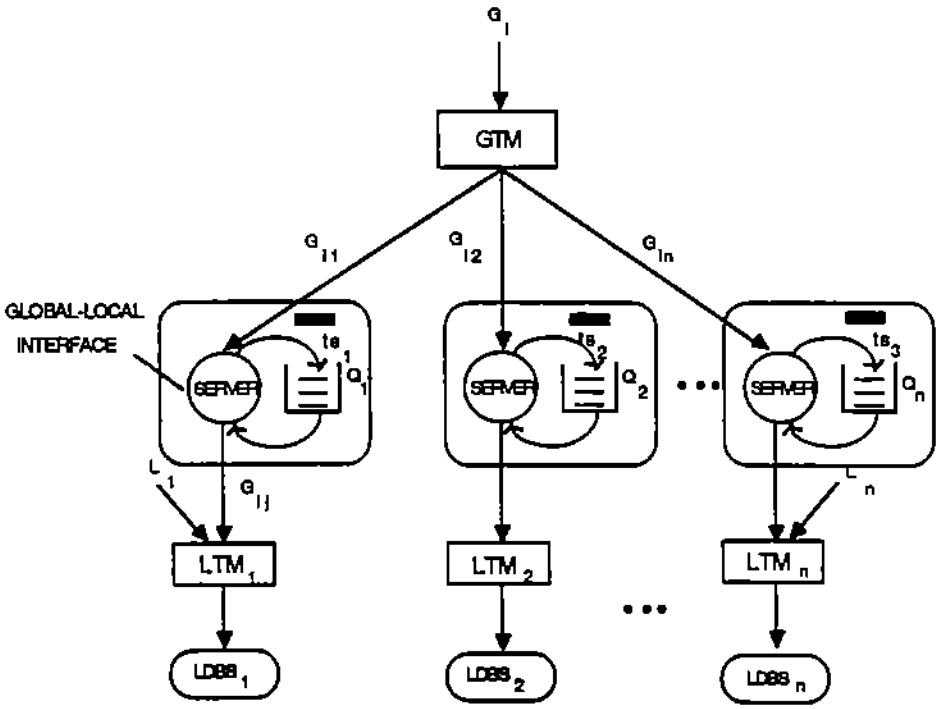

Figure 4: The adapted site queue algorithm

Receiving Rule: When the server receives a subtransaction, it compares the timestamp of the subtransaction with the timestamp of the site. If the timestamp of the subtransaction is earlier than the site timestamp, the subtransaction is aborted, otherwise, the subtransaction is inserted in a proper position in the queue (according to the timestamp order).

Submitting Rule: The server can submit a subtransaction to the LTM for execution only when the subtransaction is in the front of the queue and the previously submitted subtransaction is serialized or aborted.

The adapted site queue algorithm maintains the compatibility of the serialization orders of the global transactions, since for any two global transactions G_i and G_j, if the timestamp of G_i precedes the timestamp of G_j, then all the subtransactions of G_i will be serialized before those of G_j.

6 Conclusion

We have proposed an extended distributed transaction model which, we believe, is more suitable for multidatabase systems. The proposed extensions

address some of the problems caused by the specific characteristics of the MDBS environment, namely requirements of local autonomy, relatively low reliability and possibility of long lived transactions. We have defined the function replication in a MDBS and we have proposed modifications to the transaction processing model that allow the user to compose flexible global transactions which take advantage of this property. Under the new model, a global transaction can tolerate failures of subtransactions, if a function of the failed subtransaction can be implemented on another functioning site.

We have also introduced and formally defined the notion of execution dependency which can be used to explicitly control the submission of subtransactions of a global transaction. The proposed extensions affect the execution control, commitment and concurrency control of global transactions. We have defined functional atomicity of a transaction and used it to provide a new condition for the commitment of a global transactions. A new concurrency control scheme suitable for the new model has been also introduced.

The paper presents the basic components of the extended transaction model. However, additional problems related to subtransaction scheduling, global commitment control and the global concurrency control have to be solved before the new model can be used in practice.

References

[AGMS87] R. Alonso, H. Garcia-Molina, and K. Salem. Concurrency control and recovery for global procedures in federated database systems. In *IEEE Data Engineering*, pages 5–11, September 1987.

[BS88] Y. Breitbart and A. Silberschatz. Multidatabase systems with a decentralized concurrency control scheme. In *Distributed Processing Technical Committee–NEWSLETTER*, pages 35–41, November 1988.

[BST87] Y. Breitbart, A. Silberschatz, and G. Thompson. An update mechanism for multidatabase systems. In *IEEE Data Engineering*, pages 135–142, 1987.

[DE89] W. Du and A. Elmagarmid. Quasi serializability: a correctness criterion for global concurrency control in interbase. In *Proceedings of the International Conference on Very Large Data Bases*, Amsterdam, The Netherlands, August 1989.

[DELO89] W. Du, A. Elmagarmid, Y. Leu, and S. Ostermann. Effects of autonomy on global concurrency control in heterogeneous distributed database systems. In *Proceedings of the Second International Conference on Data and Knowledge Systems for Manufacturing and Engineering*, Gaithersburg, MD, October 1989.

[ED89] A. Elmagarmid and W. Du. Supporting value dependency for nested transactions in interbase. Technical Report CSD-TR-885, Purdue University, May 1989.

[ED90] A. Elmagarmid and W. Du. A paradigm for concurrency control in heterogeneous distributed database systems. In *Proceedings of the Sixth International Conference on Data Engineering*, February 1990.

[EH88] A.K. Elmagarmid and A.A. Helal. Supporting updates in heterogeneous distributed database systems. In *Proceedings of the International Conference on Data Engineering*, 1988.

[EL89] A. Elmagarmid and Y. Leu. A hierarchical approach to concurrency control for multidatabases. Technical Report CSD-TR-919, Department of Computer Science, Purdue University, 1989.

[EV87] F. Eliassen and J. Veijalainen. Language support for mutidatabase transactions in a cooperative, autonomous environment. In *TENCON '87, IEEE Regional Conference*, Seoul, 1987.

[GL84] V.D. Gligor and G.L. Luckenbaugh. Interconnecting heterogeneous data base management systems. *IEEE Computer*, 17(1):33–43, January 1984.

[GMK88] H. Garcia-Molina and B. Kogan. Node autonomy in distributed systems. In *Proceedings of the International Conference on Data Engineering*, pages 158–166, 1988.

[GPZ86] V.D. Gligor and R. Popescu-Zeletin. Transaction management in distributed heterogeneous database management systems. *Information Systems*, 11(4):287–297, 1986.

[Gra81] Jim Gray. The transaction concepts: Virtues and limitations. In *Proceedings of the International Conference on Very Large Data Bases*, pages 144–154, 1981.

[HR83] T. Haerder and A. Reuter. Principles of transaction-oriented database recovery. *ACM Computing Serveys*, 15(4):287–317, December 1983.

[KKB88] H. Korth, W. Kim, and F. Bancilhon. On long-duration CAD transaction. *Information Sciences*, 46(1-2):73–108, 1988.

[LT88] W. Litwin and H. Tirri. Flexible concurrency control using value dates. *IEEE Distributed Processing Technical Committee Newsletter*, 10(2):42–49, November 1988.

[Mos81] J.E. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing.* PhD thesis, Dept. of Electrical Engineering and Computer Science, MIT, April 1981.

[Pu88] C. Pu. Superdatabases for composition of heterogeneous databases. In *Proceedings of the International Conference on Data Engineering*, 1988.