1990

A Multidatabase Transaction Model for InterBase

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

Y. Leu

W. Litwin

Marek Rusinkiewicz

Report Number:
90-968

A MULTIDATABASE TRANSACTION MODEL FOR INTERBASE

Ahmed K. Elmagarmid
Y. Leu
W. Litwin
Marek Rusinkiewicz

CSD-TR 968
March 1990

A Multidatabase Transaction Model for InterBase[1]

A. K. Elmagarmid, Y. Leu, W. Litwin†[2] and M. Rusinkiewicz[3]

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
e-mail {ake, yhl, mr}@cs.purdue.edu

† Computer Science Department
Stanford University
Margaret Jacks Hall
Stanford, CA 94305

[1]This work is supported by a PYI Award from NSF under grant IRI-8857952 and grants from AT&T Foundation, Tektronix, SERC and Mobil Oil. The work by Litwin is supported by DARPA under contract N39-84-C-211, task 24.

[2]On leave from INRIA

[3]On leave from the University of Houston

Abstract

The management of multidatabase transactions presents new and interesting challenges, due mainly to the requirement of the autonomy of local database systems. In this paper, we present an extended transaction model which provides the following features useful in a multidatabase environment: (1) It allows the composition of *flexible transactions* which can tolerate failures of individual subtransactions by taking advantage of the fact that a given function can frequently be accomplished by more than one database system; (2) It supports the concept of *mixed transactions* allowing compensatable and non-compensatable subtransactions to coexist within a single global transaction; and (3) It incorporates the concept of *time* in both the subtransaction and global transaction processing, thus allowing more flexibility in transaction scheduling. We formally define the extended transaction model and discuss its transaction scheduling mechanism.

1 Introduction

The InterBase project in the department of Computer Science at Purdue University investigates multidatabase management systems. The prototype currently links the database systems Ingres, GURU, Sybase and DBASE IV, running on various hardware platforms and operating systems. Using an InterBase language called DOL, users write global programs accessing autonomous databases and other software systems. [ROEL90].

The problem of transaction processing involving data in multiple autonomous and possibly heterogeneous database systems has received more attention recently. Several concurrency control, commitment and recovery schemes for the multidatabase environment have been proposed in the literature [ED90], [EH88], [LE90], [Pu88], [BST87], [BS88], [AGS87], [WV90] [EVT88]. Most of the work in this area has been performed in the context of the traditional transaction models, assuming two-level nested transactions [Mos81], [GP86] and using serializability as a correctness criterion. However, it has been argued in [EVT88] [LER89] that these models may not suffice for the environment consisting of cooperating autonomous systems. The traditional requirements of *atomicity*, *consistency*, *isolation* and *durability* [Gra81] [HR83] may be too difficult to enforce or inappropriate when multiple databases are involved. We propose a transaction model especially designed for this new environment.

A fundamental characteristic of a multidatabase system is the autonomy of the participating database systems [EV87], [GK88], [DELO89]. The autonomy requirements have profound effect on the ability of a multidatabase system to support atomic transactions, and its performance [DEK90]. Due to design autonomy, the control of availability shifts to the local systems. A local system may choose to delay a subtransaction or even refuse its execution. This would delay the completion of a multidatabase transaction or would inhibit its success if the traditional criteria are used. The response time of different local systems may also differ by orders of magnitude simply because the sites and local database systems have different processing speeds and capabilities. A traditional transaction would be forced to proceed at the rate of the slowest system.

The new environment makes it difficult or impossible to complete a transaction if the traditional criteria are enforced. Extensions to the known transaction models are required. We propose a new model which is used in our

InterBase prototype. Failures of subtransactions in a flexible transaction are tolerated by taking advantage of the fact that a given function can frequently be accomplished by more than one database system. Furthermore, compensatable and non-compensatable subtransactions can coexist within a single global transaction. Finally, time used in conjunction with subtransaction and global transaction processing can be exploited in transaction scheduling.

In this paper, we formally define the new model, and describe an implementation of the scheduling mechanism using Predicate Petri Nets. The rest of this paper is organized as follows. In section 2, we discuss the new requirements on transaction processing in multidatabase systems. In section 3, we formally describe the new transaction model. In section 4, we present a global transaction scheduling algorithm, using the Predicate Petri Nets. Section 5 concludes this paper.

2 Extending the transaction semantics

To deal with the specific requirements of the multidatabase environment, we incorporate additional features in the new transaction model. Although a global transaction in our model is syntactically a two level nested transaction, its semantics is significantly expanded. The extensions go in three basic directions.

- We take advantage of the fact that in a multidatabase system a given objective can be frequently accomplished by submitting a functionally equivalent (sub-)transaction to one of several available local systems. This property (referred to as *function replication*) [LER89] [RELL90] allows the user additional flexibility in composing global transactions (section 2.1).
- Some subtransactions in a multidatabase system may allow their effects to be semantically "undone", after they are committed, by their corresponding compensating subtransactions. In the model, we take advantage of this fact by allowing some subtransactions to be committed before their corresponding global transaction is committed. Transactions allowing a combination of both compensatable and non-compensatable subtransactions are called *mixed transactions* (section 2.2).

- We also allow the specification of the value of completion time for the execution of (sub-)transactions. This information can be then used to schedule the execution of the global transactions (section 2.3).

These features are explained in greater detail below.

2.1 Function Replication

In contrast to conventional distributed database systems, a multidatabase system is composed of independently created and administered database systems. This kind of environment usually allows a user to perform a given task on more than one local database system. For example, if multiple car rental databases are available to the users of a multidatabase system, then a user can perform the (functionally equivalent) rent-a-car task in any of the member databases providing this service. Another example is the banking environment such as the S.W.I.F.T [EV87], where a customer can choose to withdraw money from any of the participating banks. Since this kind of flexibility seems to be quite common in multidatabase environments, it is highly desirable to be able to capture it in the transaction model. In the new model, flexibility is supported by allowing the user to specify alternative subtransactions for implementing the same task or specifying alternative sources of data. This can be further illustrated by the following example.

Example 1: Consider a travel agent information system [Gra81]; a transaction in this system may consist of the following subtasks:

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 for the purpose of this trip the only applicable airlines are Northwest and United, the only car rental company is Hertz and three hotels

in the destination city are Hilton, Sheraton and Ramada. The travel agent can then order a ticket from either Northwest or United airlines. Similarly, the agent can reserve a room for a customer at any of the three hotels. Based on these observations, the travel agent may construct a global transaction for this application 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;

In this example, t_1 and t_2 are two alternative subtransactions for ordering a ticket. In this case, t_2 will be executed when subtransaction t_1 fails to achieve its objective. Similarly, t_4, t_5 and t_6 are alternative subtransactions for reserving a room. Usually, a preference order for a set of alternatives will be given by the user, and the system should execute the alternative subtransactions according to the specified order.

An individual subtransaction may fail to achieve its objective either due to unavailability of a local site, communication failure, etc. (physical failure), or because of the checks embedded in the transaction code (logical failure). However, if a functionally equivalent alternative transaction is specified, the global transaction can execute it to achieve its (partial) objective, and be able to continue. In this sense, the global transaction is fault-tolerant and, therefore, can survive a local failure and achieve its global objectives in a multidatabase system, even if the availability of the local database systems is quite low.

2.2 Mixed Transactions

The fundamental properties of a transaction are *atomicity*, *isolation* and *durability*. These properties are important for maintaining the data consistency in many real world applications. However, when applied to the

multidatabase environment, these properties may become too restrictive. As we have discussed in the previous section, global transactions in a multidatabase environment are potentially long lived, which may cause serious performance and throughput problems. It has been argued that the presence of long lived transactions may significantly increase the possibility of deadlock [Gra81]. In addition, a long lived global transaction may block the execution of many high-priority short local transactions by holding the resources which are required by these local transactions.

To solve this problem, the granularity of isolation of the global transaction has to be reduced. Gray [Gra81] proposed to associate with each subtransaction a *compensating subtransaction* which can semantically "undo" the effects of a committed subtransaction, if required. This concept allows the global transaction to reveal its (partial) result to other transactions before it commits. By doing so, the isolation granularity of the global transaction is reduced to the subtransaction level instead of the global transaction level. A global transaction consisting only of subtransactions which can be compensated is called a *saga* [GS87].

However, in the real world, not all subtransactions can be compensated. For example, subtransactions that are accompanied by real actions are typically non-compensatable. To address the fact that some of the subtransactions may be compensatable, we introduce in our model the concept of *mixed transactions*. A global transaction is *mixed* if some of its subtransactions are compensatable and some are not. In a mixed transaction, the subtransactions which are compensatable may be allowed to commit before the global transaction commits, while the commitment of the non-compensatable subtransactions must wait for a global decision. When a decision is reached to abort a mixed transaction, the subtransactions in progress and the non-compensatable subtransactions waiting for a global decision are aborted, while the committed compensatable subtransactions are compensated. In this sense, mixed transactions are different from the *s-transactions* [EVT88] or the *sagas* [GS87] which allow only compensatable subtransactions.

Hence, mixed transactions fill the spectrum from sagas, (assuming the compensability of all subtransactions) to to traditional distributed transactions (assuming that subtransactions are non-compensatable). Mixed transactions are more flexible because they allow compensatable and non-compensatable subtransactions to coexist within a single global transaction.

2.3 Temporal Aspects of Transaction Processing

Unlike traditional distributed database systems, local database systems in a multidatabase environment are usually autonomous in deciding when to execute a subtransaction. Frequently, it is not realistic to assume that all local database systems are operational at the same time when the global transaction is submitted [WQ87]. Consider a bank transaction which involves a bank in the USA and one in Japan. It is quite possible that because of the time difference the subtransactions may not be executed at the same time. In order to execute a global transaction successfully, we may need to know when a specific subtransaction can be executed at the designated local database system. Furthermore, even if all local database systems are available at the same time, we may still prefer to execute different subtransactions at different times. For example, consider a customer who wants to reserve a car and to order a flight ticket for a vacation next week. He may want to rent a car today, while the good selection is still available, and wait to order the flight ticket until two days later, when a special discount price comes in effect.

To specify the execution time of a subtransaction, we associate a *temporal predicate* with each subtransaction. This temporal predicate indicates when the subtransaction should be executed. A subtransaction can be executed only when its temporal predicate is *true*. The temporal predicate has the following format:

temporal-operator time-spec

The time-spec has the following format:

hh:mm:MM:dd:yy

In the above definition, hh stands for hour; mm stands for minute; MM stands for month; dd stands for day; and yy stands for year. A wild card "*" can be used in any of these fields to denote a "don't care" condition. The temporal operators and their meanings are shown in the following table.

operator	use	meaning
between	*between* (08:*:*:*:*, 17:*:*:*:*)	between 8am and 5pm
after	*after* (14:*:*:*:*)	after 2pm
before	*before* (*:*:01:15:90)	before January 15th 1990

Another temporal aspect of multidatabase transaction is the *transaction completion time*. Transaction mangement based on serializability, typically does not take into account the timing characteristics of transaction execution. The only problem addressed by the serializability is the correctness of interleaved executions of multiple transactions. The question of whether the transaction has accomplished its objectives "in time" is frequently ignored. In contrast, *real time database systems* attempt to schedule the execution of transactions to meet their external real-time constraints [AG88b]. Similarly, the concept of a *value date* has been introduced to indicate the fact that a certain data item in a database can be safely accessed only after a specified point in time has been reached [LT88]. We will consider here incorporating the concept of the completion value of a transaction into transaction management. The completion value reflects the fact that some transactions may have associated with them certain utility of their completion, as a function of time. This reflects the fact that the utility of the completion of a transaction may, in general, change with time. This problem is similar to the real time constraint in real-time databases, although the time constraints in the multidatabase environment are usually less stringent. As an example consider a transaction "Sell 500 stock of XYZZY Co. on the NYSE", assuming that it is Friday and the price of stock is going down. To model this phenomenon, we adopt the *value function* [AG88a] to model the usefulness of a global transaction. A value function is a function of the global transaction execution time. A typical value function is shown in Figure 1.

In Figure 1, we assume that the origin of the time axis is the time when the global transaction is submitted. t_0 is the time that after which the completion of the global transaction has no value. As far as the scheduling of a global transaction is concerned, when the execution time of the global transaction reaches t_0, the execution should be aborted. With the value function, it is possible to formulate the inter-transaction scheduling as an optimization problem. In this case, the objective of a scheduling policy is to maximize the total value of all global transactions, subject to their precedence constraints. The problem of optimization will be left outside of the scope of this paper.

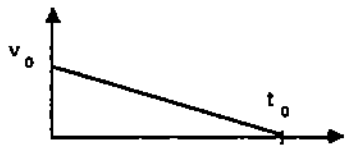

Figure 1: A value function

3 Transaction Model

In this section, we will present the new transaction model. We will first give some preliminary definitions, and then formally define the model.

3.1 Preliminary Definitions

To specify a global transaction in the new model, we need to specify the *execution dependency* among the subtransactions of a global transaction. Execution dependency is a relationship among subtransactions of a global transaction which determines the legal execution order of the subtransactions. In order to define the general execution dependency among subtransactions, we define two basic dependencies. The first is the *positive dependency*. A positive dependency between subtransaction t_1 and t_2 exists if subtransaction t_1 can not be executed until subtransaction t_2 succeeds. This occurs, for example, if subtransaction t_1 has to wait for results from subtransaction t_2 [ED89] before it can start. The second basic dependency is called the *negative dependency*, which is used to specify the alternative subtransactions. Subtransaction t_1 *negatively* depends on t_2 if t_1 has to wait until t_2 has been executed and failed before it can start. This happens when t_1 and t_2 implement the same task in a global transaction and t_2 is preferred to t_1. To facilitate the specification of the execution dependency, we define a *transaction execution state* as follows:

Definition 1 *For a global transaction T with m subtransactions, the* transaction execution state *x is an m-tuple $(x_1, x_2, ..., x_m)$ where*

$$x_i = \begin{cases} N & \textit{if subtransaction } t_i \textit{ has not been submitted for execution;} \\ E & \textit{if } t_i \textit{ is currently being executed;} \\ S & \textit{if } t_i \textit{ has successfully completed;} \\ F & \textit{if } t_i \textit{ has failed or completed without acheiveing its objective;} \end{cases}$$

The transaction execution state is used to keep track of the execution of the subtransactions. It is also used to determine if a global transaction has achieved its objectives. All x_i's are initialized to N when the global transaction starts its execution. The value of x_i is set to E when t_i is submitted for execution to its local database system. When a subtransaction t_i completes the corresponding execution state x_i is set to S if the subtransaction has achieved its objective, and to F, otherwise. The execution state, x, changes as the subtransactions are executed. The set of all possible execution states is denoted by X.

At a certain point of execution, the objectives of the global transaction may be achieved. In this case, the global transaction is considered to be successfully completed and can be committed. An execution state in which a global transaction achieves its objectives is called an *acceptable state*. Frequently, there is more than one acceptable state for a global transaction. The set of all acceptable states of a global transaction is denoted by A.

Definition 2 *The* acceptable state set, A, *of a global transaction T is a subset of X, where*

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

In order to express execution dependencies, we associate with each subtransaction t_i, a *precedence predicate*, pp_i. The precedence predicate is a boolean function defined on the transaction execution state, as follows:

Definition 3 *A precedence predicate pp_i for a subtransaction t_i is a predicate defined on X, where*

$$pp_i : X \rightarrow \{1, 0\}$$

To indicate that t_j positively depends on t_i, we formulate the precedence predicate $pp_j := (x_i = S)$. We use the precedence predicate $pp_j := (x_i = F)$ to denote that t_j negatively depends on t_i. Having defined the basic dependencies, we can express any execution dependency in terms of boolean combination of the basic dependencies. A predicate pp_i is defined on the transaction execution state and is used to determine whether the corresponding subtransaction can be submitted for execution at the current time. The value of the precedence predicate changes as the global transaction is executed.

3.2 The Extended Transactions

To capture all the previously discussed semantics of a multidatabase transaction, we use additional primitives in the definition of a transaction. A global transaction in our model is formally defined as follows:

Definition 4 *A* global transaction *T is a 6-tuple (ST, O, PP, TP, A, V) where*

- *ST is subtransaction set of T*
- *O is the partial order on ST*
- *PP is the set of all precedence predicates of ST*
- *TP is the set of all temporal predicates of ST*
- *A is the set of all acceptable states of T*
- *V is the value function of T*

In order to specify a global transaction, we have to specify, at the subtransaction level, the set of subtransactions. Then with every subtransaction we specify its subtransaction type as follows:

Subtransaction type:

- C - if the subtransaction is compensatable

- NC - if the subtransaction is non-compensatable

We also specify the precedence predicate and the temporal predicate of the subtransaction. At the global transaction level, we specify the partial order O, the set of acceptable states A and the value function.

We illustrate the above definition using as an example the travel agent transaction, introduced in the previous section.

Example 2: Consider the travel agent transaction introduced in example 1. In addition, we assume the following: (1) the subtransactions for ordering tickets are non-compensatable; (2) ticket ordering subtransactions must run within business hours from 8am to 5pm, other subtransactions do not have time constraints; (3) the global transaction has to complete within one day in order to be useful, and within the time limit, the utility of the transaction completion depends on the completion time. This transaction can be formally specified as follows:

$$\begin{aligned} \mathrm{ST} &= \{t_1(NC), t_2(NC), t_3(C), t_4(C), t_5(C), t_6(C)\} \\ \mathrm{O} &: t_1 \prec t_3,\ t_2 \prec t_3,\ t_3 \prec t_4,\ t_3 \prec t_5,\ t_3 \prec t_6 \end{aligned}$$

$$PP : \begin{cases} pp_1 := true \\ pp_2 := (x_1 = F) \\ pp_3 := (x_1 = S) \vee (x_2 = S) \\ pp_4 := (x_3 = S) \wedge (x_5 = F) \\ pp_5 := (x_3 = S) \\ pp_6 := (x_3 = S) \wedge (x_4 = F) \wedge (x_5 = F) \end{cases}$$

$$TP : \begin{cases} tp_1 = between(08:*:*:*:*, 17:*:*:*:*:*) \\ tp_2 = between(08:*:*:*:*, 17:*:*:*:*:*) \\ tp_3 = * \\ tp_4 = * \\ tp_5 = * \\ tp_6 = * \end{cases}$$

$$\begin{aligned} \mathrm{A} = \{ &(S,N,S,S,N,N), \\ &(S,N,S,F,S,N), \\ &(S,N,S,F,F,S), \end{aligned}$$

$$(F,S,S,S,N,N),$$
$$(F,S,S,F,S,N),$$
$$(F,S,S,F,F,S)\ \}$$

$$v(t) = \begin{cases} 1 & \text{if } t <= 12\ hours \\ 0.5 & \text{if } 12 < t <= 24\ hours \\ 0 & \text{otherwise} \end{cases}$$

The execution of a global transaction has to abide by a set of execution rules. Before we formulate the set of execution rules for the extended transactions, we will introduce an additional definition.

Definition 5 *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 all predecessors of t_i, i.e.*

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

For a given execution state x, we define a subtransaction t_i as executable if

1. t_i has not been submitted for execution;
2. $\forall\ t_k \in pred(t_i)$, either t_k has been executed or the pp_k is false; and
3. both the pp_i (the precedence predicate of t_i) and the $tp_i(t)$ (the temporal predicate of t_i) are true.

We can now formulate the execution rules as follows:

1. Start from the initial execution state of the global transaction;
2. Schedule the executable subtransactions for execution until the termination condition has been met;
3. When a subtransaction t_i is submitted, x_i is set to E. When the execution of a subtransaction is completed, set x_i to S, if the objective of the subtransaction has been achieved and to F, otherwise.

4. The execution of a global transaction terminates when any of the following conditions occurs:

   - the current execution state is acceptable,
   - none of the subtransactions is executable and no subtransaction is currently executing,
   - time t_0 of the value function is reached (if applicable).

According to the above execution rules, concurrent execution of subtransactions is allowed if they are executable at the same time. When the result of the execution is known, we modify the transaction execution accordingly. After the completion of a subtransaction, we check if the termination condition is satisfied. If the termination condition is not satisfied, we continue scheduling the executable subtransactions. If the global transaction terminates and an acceptable state has been reached, we can commit the global transaction; otherwise, it must be aborted. To commit a global transaction, we send a "commit" message to all non-compensatable subtransactions which are waiting in their "prepared to commit" states (the compensatable subtransactions may have been committed earlier). If the global transaction terminates without reaching an acceptable state, the global transaction must be aborted. To abort a global transaction, we send an "abort" message to all subtransactions which are waiting in a prepared state, and then issue compensating subtransactions for those compensatable subtransactions that are committed.

4 Execution of the Global Transaction

In this section we will discuss the execution of extended global transactions specified using the extended transaction model. Since our discussion will be based on the Predicate Petri Nets (PPN) formalism, we will review briefly the basic concepts of Predicate Petri Nets. Then we will show how the problem of scheduling extended transactions can be mapped into an appropriate PPN. Finally, we will show how the execution of the multidatabase transactions can be controlled using this mechanism.

4.1 The Predicate Petri Nets

To control the execution of global transactions, we will use the *Predicate Petri Nets* [Kel76], [LM86], [Gen87]. The PPN control structure can identify, at any execution step of the global transaction, the set of related (and possibly executable) subtransactions. We assume that the reader is familiar with the basic Petri Nets theory [Pet81].

To represent a global transaction we associate with each transition of a Predicate Petri Net a subtransaction and its corresponding precedence predicate and temporal predicate. The partial order O of the global transaction is reflected in a PPN graph.

For a given global transaction, a PPN consists of:

1. a *bipartite graph* G = (P,T,F) where P and T are called *places* and *transitions* respectively, and F is a set of directed arcs, each connecting a place p $\in$ P to a transition tr $\in$ T or vice versa. Places are represented by circles while transitions are represented as a bars. For each transition, those places that have edges directed into the transition are called the *input places* of the transition, and the places that have edges directed out of this transition are called the *output places* of the transition. A place can hold *token*. A token is represented as a dot.

2. A function *PP* (stands for Precedence Predicate), which maps the set of transitions to the set of precedence predicates.

3. A function *TP* (stands for Temporal Predicate), which maps the set of transitions to the set of temporal predicates.

4. A function K, which associates each transition with a subtransaction of the global transaction.

The PPN graph can be derived from the precedence predicate and the partial order O of the global transaction. The dynamic aspect of a PPN corresponds to the execution of the corresponding global transaction. A *marking* M is a distribution of tokens over the places of a PPN which represents the current status of the global transaction execution. A PPN models the execution of a global transaction by *firing* transitions in accordance with the conditions specified by the predicates associated with each transition.

As usual, we define a transition to be *enabled* if all of its input places contain at least one token. We then define a transition to be *executable* [1] if it is enabled and both the associated precedence predicate and temporal predicate are *true*. Finally, we define a transition to be *firable* if it is executable and the associated subtransaction has been executed successfully. In a marking M, the set of all enabled transitions is called the *enabled set*. Similarly, the set of all executable transitions is called the *executable set*. We attempt to fire a transition by submitting its associated subtransaction for execution. If the execution is successful, then we fire the transition; otherwise, we update the corresponding execution state variable. To be more specific, when transition tr_i is fired, we perform the following actions:

- Update the execution state variable x_i by setting it to S.
- calculate the new marking by taking one *token* from each of the input places of the transition tr_i, and put one token into each of tr_i's output places.

4.2 Constructing the Predicate Petri Nets

In constructing a PPN graph for a global transaction, we have to use the information of partial order O and the positive dependency of the global transaction. The negative dependency is not considered in constructing the graph since it will be taken care of when the corresponding precedence predicate is evaluated. In the construction process, we first filter all negative dependencies from every precedence predicate. We then transform the results into *conjunctive normal forms* [2].

Afterwards, we apply the following procedure to construct the PPN graph:

1. construct a graph as shown in Figure 2 for every subtransaction t_j which has an empty predecessor set.

[1] This implies that its associated subtransaction is executable.

[2] As shown in the *propositional calculus* [LP81], every predicate has at least one conjunctive normal form, and there is an algorithm that transforms any predicate into its corresponding conjunctive normal form.

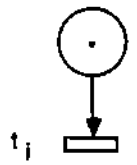

Figure 2: PPN graph for subtransactions with no predecessor

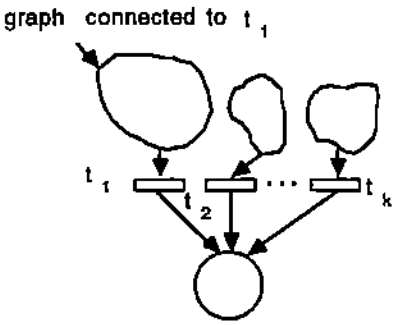

Figure 3: PPN graph for a disjunction form

2. repeat the following two steps until all transitions are constructed.

   For any subtransaction t_i which has all of its predecessors been constructed, do the following:

   (a) For every disjunction form $(x_1 = S) \vee (x_2 = S) \vee \cdots (x_k = S)$, construct a graph as shown in Figure 3.

   (b) Connect the resulting graph of (a) to transition t_i.

3. connect each transition which does not have an output place to an output place; then terminate this procedure.

To illustrate this procedure, we will apply it to construct the PPN graph for the global transaction in example 2. Figure 4 shows the PPN graph after step 3. Now consider the construction of the graph for t_3. Since there is only one disjunction form $(x_1 = S) \vee (x_2 = S)$, we construct a graph as in Figure 5. The final graph becomes Figure 6 after we complete this procedure.

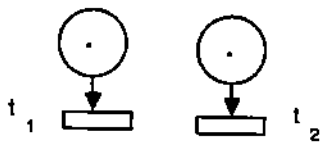

Figure 4: PPN graph after step 3

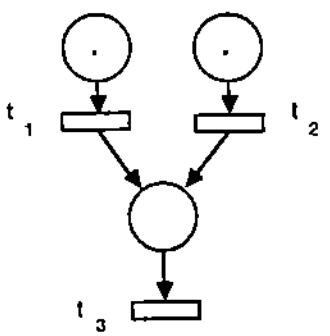

Figure 5: PPN graph for t_3

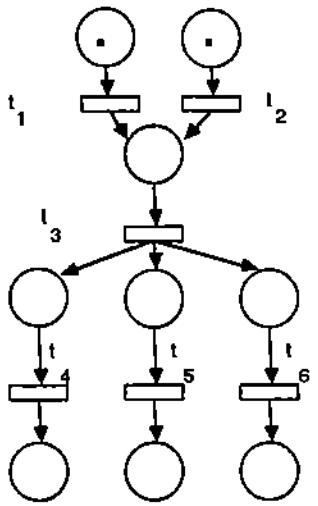

Figure 6: PPN graph for the travel agent transaction

4.3 The PPN Execution Control Algorithm

Global transaction execution must satisfy the partial order O, the precedence predicate and the temporal predicate. By capturing all this information, the PPN significantly simplifies the execution control of the global transaction. To maintain the utility of the completion of a global transaction, we use a **timeout** mechanism to abort the execution of a global transaction when the value function becomes 0 and the global transaction is still in execution. Before invoking the algorithm, we calculate t_0 (the time at which the value function becomes 0). We then use t_0 as a parameter for the **timeout** mechanism.

The global transaction scheduling problem is *event driven* in the sense that the scheduling activity is invoked when a subtransaction completes (either successfully or unsuccessfully). We use a *queue* Q to buffer the responses (i.e. events) from the local database systems. The algorithm will enter a response in Q when it receives the response from a local database system.

The algorithm is shown in figure 7.

```
procedure evaluate_PPN(NT, ST, PP, TP, n, A, t0)
   Initialize timeout mechanism with timeout interval t0;
 begin
      x ← (N, N, N, ···, N) /* n N's */
      on timeout flex_abort;
      E ← φ;     /* E – enabled set */
      U ← φ;     /* U – executable set */
      G ← φ;     /* G – scheduled set */
      Q ← empty;     /* Q – response Queue */
      compute_enabled_set E from NT;
      compute_executable_set U from the new enabled set E;
   repeat
      G+ ← U - G;
      For each tri ∈ G+ do
        begin
          submit tri to the local database system;
          G ← G ∪ { tri};
          xi ← E
        end;
      on receiving response enqueue the response in Q;
      while (Q = empty) do
        begin
          if (check_terminate) then
            begin
              flex_abort;
              exit
            end
        end;
      RESP← dequeue(Q);
     /*assume that RESP is from ti;*/
      G ← G - { tri };
      if (RESP = SUCCESS)
      then
        begin
          xi ← S;
          fire(tri);
          compute_enabled_set E
        end
      else
        xi ← F
      endif;
      compute_executable_set U;
   until (check_terminate);
     if x ∈ A then flex_commit
     else flex_abort;
 end.
```

Figure 7: The execution control algorithm

In the algorithm, $\mathcal{E}$ is the current enabled set; $\mathcal{U}$ is the current executable set derived from $\mathcal{E}$; $\mathcal{G}$ is the scheduled set which contains the transitions whose corresponding subtransactions have been submitted for execution and whose results are still not known. The algorithm starts from the initial transaction execution state (all state variables are initialized to N). When started, the algorithm calculates, from the initial marking, the enabled set $\mathcal{E}$; and uses it to calculate $\mathcal{U}$. All subtransactions whose corresponding transitions are in $\mathcal{U}$ are concurrently submitted to the local database systems for execution.

Whenever a subtransaction completes, successfully or unsuccessfully, a new executable set $\mathcal{U}$ is calculated. In the new executable set, some of the transitions have been submitted (for those contained in $\mathcal{G}$) while some are not. The transitions which are executable and not yet submitted are contained in $\mathcal{G}^+$ which is the difference of $\mathcal{U}$ and $\mathcal{G}$. Only subtransactions whose corresponding transitions are in $\mathcal{G}^+$ need to be submitted each time when a new executable set is derived. In order not to overlook responses from the local database systems, the responses are first buffered in Q. If Q is empty, it is possible that the execution of the global transaction has failed. In this case, the termination condition is checked. If Q is not empty, we can dequeue a response from Q. After dequeue a response from Q, if the response reports that subtransaction t_i is executed successfully, x_i, the corresponding execution state variable of t_i is set to S; otherwise, it is set to F. After this, the algorithm evaluates the enabled set $\mathcal{E}$ (in the first case) and the executable set $\mathcal{U}$ and then continues scheduling the global transaction. When the execution terminates, if the final execution state x is acceptable (i.e. $x \in A$), the global transaction is committed; otherwise, it is aborted.

The execution of a global transaction terminates in either of the following two cases:

1. x is acceptable (x $\in A$). In this case, the global transaction is successfully executed;
2. there is no executable transition and no subtransition is waiting for its temporal predicate to become *true*.

The function for the termination detection is shown in figure 8. To commit

```
Function check_terminate(x, A, NT, E, U, TP)
    W:Waiting set;
  begin
    check_terminate ← false   /* initialized to false */
    if (x ∈ A)
    then
      check_terminate ← true
    else
      begin
        W ← φ;
        For each tr_i ∈ E and TP(tr_i) = false
        and operator of TP(tr_i) ≠ before
          W ← W ∪ { tr_i };
        if (U = φ and W = φ)
        then
          check_terminate ← true
      end
  end
```

Figure 8: The function for checking termination condition

a global transaction, for each non-compensatable subtransaction t_i whose corresponding execution state variable x_i is S, send a "commit" message to its local database system; for each non-compensatable subtransaction t_j (if any) in $\mathcal{G}$, send an "abort" message to its local database system; and then *compensate* each compensatable subtransaction in $\mathcal{G}$.

To abort a global transaction, each subtransaction t_i, whose corresponding execution state variable x_i is S, has to be aborted or compensated depending on its type.

5 Conclusion

The need for local autonomy in a multidatabase system makes the traditional models of a transaction obsolete. More flexible and powerful models are needed. The model presented in this paper, addresses the need for multiple execution alternatives, very frequent in practical applications. It also provides for mixed transactions consisting of compensatable and non-compensatable subtransactions. Mixed transactions generaliz e the concepts of nested transactions and sagas. It also deals with time and utility functions, providing new possibilities for transaction scheduling. We have ex-

plained the rationals of our model, and have formalized it. We have also proposed an implementation using the Predicate Petri Nets.

The InterBase project is currently investigating these and other related issues [Leu90]. The need for a transaction specification language based on our model has become apparent and is the subject of attention at the Laboratory. Predicate Petri Nets appear also useful as an analysis tool for global transactions. Additional work is needed to design schemes for concurrency control and recovery in the new environment. The results are of great importance, as multidatabase systems become more widely used and needed. Our results may be also applicable in the CAD/CAM, CASE and SDE database areas, where similar work on extending the conventional notions of transaction processing and correctness criteria is also being carried out.

References

[AG88a] R. Abbott and H. Garcia-Molina. Scheduling real-time transaction. *SIGMOD RECORD*, 17(1), March 1988.

[AG88b] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions: a performance evaluation. In *Proceedings of the fourteenth international conference on very large data bases*, pages 1–12, August 1988.

[AGS87] 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.

[DEK90] W. Du, A. Elmagarmid, and W. Kim. Effects of local autonomy on heterogeneous distributed database systems. Technical Report ACT-OODS-EI-059-90, MCC, 1990.

[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.

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

[EVT88] F. Eliassen, J. Veijalainen, and H. Tirri. Aspects of transaction modelling for interoperable information systems. In *Interim Report of the COST 11ter Project*, pages 39–55, 1988.

[Gen87] H. J. Genrich. *Predicate / Transition Nets.* Number 2. Springer-Verlag, 1987.

[GK88] H. Garcia-Molina and B. Kogan. Node autonomy in distributed systems. In *Proc. Int'l Conf. on Data Engineering*, pages 158–166, 1988.

[GS87] H. Garcia-Molina and K. Salem. Sagas. In *Proceedings of the ACM Conference on Management of Data*, pages 249–259, May 1987.

[GP86] V.D. Gligor and R. Popescu-Zeletin. Transaction management in distributed heterogeneous database management systems. *Inform. Systems*, 11(4):287–297, 1986.

[Gra81] J. 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.

[Kel76] R. M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7), July 1976.

[LE90] Y. Leu and A. Elmagarmid. A hierarchical approach to concurrency control for multidatabases. In *Second International Symposium on Databases in Parallel and Distributed Systems*, July 1990.

[LER89] Y. Leu, A. Elmagarmid, and M. Rusinkiewicz. An extended transaction model for multidatabase systems. Technical Report CSD-TR-925, Department of Computer Science, Purdue University, 1989.

[Leu90] Y. Leu. *Transaction Management for Multidatabase Systems.* PhD thesis, Department of Computer Sciences, Purdue University, December 1990, In preparation.

[LM86] C. Lin and D. C. Marinescu. Application of modified predicate transition nets to modeling and simulation of communication protocols. Technical Report CSD-TR-599, Purdue University, May 1986.

[LP81] H.R. Lewis and C.H. Papadimitriou. *Elements of the theory of computation.* Prentice-Hall Book Company, 1981.

[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.

[Pet81] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Englewood Cliffs, 1981.

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

[RELL90] M. Rusinkiewicz, A. Elmagarmid, Y. Leu, and W. Litwin. Extending the transaction model to capture more meaning. In *ACM SIGMOD RECORD*, volume 19, 1990.

[ROEL90] M. Rusinkiewicz, S. Ostermann, A. Elmagarmid, and K. Loa. The distributed operational language for specifying multi-system applications. In *Proceedings of the 1st International Conference on Systems Integration*, 1990.

[WQ87] G. Wiederhold and X. Qian. Modeling asynchrony in distributed databases. In *Proc. Int'l Conf. On Data Engineering*, 1987.

[WV90] A. Wolski and J. Veijalainen. 2PC agent method: achieving serializability in presence of failures in a heterogeneous multidatabase. In *Proc. PARBASE-90 Conference*, March 1990.