Modelling another transaction manager using parallel decision processes and a translation into an object-oriented program.

Drs S.C. van der Made-Potuijt
Erasmus University Rotterdam

Dr. L.P.J. Groenewegen
State University Leiden

September 23, 1992

1 Preface

Parallel behaviour can be modelled succesfully by Paradigm, a modelling method using parallel decision processes. Its name is an acronym for PARalellism, its Analysis, Design and Implementation by a General Method. For a comprehensive discussion the reader is referred to [Groe].

In this paper Paradigm will be used to present a model for a transaction manager for a database management system. The purpose of the presented model is to split the tasks of a transaction manager into components in order to support the developping of a program that performs the tasks of these components including their interaction. This model will be translated into an object-oriented program. Such a program is added not to give an efficient program for transaction management, but to show how the model can easily be translated into modules that form together the program. The program can serve as a basis for the implementation of a transaction manager on a system using several processsors, using shared memory, allthough this might not be the most efficient implementation.

- 1984 CR catagories: G.1.0, H.2.4, H.2.7.
- 1980 Mathematics subject classification (1985 revision): 68N15, 68P15.
- Keywords & Phrases: Parallel Algorithms, Transaction Processing.

2 A description of the transaction manager.

A transaction manager can be seen as a part of the interface between the user and the database. The main responsibility of a transaction manager is to support the basic operations of a transaction: Start, Commit and Reject. Therefore, the transaction manager is responsible for concurrency control and recovery. In this paper query optimization is not considered to be part of the task of the transaction manager.

As we are going to describe our transaction manager on a rather detailed level, we have to choose some specific protocols for this manager. Observe that for other protocols the same modelling approach gives analogous results. The transaction manager described in this paper uses the strict two-phase locking protocol [SLR] for concurrency control with the wound-wait protocol [Esea] in order to prevent deadlock. The wound-wait protocol is normally not used in a non-distrubuted database management system. A deadlock-detection algorithm however would give similar results and the wound-wait protocol will also be used in more complicated transction managers to be modelled. If the wound-wait protocol forces a transaction to be made undone and restarted, this will be called a *restart* in this paper.

The immediate update protocol is used with a log, and the logbuffers are assumed to be written to stable storage immediately. It is assumed that no checkpoints are made. If a transaction is executed succesfully it will be called *committed*, otherwise it will be called *rejected*.

As the transaction manager's task with respect to one transaction is usually considered as being complex, the more so it is complex because of the (pseudo) parallel handling of simultaneously offered transactions. In order to simplify the programming of the transaction manager's task, we model the transaction manager by means of three components: the transaction buffer manager (TBM), the scheduler (SCH) and the recovery manager (RM).

To complete the model we will also describe the role of the users. Although many different users could be involved, it may be assumed that all the transactions are submitted to the system at a different moment in time, so it is similar to a situation where all the transactions are submitted by one user. But, of course, the transactions can interfere with each other, as their execution can be overlapping. Therefore concurrency control is necessary. The tasks of the three components we distinguish within the transaction manager are described hereafter, together with the user. We will denote these four components of our model with a capital, indicating the features we describe only apply to these components as relevant for our model.

2.1 The Transaction Buffer Manager.

The two main tasks of the TBM are to initiate the start of a transaction and to terminate a transaction. To start a transaction the TBM writes a *start* record in the log, assigns a unique timestamp to the transaction and submits the transaction to the Scheduler. If the transaction has been succesfull, the Scheduler returns the transaction to the TBM. The TBM writes a *commit* record in the log and notifies the User of the succesfull end of the transaction.

If the transaction has not been succesfull, the TBM hands the transaction over from the SCH to the RM. The RM has to take appropriate action. When the RM is ready, the TBM reacts according to the result. A transaction that can be redone or has to be restarted will be resubmitted to the Scheduler. If the transaction was rejected and had to be undone, a *reject* record is written in the log and the user will be informed of the negative result.

Notice that the TBM hands the transactions over from one component to the other component, e.g. the Scheduler does not hand a transaction over to the Recovery Manager directly, this is done via the TBM.

2.2 The Scheduler.

The Scheduler is responsible for a correct interleaved execution of the transactions, so it is responsible for concurrency control. Furthermore the SCH is responsible for writing the *read* and *write* records in the log. Also it writes a *partial-commit* record in the log after a succesfull end of a transaction, just before the SCH submits the transaction to the TBM.

If due to some error or failure the transaction cannot end succesfully, the TBM hands the transaction over from the SCH to the RM. And in case of a restart the TBM hands the transaction over from the SCH to the RM.

2.3 The Recovery Manager.

From the Scheduler and via the TBM the RM receives transactions that cannot proceed succesfully. We distinguish the following failures:

- A failure that effects only one transaction. We will call this a One Transaction Failure (OTF). This happens because of some unfullfilled condition in the transaction, some erroneous situation in case the transaction has not been tested well enough etc. The Recovery Manager has to undo the results of this particular transaction and then to submit the transaction to the TBM that will hand it over to the User. It is for the User to decide whether to resubmit the transaction immediately to the database system or to look first into the problems that lead to the rejection of this transaction.

- A failure that effects all the transactions that are active at the moment, a so called All Transaction Failure (ATF). If the transaction has ended succesfully but has not yet been returned by the Scheduler to the TBM, this transaction can be redone. These transacations have already a partial-commit record in the log and the information in the write records of the log can be used to install the correct data in the database. These transactions do not have to be made undone. If at the time of the failure there is no partial-commit record in the log for a particular transaction, this transaction has to be made undone.

2.4 The User

The User has three tasks, to submit transactions to the TBM, to receive the messages of committed and rejected transactions and to decide whether to resubmit a rejected transaction or not.

3 A Paradigm model for the execution of a transaction

3.1 A short introduction to Paradigm

Originally the sequential components occurring in a Paradigm model are decision processes, being a type of timed, stochastic finite automata. As plain finite automata are more common in computer science, this introduction will be based on finite automata. Stochastic aspects will be completely omitted and the time aspect will be treated intuitively only. The finite automata together with the time aspects will be called a process. A process can be represented by a graph, states are denoted by numbers, transitions by directed edges. Each transition corresponds to a unique action in the state where the edge points from. A parallel phenomenon, consisting of several sequential components is modelled as a parallel

process, being a vector of processes representing these components. The behaviour of one process can depend directly on the behaviour of other processes. Whenever this occers, this is explicitely incorporated in the model by means of some specific features. A Paradigm model for a parallel phenomenon is constructed in a stepwise manner:

- The behaviour of the sequential components of the phenomenon are modelled seperately.

- These components are split into parts, the so called *subprocesses* and *traps*. They model effects of the the communication between the the constituent processes. When a process is dependent on the behaviour of another process and no communication has yet taken place, the first process remains restricted in its behaviour until the communication has taken place. A subprocess is a temporary behaviour restriction of a process, valid between the receiving of two subsequent communications. A trap of a subprocess is a further restriction of the subprocess, marking its readiness to receive communication, i.e. its readiness to receive an order to behave according to a next subprocess. So by entering a trap, a subprocess explicitily asks for the next subprocess i.e. for the next behaviour restriction. Therefore, a trap of a subprocess is actually called a trap *from* that very subprocess *to* the next subprocess. Thus any such trap explicitly indicates which is the next asked for subprocess. A trap of a subprocess is defined as a subset of the state space that cannot be left as long as this subprocess is prescribed. A collection of subprocesses relevant for the communication is called a *partition*. A collection of traps relevant for the communication is called a *trapstructure*

- The traps from the trapstructure make it possible to control the changeover of one subprocess to another subprocess. So traps are instruments for controlling the communication. This control of communication can be modelled by a seperate decision process, a so called *trap process*. As trap processes of a parallel phenomenon are processes themselves, the communication between the trapprocesses can also be modelled. This is sometimes done using another process, but sometimes one of the trapprocesses can be used for this purpose. A formal definition of a trap process will be given in section 3.3.

We will model the three components of the transaction manager, together with the user handling a transaction. Therefore as step one, we describe in section 3.2 the process reflecting the life-cycle of a transaction. This process, called TDE, an acronym for Transaction During Execution, will be split into parts in four different ways, one way for each of the participating components. For each of these collections of subprocesses a trapstructure and trapprocess will be given. This will be the subject of section 3.3, 3.4, 3.5 and 3.6. In section 3.7 we will moreover show how one of the trapprocesses, the one representing the Transaction Buffer Manager can be used as the trapproces controlling the communication between the previously described trapprocesses.

3.2 The model of the transaction

The decision process representing the transaction during its execution (TDE) can be graphically represented as a directed graph in figure 1, like a finite state machine.

TDE can be modelled by means of the following states and transitions:

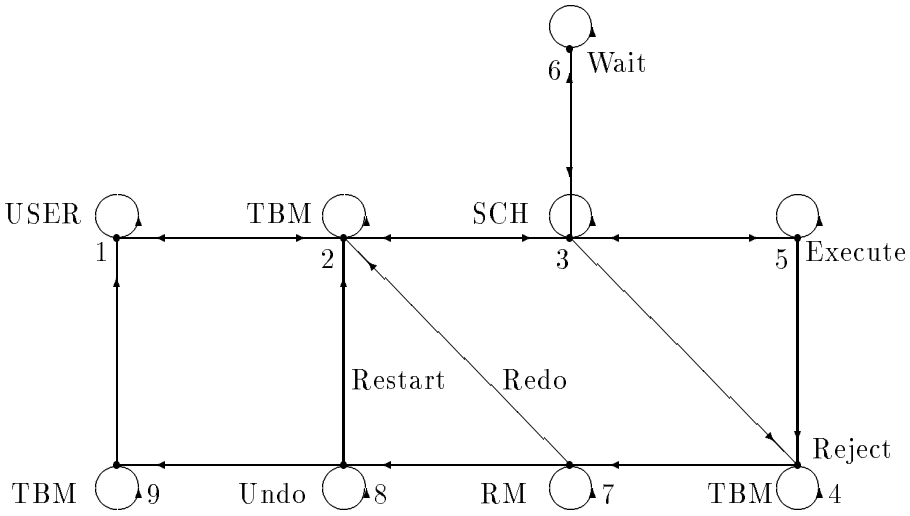

Figure 1: TDE

1 The transaction has not been submitted by the User to the Transaction Manager. If the transaction has been committed this is also reflected by state 1, in that case we say a new transaction has not yet been submitted.

2 The transaction has been submitted to the TBM, the TBM handles the transaction and writes the approriate log-record in the log.

3 The transaction has been submitted to the Scheduler, the Scheduler handles the transaction.

4 After a part of the execution the transaction has to be rejected, or the concurrency control method forces the transaction to be restarted. The rejected transaction is submitted by the Scheduler to the TBM.

5 The transaction is being executed.

6 The transaction has to wait, to give way to another transaction either because of its time slice having passed or because of the wound-wait protocol.

7 The Recovery Manager is deciding what should be done with the transaction.

8 The transaction is being made undone.

9 The undone transaction is submitted by the Recovery Manager to the TBM.

The interpretations of the transitions between the states is as follows:

1→1 The transaction has not yet been submitted to the Transaction Manager.

1→2 The transaction is submitted to the Transaction Manager.

2→1 The committed transaction is submitted to the User.

2→2 The TBM is handling the transaction.

2→3 The TBM submits the transaction to the Scheduler.

3→3 The Scheduler is handling the transaction.

3→2 The Scheduler submits a succesfull transaction to the TBM.

3→5 The transaction will be executed.

3→6 The transaction has to wait.

3→4 The transaction is rejected. This can have its origin in two reasons: either the wound-wait protocol has decided so and the transaction can be restarted, or a failure has taken place at a moment the transaction was not in execution.

5→5 The transaction is being executed.

5→3 The transaction cannot be executed further, because it has ended succesfully or because it has to wait.

5→4 The transaction is rejected during its execution.

4→4 The TBM prepares a submit of the rejected transaction to the Recovery Manager.

4→7 The rejected transaction is submitted to the Recovery Manager.

6→6 The transaction is waiting.

6→3 The transaction is reactivated.

7→7 The Recovery Manager is handling the transaction, e.g. information is gathered to support the decision whether a redo of the transaction will take place or an undo.

7→2 The transaction will be redone.

7→8 The transaction will be undone.

8→8 The transaction is being made undone.

8→2 The undone transaction will be restarted automatically by the TBM.

8→9 The undone transaction is submitted to the TBM.

9→1 The undone transaction is submitted from the TBM to the User.

9→9 The TBM prepares a submit of the undone transaction to the User.

Because several transactions can be executed simultaneously, the database can be in several of these states at the same time. The three components of the Transaction Manager together with the User can be considered as being able to hande two or more transactions simultaneously, therefore two or more transactions can be in the same state. These components are now to be discussed in more detail.

3.3 The Transaction Buffer Manager

Four subprocesses, AI, AII, AIII and AIV of the TDE can be distinguished in relation to the TBM. They will not be discussed in detail, only the corresponding graphs are presented, in the figures 2 and 3. Subprocesses are drawn as parts of the finite automata they are subprocesses of. Traps are visualized by means of double-sided polygons.

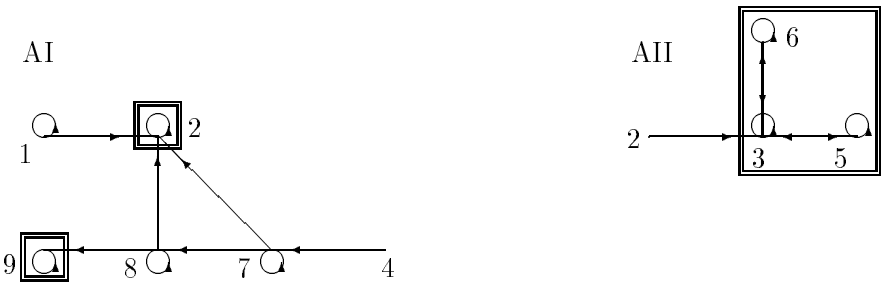

Figure 2: The subprocesses AI and AII with respect to TBM.

The two traps of subproces AI indicate that a transaction has been submitted to the TBM, either by the User or by the Recovery Manager. A rejected transaction ends, as far as the Scheduler is concerned, in state 4. In this state, the Scheduler has submitted the transaction already to the TBM. The transaction can never enter the process of execution again unless first the TBM has handed the transaction over to the RM. The succesfull transactions end, as far as the Scheduler is concerned, in state 3. This can be in subprocess AII or AIII. The traps of AIII indicate the TBM waiting for the result of the transaction with respect to the Scheduler. In AIV the TBM hands a transaction over to the User.

TDE is a decision process with partition V^1 = { AI, AII, AIII, AIV }. We choose as a trapstructure TS^1 for this partition:

- Trap from AI to AII { 2 }
- Trap from AI to AIV { 9 }
- Trap from AII to AIII { 3, 5, 6 }
- Trap from AIII to AIV { 2 }
- Trap from AIII to AI { 4 }

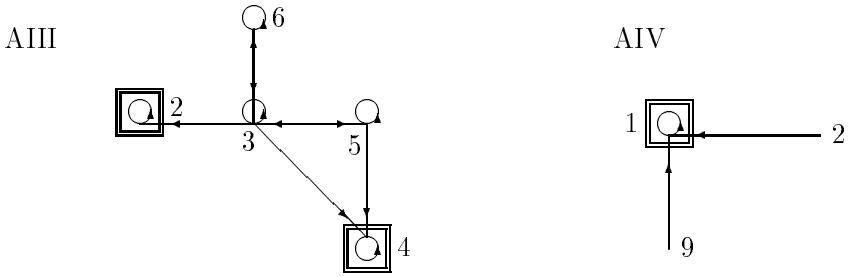

Figure 3: The subprocesses AIII and AIV with respect to TBM.

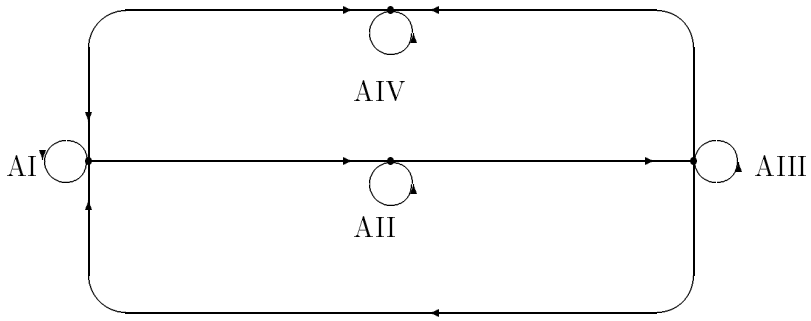

Figure 4: The trap proces for partition V^1

- Trap from AIV to AI { 1 }

A trap process is always defined with respect to a decision process, say DP with a partition, say V and a trapstructure, say TS. The *trap process of DP with respect to V and TS* is a decision process T^* with state space the set V, such that the following dependencies hold:

1. Whenever T^* is in state $v^i \in V$, the decision process DP will behave according to subprocesses v^i
2. only after DP is trapped in a trap $S^{i,j} \in TS$, with $S^{i,j}$ a trap from v^i to v^j, T^* can transit to state $v^j \in V$, thereby forcing DP to behave according to subprocess v^j.

In this way the trap process controls the transitions between the subprocesses defined by the partition V and the trapstructure TS. In this paper we will give a somewhat informal description of the trap processes by presenting the corresponding graph. The graphical representation of the trap process for V^1 with respect to one transaction is shown in figure 4.

This trap process can be interpreted as follows. (Note that the interpretation is formulated with respect to one fixed transaction.) Initially the transaction has not yet been submitted to the TBM by the user: the TBM prescribes subprocess AI. After the transaction has been submitted to the TBM, TBM will prescribe subprocess AII in order to submit the transaction to the Scheduler. After the Scheduler has taken over, TBM prescribes subprocess AIII, indicating TBM waits until the Scheduler is ready with the transaction. If the transaction has been succesfull, TBM will prescribe subprocess AIV (submitting to the User) and somewhat later it prescribes AI. If the transaction has not been succesfull, TBM will prescribe subprocess AI, so then the Recovery Manager processes the transaction. After the RM has finished its processing, the transaction is resubmitted to the TBM, either through trap {2} or trap {9}. In case of trap {2}, TBM again prescribes AII, thereby submitting the transaction to the Scheduler. And in case of trap {9} TBM prescribes AIV, returning the transaction to the User.

In the multi-user environment the TBM will have to handle a number of transactions. Notice that only at the moment TBM is in subprocesses AII and AIV the TBM cannot handle another transaction, for instance by selecting a new transaction to be handled. As soon as a transaction is transferred to the Scheduler, to the Recovery Manager or to the User, TBM returns to state AI or AIII in which it can select a new transaction. We will call this trap process, adapted to the multi-user environment TP^1. The partition of TDE with

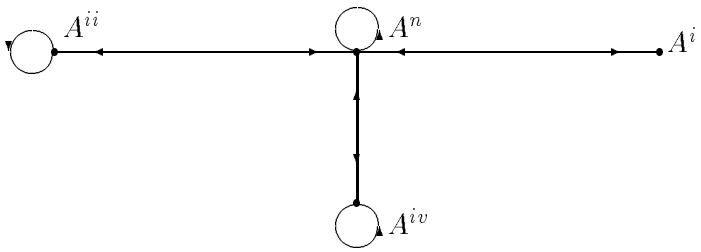

Figure 5: The adapted trap proces TP^1.

respect to this adapted trapprocess will be denoted by P^1 and it consists of the subprocesses A^n, (A- neutral), A^{ii}, A^{iv} and A^i, corresponding to respectively AI/AIII, AII, AIV and AI. This trap process for the multiuser environment can be represented appropriately by figure 5.

The states have the following interpretations, i.e. they prescribe the following subprocesses to TDE:

A^i , submit to RM.

A^{ii} , submit to Scheduler.

A^n , the neutral subprocess

A^{iv} submit to User

The transitions only take place after TDE has entered the following traps:

$A^n \rightarrow A^{ii}$ trapped in $\{2\}$

$A^{ii} \rightarrow A^n$ trapped in $\{3, 5, 6\}$

$A^n \rightarrow A^{iv}$ trapped in $\{2\}$ or $\{9\}$

$A^{iv} \rightarrow A^n$ trapped in $\{1\}$

$A^n \rightarrow A^i\mathbf{11}$ trapped in $\{4\}$

$A^i \rightarrow A^i\mathbf{13}$ immediate hence no change occurs.

Although AI and AIII represent together the neutral position, in the graph AI also received an extra node. This is to emphasize that submitting a transaction to the RM is done in subprocess AI. Note that in state 11 there is no loop. The return to state 13 is immmediate, as this does not correspond to any change of TDE's current subprocess.

3.4 The Scheduler

The Scheduler determines the order of the execution of the transactions. In this model we moreover assume it is responsible for the execution itself. In a more comprehensive model however, we should prefer to incorporate a seperate component, for instance called Data Manager, responsible for the actual execution of the transaction(s). In the model discussed here, we just let SCH itself react to TDE being in state 5 or in 6 or in 3. We distinguish

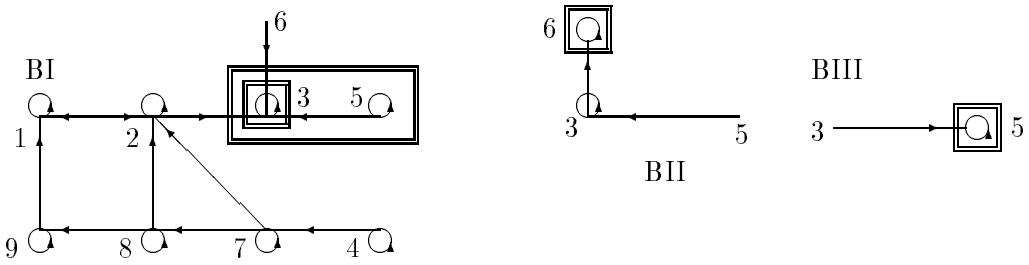

Figure 6: The subprocesses BI, BII and BIII with respect to SCH.

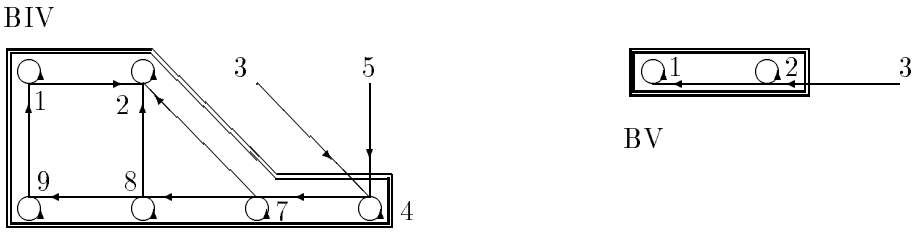

Figure 7: The subprocesses BIV and BV with respect to SCH

the following five subprocesses of TDE in relation to the Scheduler: BI, BII, BIII, BIV and BV, represented in the figures 6 and 7.

The traps of subproces BI indicate that a transaction is to be handled by the Scheduler. The trap {3, 5} indicates the position of a transaction that will not be executed further: it either has to be queued or to be rejected. In BII and BIII the traps indicate that a transaction is in a wait state or will be executed respectively. The trap of subproces BIV indicates that an unsuccesfull transaction is given to the TBM. In the trap of BV the Scheduler handes a succesfull transaction over to the TBM.

TDE is a decision process with partition V^2 = { BI, BII, BIII, BIV, BV }. We choose as a trapstructure TS^2 for this partition:

- Trap from BI to BII or BIV { 3, 5 }
- Trap from BI to BIII or BV { 3 }
- Trap from BII to BI { 6 }
- Trap from BIII to BI or BIV { 5 }
- Trap from BIV to BI { 1, 2, 4, 7, 8, 9 }
- Trap from BV to BI { 1, 2 }

The graphical representation of this trap process is shown in figure 8. For a specific transaction, the Scheduler prescribes BI or BII most of the time. BIV and BV are only prescribed if the Scheduler has to hand the transaction over to the TBM. Observe that it is TBM's responsibility to determine when it will react to this. Therefore the traps of BIV and BV have been chosen such that the Scheduler can prescribe BI as soon as possible. without having to wait until the TBM is taking over. Because the transitions between the subprocesses BI, BII and BIII are not dependent of the behaviour of another component, together they can represent a neutral subprocess. This adaption of the trap process is shown in figure 9. We

will call this trap process TP^2. The partition for this adapted trapprocess will be denoted by P^2 and consists of the subprocesses B^n, (B-neutral), B^{iv} and B^v.

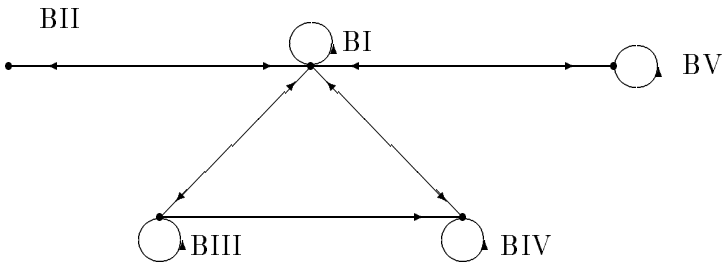

Figure 8: The trap proces for V^2.

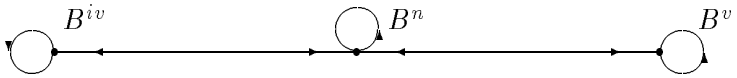

Figure 9: The adapted trap proces for V^2.

3.5 The Recovery Manager

The subprocesses we distinguish in relation to the Recovery Manager are CI, CII, CIII and CIV. These subprocesses are represented in the figures 10 and 11.

The trap in CI indicates the Recovery Manager is deciding what to do with the transaction. The trap in subprocess CII indicates a transaction has been resubmitted to the TBM and in CIII that the transaction has to be made undone. In CIV the trap indicates a transaction has been made undone and is resubmitted to the User. The TDE is also a decision process with partition V^3 = { CI, CII, CIII, CIV }. We choose as a trapstructure TS^3 for this partition:

- Trap from CI to CII or CIII { 7 }
- Trap from CII to CI { 1, 2, 3, 4, 5, 6 }
- Trap from CIV to CI { 1, 2, 3, 4, 5, 6, 9 }
- Trap from CIII to CII or CIV { 8 }

RM can play the role of this trapprocess of TDE with respect to partition V^3 and trapstructure TS^3. The trap process, represented in figure 12, can be interpreted as follows. In CI and only after the transaction is trapped in {7}, the RM has to decide what has to be done with the submitted transaction. If there is enough information in the log to redo the transaction, first CII will be prescribed and after that the RM can return to CI as soon as possible. If the transaction has to be undone it prescribes CIII. If the transaction has to be restarted after being made undone, CII will be prescribed after CIII and after that the RM can return to subprocess CI as soon as possible. If the transaction will not be restarted the transaction will be returned to the User, so subprocess CIV will be prescribed whereafter RM can return to CI (as soon as posible).

For a specific transaction, the RM prescribes CI or CIII most of the time. CII and CIV are only prescribed if the RM hands the transaction over to the TBM. Analogue to the

situation with the Scheduler, it is the responsibility of the TBM to determine when to react to this, and the traps of CII and CIV have been chosen in a way that the RM can prescribe CI as soon as possible, without having to wait until the TBM is taking over. Because the transitions between the subprocesses CI and CIII are not dependent of the behaviour of another component, together they can represent a neutral subprocess. This adaption of the trapprocess is shown in figure 9. We will call this trapprocess TP^3. The partition for this adapted trapprocess will be denoted by P^3 and consists of the subprocesses C^n, (C-neutral), C^{ii} and C^{iv}.

3.6 The User

The subprocesses of the TDE in relation to the user are DI and DII, as represented in figure 14. TDE is a decision process with partition V^4 = { DI, DII }. We choose as a trap structure TS^4 for this partition:

- Trap from DI to DII { 1 }
- Trap from DII to DI { 2, 3, 4, 5, 6, 7, 8, 9 }

The User can play the role of this trap process of TDE with respect to partition V^4 and trapstructure TS^4. We will call this trap process TP^4. This trap process does not need to be adapted for the multi-user environment as it is assumed that all the transactions are submitted to the system at a different moment. DI can be considered as the neutral subprocess. In this subprocess the User prepares to submit a transaction to the system and receives the messages of the transactions that have come to an end, succesfull or not. In DII it hands a transaction over to Transaction Manager. The partition this trapprocess will be denoted by P^4 and consists of the subprocesses D^n, (D- neutral, the same as DI) and D^{ii}. A graphical representation of the trap process is shown in figure 15.

3.7 A trapprocess for the subprocesses

In this section we will show that the adapted trap process for TBM TP^1 can serve as the parallel trapprocess for the four partitions.

To this aim we will first describe the parallel trap process for the four participating processes TP^i, $i \in 1,2,3,4$. The trapproces T^*, defined as $[T^1, T^2, .., T^4]$ is the trap process with respect to the partition of the four participating processes and trapstructures. So this process is the controlling process for all the TDE's. We are interested in the question whether we can describe this process as a sequential process. Therefore we look into the state space of T^* , being $X_{T^*} = \prod_{i=1}^{4}\{V^i\}$. Not all the states of this control process T^* are acceptable however, as this would allow several transactions to be handled over from one component to another component at the same time. This we did not allow by demanding that the TBM can handle over transactions only one at the time. Therefore we will restrict the state space of T^* to be the set of the tuples having maximal one value unequal to the neutral position of the TBM. This set consists of exactly the following 9 tuples:

1 A^n, B^n, C^n and D^n, the three components and the User can be handling transactions.

2 A^{ii}, B^n, C^n and D^n, the TBM submits a transaction to the SCH, the other two components can handle other transactions.

3 A^{iv}, B^n, C^n and D^n, the TBM submits a transaction to the User, the other two components can handle other transactions..

4 A^i, B^n, C^n and D^n, the TBM submits a transaction to the RM and, as in the previous and the following states, the other two components can handle other transactions.

5 A^n, B^{iv}, C^n and D^n, the SCH submits a succesfull transaction to the TBM.

6 A^n, B^v, C^n and D^n, the SCH submits an unsuccesfull transaction to the TBM.

7 A^n, B^n, C^{ii} and D^n, the RM submits a transaction to the TBM that has to be redone or restarted.

8 A^n, B^n, C^{iv} and D^n the RM submits an undone transaction to the TBM.

9 A^n, B^n, C^n and D^{ii}, the User submits a transaction to the TBM.

A graphical representation of this process is given in figure 16. We can conclude that it is possible to describe the trapproces T^* as a sequential process. Furthermore we see a clear a anology with trapproces TS^1 as described in figure 5 together with the textual comments:

- In both processes the TBM submits transactions to the SCH. In this parallel trapprocess the different reactions of the SCH are visible: a transaction can be rejected (state 5) or the transactions can be successfull (state 6).
- Also, in both processes the TBM hands a transaction over to the RM. In the parallel trapprocess two reactions of the RM are distinguished: on the one hand the redo and restarts (state 7) and on the other hand the undone transactions (state 8).
- Submitting a transaction to the User is done in complete anology.
- The way the user handes a transaction over to the TBM is done in an explicite state, state 9, in T^*. Because this can only be done if the TBM is in its neutral position, this can be reflected as one single state.

So the trapprocess TS^1 can control the communication of the different components of the transaction manager as we have modelled it, no extra trap process is necessary, only additional variables to transfer certain values.

We now formulate the following conclusions. The TDE and the transaction manager both are relatively simple, therefore the corresponding Paradigm model of this situation is correspondingly simple. Therefore, this example gives us hope that also in more complicated and more realistic situations a Paradigm model of the various processes involved will result in a clear and precise specification thereof.

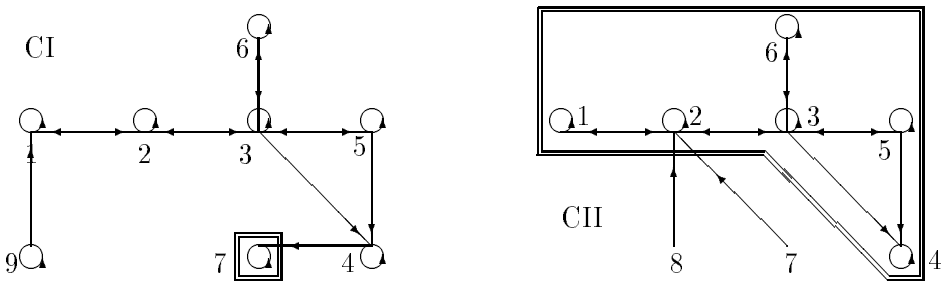

Figure 10: Subprocesses CI and CII with respect to the RM.

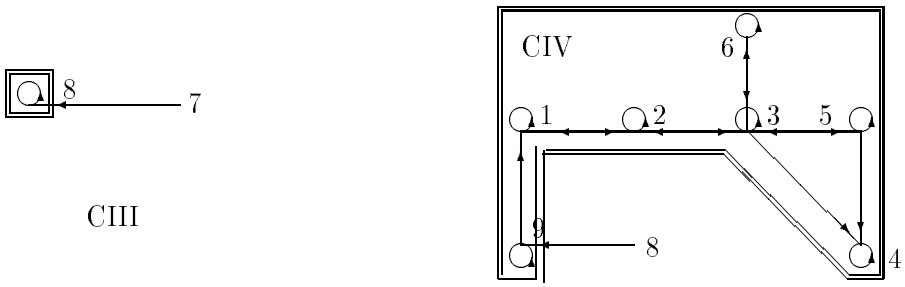

Figure 11: Subprocesses CIII and CIV with respect to the RM.

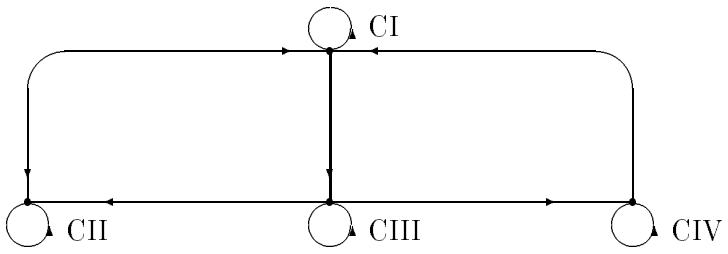

Figure 12: The trap proces for partition V^3.

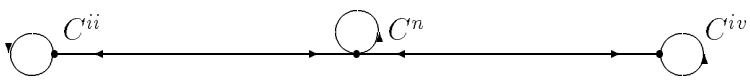

Figure 13: The adapted trap proces TP^3.

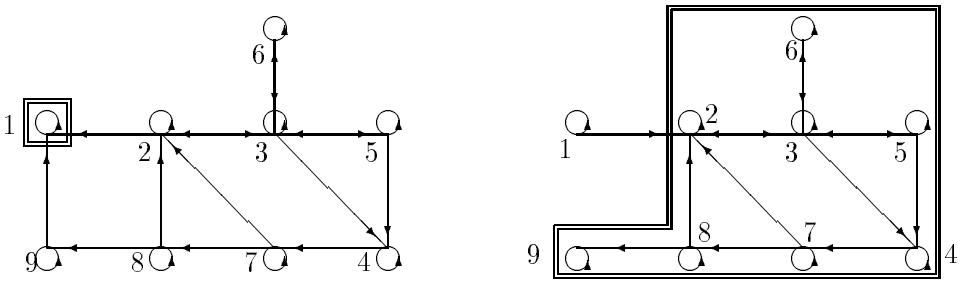

Figure 14: The subprocesses DI and DII with respect to the User.

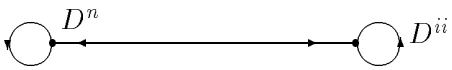

Figure 15: The trapproces for partition V^4.

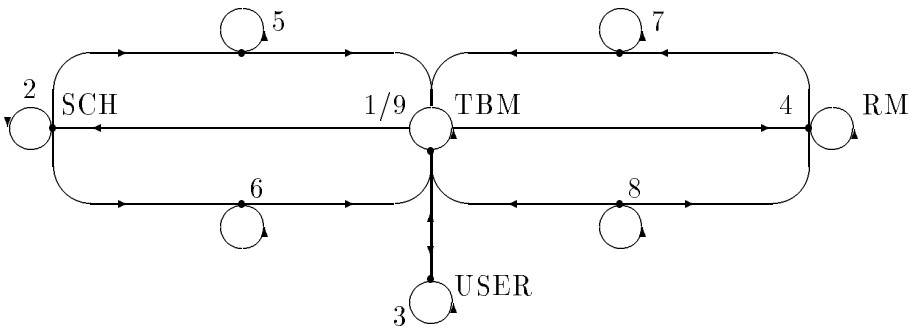

Figure 16: The parallel trapprocess

4 A translation into an object-oriented program

In this context object-oriented refers to the structure of the program. The program is a collection of autonomous, interacting components, realizing the function of the User and the three components of the Transaction Manager. The behaviour of the four components is reflected in four procedures serving as objects which are supposed to be simultaneously in execution. The interaction is realized by means of waiting queues and flags. The TBM, the Scheduler and the Recovery Manager each have their own waiting queue and behave in the following manner:

1 The component is in its neutral subprocess and chooses a transaction from its waiting queue.

2 It reads the flag of the transaction, the old flag.

3 It handles the transaction.

4 The component prescibes another subprocess and puts the transaction in the appropriate waiting queue for another component with a new flag.

5 The component returns to its neutral subprocess.

The User behaves in a different manner. It "produces" transactions and can "destroy" them. Producing a transaction is done by giving a transaction an identification and putting the transaction in the waiting queue for the TBM. The User produces transactions as long as (s)he has the urge to do so. In the program this is reflected by the boolean variable Urge. The waiting queue for the User consists of the committed and undone transactions. The User can decide to resubmit an undone transaction, that is, to put the transaction in the waiting queue for the TBM. The user destroys the committed transactions and the transactions that will not be resubmited by removing them from the waiting queue.

The function of the main program is only to start the four procedures and to create the structure of the global variables, i.e. the waiting queues. In the procedures the states of the adapted trapprocess are used to show the behaviour of the components. The states of the original trapprocesses are only used if it can illustrate relevant detail information about this behaviour.

Program Transactie Management;
Global Variables:

Priority Queues:	UserList,	TBMList,	SCHList,	RMList;
Boolean:	UserListempty,	TBMListempty,	SCHListempty,	RMListempty;
Init:	True,	True,	True,	True;

Begin Main Program
    begin
        Start Procedure TrapUser;
        Start Procedure TrapTBM;
        Start Procedure TrapSCH;
        Start Procedure TrapRM;
    end
End Program

```
Procedure TrapUser; See fig.15
OldFlag : List = ( Undone, Commit ); { from TBM }
NewFlag : List = ( Start );              { for TBM }
Boolean : Resubmit, Urge;
Init      : True,      True;
While True do { User is in D^n}
    while Urge do
        begin
            give the transaction an identifcation, say Tx;
            NewFlag=Start;
            enter subprocess D^ii;
            put Tx into the TBMList;
            return to subprocess D^n
        end; { User is in D^n }
    while no Urge and Userlistempty=false do
        begin
            Choose a transaction, say Tx from the UserList; read Oldflag;
            if OldFlag = Commit then remove Tx from UserList;
            if OldFlag = Undone then
                begin
                    decide whether Tx has to be resubmitted or not
                    if Resubmit then
                        begin
                            NewFlag:= Start;
                            enter subprocess D^ii;
                            put Tx into the TBMList;
                            return to subprocess D^n;
                        end
                    else remove Tx from UserList
                end { User is in D^n }
        end
End Procedure User

Procedure TrapTBM; see fig.1 and 5.
OldFlag: List=( Start, Ok, Unknown, Restart, Undo, Undone, Redo );
                { Start from User; OK, Unknown, Restart, Undo from SCH;
                Restart, Undone and Redo from RM}
Newflag: List=( Go, Goredo, Commit, Unknown, Restart, Undo, Undone);
                { Go and Goredo for SCH; Commit and Undone for User,
                Unknown, Restart, Redo and Undo for RM}
While True do { TBM in A^n}
Begin
        if TBMListempty= true then wait
{The operating system is assumed to check regularly the conditions of waiting processes in order to reactiv
                begin
                Choose a transaction, say Tx from the TBMList;
                Read OldFlag; { Tx in 2 or 4}
                if OldFlag = Start { Tx in 2, TBM in AI } then
```

```
            begin
                NewFlag = Go;
                enter subprocess A^ii;
                put Tx into SCHList;
                return to subprocess A^n; {TBM in A^n}
            end
        if OldFlag = Ok { Tx in 2, TBM in AIII} then
            begin
                NewFlag = Commit;
                enter subprocess A^iv;
                put Tx into UserList;
                return to subprocess A^n; {TBM in A^n}
            end
        if OldFlag = Unknown or Restart or Undo { Tx in 4, TBM in AIII} then
            begin
                NewFlag:=OldFlag;
                enter subprocess A^i;
                put Tx into RMList; {TBM in A^n}
            end
        if OldFlag = Redo { Tx in 2, TBM in AI } then
            begin
                NewFlag:= Goredo;
                enter subprocess A^ii;
                put Tx into SCHList;
                return to subprocess A^n;{TBM in A^n}
            end
        if OldFlag = Restart { Tx in 2, TBM in AI } then
            begin
                NewFlag:= Go;
                enter subprocess A^ii;
                put Tx into SCHList;
                return to subprocess A^n;
            end
        if OldFlag = Undone {Tx in 9, TBM in AI} then
            begin
                NewFlag:=OldFlag;
                enter subprocess A^iv;
                put Tx into UserList;
                return to subprocess A^n; { TBM in A^n}
            end;
        remove Tx from TBMList; { TBM in A^n}
    end
End Procedure TBM;

Procedure TrapScheduler; see fig. 1 and 9.
OldFlag: List= ( Go, Goredo, Pause);
                { Go and Goredo from TBM, Pause from Sch}
NewFlag: List=( Pause, OK, Unknown, Restart, Redo, Undo );
```

```
                { Pause for Sch; OK, Unknown, Restart, Redo and Undo for the TBM}
Boolean : ATF,  OTF,   P,      WWr,   AR;
Init    : False, False, False, False, False;
{ ATF is short for All-Transactions-Failure, OTF for One-Transaction- Failure
P is short for Pause and WWr stands for a restart due to Wound-Wait,
AR stands for AllRight, indicating a succesfull end of the execution of a transaction.}
While True do { SCH in B^n}
Begin
    if SchListempty= true then wait
    while no ATF do
        begin
            choose a transaction, say Tx from the SCHList; {Tx in 3, Sch in B^n}
            read Oldflag;
            if OldFlag = Go or Pause then
                begin
                    while not (ATF or OTF or P or WWr or AR ) do BIII { execute Tx }
                            {Tx in 5, SCH in B^n};
                    if ATF then
                        begin
                            newflag:=Unknown;
                            enter subprocess B^iv; {Tx in 7}
                            put Tx into TBMList;
                            return to subprocess BI;
                            ATF:=False; { SCH in B^n}
                        end
                    if OTF then
                        begin
                            newflag:=Undo;
                            enter subprocess B^iv; {Tx in 7}
                            put Tx into TBMList;
                            return to subprocess BI;
                            { SCH in B^n}; OTF:=False
                        end
                    if P(ause) then
                        begin
                            NewFlag:=Pause;
                            enter subprocess BI; {Tx in 3}
                            enter subprocess BII;
                            put Tx into SCHList; {Tx in 6}
                            return to subprocess BI;
                            Pause:=False; { SCH in B^n}
                        end
                    if WWr=True then
The following is not relevant for the understanding of this procedure. If another algorithm was used for de
                        begin
                            newflag:=Pause;
                            enter subprocess BI; {Tx in 3}
                            enter subprocess BII;
                            put Tx into SCHList; {Tx in 6}
```

```
                    enter subprocess BI;
                    get transaction Ty from SCHList;
                    NewFlag(Ty) = Restart;
                    enter subprocess B^iv;
                    put Ty into TBMlist; {Ty in 4}
                    WWr:=false;
                    enter subprocess BI;
                    get transaction Tx from SCHList;
                    return to subprocess BIII;
                    WWr:=False; {Tx in 5, SCH in B^n}
                end
            if AR then
                    begin
                        NewFlag:=OK;
                    enter subprocess B^v; {Tx in 2}
                        put into TBMList;
                    return to subprocess BI;
                        AR:=False; { SCH in B^n}
                    end
        end
    else {OldFlag = Goredo} { these transactions had a commit
        and are now handled without any pause. An ATF can occur.}
        begin
            while not (ATF or AR) do BI;
            if ATF then
                begin
                    NewFlag:=Redo;
                    enter subprocess B^iv;
                    put Tx into TBMList; { Tx in 4 };
                    return to subprocess BI; { SCH in B^n}
                end
            else {AR=True}
                begin
                    enter subprocess B^v;
                    put Tx into TBMList;
                    return to subprocess BI; { SCH in B^n}
                end
        end
    remove Tx from SCHlist;
end
{ATF=true};
put all elements from SCHList into TBMList with NewFlag Unknown
```

End Procedure TrapScheduler

Procedure TrapRM; see fig.1 and 13.

```
OldFlag: List=( Unknown, Restart, Redo, Undo);{ from TBM }
NewFlag: List=( Restart, Undone, Redo ); { for TBM }
Boolean: U(ndo);
```

```
Init     : False;
While True do { RM in C^n}
Begin
    if RMListempty= true then wait
    else
        begin
            Choose a transaction, say Tx from the RMList; {Tx in 7 }
            Read Oldflag;
            if OldFlag = Unknown then
                begin
                    decide whether the transaction has to be
                    redone or be made undone;
                    if undone necessary then U=true
                end
            if OldFlag=Restart or OldFlag=Redo or OldFlag=Undo or U=true then
                begin
                    enter subprocess CIII; {Tx in 8};
                    make Tx undone;
                    newflag:=Undone;
                    if Oldflag=Undo or U=True then
                        begin
                            enter subprocess C^iv;
                            put Tx into TBMList;
                            return to subprocess CI; { RM in C^n}
                        end
                    if OldFlag= Restart then
                        begin
                            enter subprocess C^ii; NewFlag:=Restart;
                            put into TBMlist; { Tx in 2}
                            return to subprocess CI; { RM in C^n}
                        end
                    if U=false or Oldflag=Redo then
                        begin
                            NewFlag = Redo;
                            enter subprocess C^ii; {Tx in 2}
                            put into TBMlist;
                            enter subprocess CI; {RM in C^n}
                        end
                    { RM in C^n }
            U:=False;
        remove Tx from RMlist;
        end
End Procedure TrapRM;
```

We see how the trapprocesses can easily be translated into procedures. The procedure TrapTBM having a central role in handing the transaction over from one component to another. Allthough the model is clear, it might not be a solid base for an implementation

as it might put a realtive heavy load on the communication links among the processors. Still, the model and the corresponding procedures offer more insight in the different tasks of a transaction manager and the corresponding intricate influences.

References

[Groe] Groenewegen, L.P.J. *Parallel Phenomena, a series consisting of technical reports. 1986-1990* Department of Computer Science, University of Leiden.

[MoGr] Morsink, J.A., and Groenewegen, L.P.J. *Behavioural-oriented modelling if structural restrictions. 1990* Department of Computer Science, University of Leiden no 90-01.

[Gray] Gray, J. *The transaction Concept: Virtues and Limitations.1981* Lecture Notes in Computer Science, Conference Proceedings, The Netherlands 1981.

[MaGr1] Van der Made-Potuyt, S.C. and Groenewegen, L.P.J. *Modelling a transaction manager using parallel decision processes, 1990* Department of Computer Science, Erasmus University Rotterdam, Report EUR-CS-90-07.

[MaTr] Van der Made-Potuyt, S.C.*The transaction concept and its implementation. 1989* Proc. Benchmarking Database Functionality. Codd & Date Ltd., Berlin.

[SLR] Stearns, R.E., Lewis, P.M.,II, Rosenkrantz, D.J. *Concurrency Controls for Database Systems* Proc.17th Symp. on foundations of Computer Sciences,pages 19-32.IEEE, 1976.

[Esea] Eswaran, K.P., Gray, J.N., Lorie, R.A., Taiger, I.L. *The notions of Consistency and Predicate Locks in a Database System* Comm.ACM 19(11):624-633,November,1976.

[BHG] Bernstein, P.A., Hadzilacos, V., Goodman, N. *Concurrency Control and Recovery in Database Systems* Addison Wesley, 1987.