Extending the Transaction Model to Capture more Meaning *

M. E. Rusinkiewicz, A. K. Elmagarmid, Y. Leu and W. Litwin †

Computer Sciences Department
Purdue University
West Lafayette, IN 47907

1 Introduction

Since it has been first introduced, the concept of transaction has become one of the fundamental abstractions used in the design and analysis of information systems that provide concurrent access to share information. The basic idea of transactions is to divide application programs into well defined units that provide semantically correct transitions between consistent states of the information system. The fundamental properties of transactions, as defined by Gray [Gra81], namely atomicity, isolation and durability turned out to be very useful in the modeling of real world applications. In particular, most work on concurrency control, commitment and recovery has been performed using transactions as a basic unit of work.

However, as the new computing environments encompassing heterogeneous and autonomous information systems begun to emerge, it became increasingly clear that the limitations of the traditional transaction concept begun to outweigh its virtues. We will examine the problems of transaction management in the context of multidatabase systems where transactions must access multiple autonomous (and frequently heterogeneous) database systems, in order to accomplish their objectives. The main source of difficulty in applying the traditional transaction management techniques to these new environments is the requirement of *local autonomy* for the individual systems participating in the transactions. Another problem is the potential for *long lived* transactions that make many basic techniques developed in the context of centralized databases (strict locking, two-phase commitment, etc), totally inapplicable in these environments.

There have been several attempts to overcome the inadequacy of the traditional transaction concepts and the limitations of serializability as a basic correctness criterion for the concurrent execution of transactions spanning across multiple and autonomous systems. Gray proposed to divide a global transaction into relatively independent subtransactions. He proposed to associate with each subtransaction a *compensating transaction* that can "undo" the effects of a committed subtransaction if required by the global transaction. This idea was further extended by Garcia-Molina who developed a notion of *sagas* [GMS87] which reject serializability as a basic correctness criterion. Another idea that has received attention as a means for overcoming the above mentioned difficulties is the concept of nested transactions [Mos81]. However the notion of nested transactions as proposed by Moss, does not address at all the autonomy of local systems.

In this paper we outline an extended transaction model, which we believe, is much more suitable for computing environments consisting of autonomous systems. The proposed model allows us to utilize knowledge of the semantics of the application that is to be modeled by a transaction. The model allows composition of flexible transactions consisting of mutually dependent subtransactions. The execution of these subtransactions may depend on the success of previous subtransactions, and alternative sources of information may be specified. This approach requires redefinition of the notion of successful execution transactions, their scheduling and commitment.

Some of the ideas incorporated in the proposed extended transaction model have been introduced earlier. For example, the concepts of functional equivalence of subtransactions and conditional execution of subtransactions have been formalized in [LER89]. Similarly, the idea of using time in specification of transaction execution can be found in [WQ87] and

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

†On leave at Department of Computer Science, Stanford University from INRA.

[LT88]. Here, we attempt to combine these and other ideas to provide a unified framework for the discussion of multidatabase transactions.

The rest of this paper is organized as follows: In the next section we introduce the extended transaction model and its basic components: execution dependencies, acceptable state set and the completion value as a function of time. Then we will discuss the basic problems of transaction scheduling as an optimization problem. In the final section, we will discuss how the new model affects the basic notions of transaction commitment, concurrency control and recovery.

2 Transaction Model

Definition 1 *A transaction is a 4-tuple (S, D, A, V) where:*

- $S = \{s_1, s_2, ..., s_n\}$ *is a set of subtransactions,*
- *D is a set of predicates describing execution dependencies among subtransactions,*
- *A is a set of acceptable execution states, and*
- *V is a function describing the value of completion of the transaction in time*

Below, we will explain the basic components of a transaction.

Subtransaction Set

With every subtransaction from the set S we associate the following descriptors:

Transaction type:

- C - transaction is compensatable (i.e. can be undone)
- NC - transaction performs real actions (e.g. fires a missile)
- R - transaction is repeatable (i.e. can be executed in accordance with the "at least once" semantics)
- NR - transaction is nonrepeatable (i.e. must be executed in accordance with the "at most once" semantics)

Commitment date (t_c): Date after which the transaction is considered to be implicitly committed (optional) [1]

Expiration date (t_e) : Date after which the transaction is considered to be implicitly aborted (optional).

Execution Dependencies

D - is a set of predicates that define the dependencies among subtransactions of a (global) transaction. The following types of dependencies can be defined [LER89]:

- *Positive execution dependency* determines a (partial) ordering of subtransactions. (Value dependency, as defined in [ED89] constitutes a special case of positive dependency.) We say that a subtransaction T_i is positively dependent on subtransaction T_j, if T_i cannot be executed until T_j completes successfully.
- *Negative execution dependency* exists between subtransactions T_i and T_j if T_i cannot be executed until T_j has been scheduled and failed [2].
- *Alternative dependency* is defined by providing equivalence classes for transactions that are functionally equivalent. With every class we may associate a preference ordering relation defined over all subtransactions in the class. Two subtransactions are *functionally equivalent* if they accomplish the same function in achieving the transaction's objective. If there are several functionally equivalent transactions, typically only one of them will need to be executed.
- *Compensating execution dependency* exists between subtransactions T_i and T_j if T_j "undoes" the effects of T_i when executed after the successful completion of T_i.

A subtransaction may be in one of four states: {not executed, executing, executed successfully, executed and failed}. A *transaction execution state* is a vector of execution states for all its subtransactions. The transaction execution state is modified, whenever a scheduled subtransaction completes (commits or aborts).

[1] The idea of value date can be traced to the early paper by Wiederhold et al. [WQ87]; it has been formally defined by Litwin [LT88].

[2] We say that a subtransaction fails, if it is executed but does not achieve its objectives (logical failure) or if it can not be successfully completed, because of a physical failure.

At any point of time there exists a set ST of *schedulable transactions*. A subtransaction T_i is *schedulable* if the current execution state of the transaction satisfies all prerequisite dependencies (positive, negative and alternative) of the subtransaction T_i.

Acceptable Execution State

The execution state of a transaction is acceptable if the objectives of the transaction have been accomplished. The set of all acceptable states of a transaction constitutes the *acceptable state set* which is denoted by A. Once a transaction execution reaches an acceptable state, no new subtransactions are scheduled.

The extended transactions are submitted to the scheduler that attempts to execute them by scheduling subtransactions in accordance with their execution dependencies. A transaction completes successfully when it reaches an acceptable state; it must be aborted when its execution state is not acceptable and there is no schedulable subtransaction. The problems of committing or aborting a global transaction will be discussed further in the next section.

Completion Value of Transaction

With every transaction we associate its (relative) value as a function of time. The value function, V, reflects the fact that the completion of a transaction may represent a different value (utility) to the user depending on the completion time. Typically, the value will decrease with time, although the changes may be cyclical or multimodal. For example, the function may indicate that we may attempt to execute a transaction only during particular hours, say 9-5 in Japan.

Examples:

We will illustrate the concepts introduced above with some intuitive examples using banking and travel agent examples.

- A deposit(100, Acct_No) may be **compensatable**. A withdraw(100, Acct_No) may be not compensatable, if it is accompanied by a real action.
- A read(fileid, Key_value) from an indexed file is **repeatable**. A read(fileid, next) from a sequential file is **not repeatable**.
- **Expiration date**: Let reservation transaction abort automatically after 24 hours.
- **Value date**: Let the check withdrawal commit automatically on December 2, 1989.
- **Positive dependency**: Make hotel reservation in a destination city, if you made a flight reservation.
- **Negative dependency**: Get reservations on American, if your attempt to get reservations on Continental failed.
- **Compensating transaction**: Cancel car rental in NYC for December 24th, 1989.
- **Alternative transactions**: Rent a room in Chicago on December 11, 1989, at Hilton, Sheraton or Ramada, in this order of preference.
- **Execution state, acceptable state**: Consider the following transaction:

  make fight reservation {American, Delta, UsAir}
  rent a car {Avis, National, Dollar}
  make hotel reservation {Hilton, Sheraton}.

  Let us consider the following execution state:
  (failed, successful, not executed,
  successful, not executed, not executed,
  failed, not executed)

  In this case, the set of schedulable subtransactions consists of one transaction: make hotel reservation (Sheraton) If this subtransaction executes successfully, then the global transaction reaches an acceptable state (please notice that this occurs despite the fact, that some of the subtransactions have failed and some were never executed). Otherwise the global transaction fails, since the set of schedulable transactions becomes empty.
- **Value function**: Buy a ticket to Chicago for departure on December 21st, before closing today (at a special fare X), or before next Monday (at a normal fare Y), or before the departure date (at a ridiculously inflated "full coach" fare).

3 Transaction Processing

With the addition of the time value function the problem of scheduling multidatabase transactions can be formulated as an optimization problem.

The objective is to maximize the total value of committed transactions within a particular period of

time, while observing the intra- and inter-transaction dependencies. The intra-transaction dependencies are specified explicitly in each (global) transaction. The inter-transaction dependencies are determined by the correctness criteria used by the multidatabase systems.

For example, if we assume that no dependencies exist between local database systems involved in a multidatabase transaction, *quasi serializability* [DE89] can be used as an inter-transaction correctness criterion. In this case, local executions are serializable and there is an order of global transactions. Another possibility is to use *value-date safe* executions as defined by the value date paradigm [LT88]. Under this proposal a data item can be accessed "safely", only if the current time is greater than the value date associated with this data item.

Several possible transaction scheduling algorithms can be envisioned here.

A greedy algorithm will attempt to schedule all subtransactions immediately, attempting to maximize the total value of scheduled transactions. This may lead to a situation where the newly arriving transactions of higher value would have to wait for the completion of low value transactions in progress.

A Maitre d'Hotel algorithm will leave certain processing potential (say 20%) usually unused at each point of time, so that when a new transaction of a very high value arrives it can be accommodated immediately, possibly by delaying the processing of other transactions, with lower completion values. [3]

In general, the addition of the value function for completion may alleviate some problems with the pure "value date" approach, that are shared with other algorithms based on deadline scheduling.

The notion of commitment would have to be reexamined in the context of the extended transaction model. We have already mentioned that a transaction may complete successfully, even if some of its subtransactions failed or were not executed. We have assumed that subtransactions are scheduled in accordance with their execution dependencies. When an acceptable state is reached we may need to perform the following actions:

- All subtransactions needed to achieve the objectives of the global transaction are committed.
- All subtransactions that are currently executing are aborted, and
- Compensating subtransactions are scheduled for all those completed subtransactions that violate the alternative dependencies.

For example, let us consider our travel agent transaction. Let us suppose that the three airline reservation subtransactions were scheduled concurrently and the last two of them have completed successfully. Then, one of the "rent a car" subtransactions has been successful. Let us assume that currently the two hotel reservation subtransactions are in progress. As soon as the first completes successfully, the objective of the global transaction is achieved. The global transaction can be now committed as follows: The second hotel subtransaction in progress, is aborted, the second airline reservation subtransaction must be compensated, and the remaining completed subtransactions are committed.

Of course, the commitment process as described above, may not always be applicable. In such cases, we can take advantage of the mechanisms for implicit commitment or expiration of subtransactions that are provided in the model. By choosing the appropriate commitment and expiration dates for the subtransaction, we may *implicitly commit or abort* extended transactions.

4 Conclusions

In this paper we have proposed a new transaction model which allows specification of complex transactions, that are capable of capturing more semantics of the applications. This model is, in our opinion, much more suitable for computing environments consisting of multiple autonomous and heterogeneous systems, in which a given objective can be frequently accomplished in more than one way. The most important elements of this proposal are:

- Clear recognition of the fact that global transactions accessing multiple autonomous systems can not be efficiently processed according to the traditional transaction model and using serializability as a correctness criterion. Hence, a global transaction must be treated as a set of subtransactions with complex intra-transaction dependencies, whose execution may not always be completely controlled by a centralized scheduler.

[3] This algorithm is frequently used by good restaurants and by state airlines in some countries that keep places on their planes open for high officials who may arrive at the last minute.

- A transaction may complete successfully even if some of its subtransactions were never executed or failed. In this case some of the scheduled subtransactions may need to be aborted and some of the committed subtransactions may have to be compensated (if applicable).
- Since the same function can be frequently accomplished in more than one system, we need to have a mechanism to define the functional equivalence of subtransactions and to specify their alternative scheduling.
- Transaction models proposed so far, either completely ignored the completion time of a transaction (serializability), or treated it as a hard deadline that a transaction must meet in order to complete successfully (real time transactions, value dates). Here, we propose a much more realistic paradigm under which a transaction completion has its value as a function of time. Hence, if a transaction deadline cannot be guaranteed we may still want to complete it, recognizing that the utility of such completion may be lower, but not necessarily zero.

Although the ideas presented in this paper are only preliminary, they are feasible and correspond to real life needs, as illustrated by our examples. However, we recognize that a significant amount of research on the properties of the extended transactions and their scheduling would have to be done, before practical transaction processing systems utilizing these ideas can be build.

5 Acknowledgements

Many ideas presented in this paper were clarified during discussions with Professor Gio Wiederhold.

References

[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.

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

[GMS87] H. Garcia-Molina and K. Salem. Sagas. In *Proceeding of the ACM Conference on Management of Data*, pages 249–259, May 1987.

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

[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.

[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.

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