

Atomicity Implementation in Mobile Computing

LARS FRANK

Department of Informatics, Copenhagen Business School,
Howitzvej 60, DK-2000 Frederiksberg, Denmark
e-mail: frank@CBS.DK

Abstract

Distributed systems using mobile computing do not have the traditional ACID properties (Atomicity, Consistency, Isolation and Durability), and, therefore, consistency problems may occur. These problems may be managed by using semantic ACID properties, i.e. from an application point of view, the system should function as if all the traditional ACID properties had been implemented. In this paper the objective is to describe in detail how the global semantic atomicity property may be implemented in systems using mobile computing. How to implement the other global semantic ACID properties is described in Frank and Zahle (1998).

Keywords: ACID properties, atomicity, mobile computing, multidatabases, fault tolerance, client/server technology

1. Introduction

In the transaction model described in this paper, the global atomicity property is implemented by using retrievable, pivot and compensatable subtransactions in that order. The global consistency property can be managed by the application programs themselves supported by tools. The global isolation property is implemented by using countermeasures (Frank and Zahle, 1998) to the missing isolation of the updating transactions. The global durability property is implemented by using the durability property of the local DBMS systems.

The atomicity algorithm described in this paper is a transaction shell with all the necessary types of database accesses, but without the application logic. This transaction shell must be used by all distributed transactions. However, by using our transaction shell the development costs of new applications may be reduced.

We will also integrate the basic backup/replication designs in our transaction model in order to make mobile computing more fault tolerant. This will also allow us to improve performance by substituting remote data accesses by local data accesses. The response time of the system may be improved too.

We have been cooperating with one of the major ERP (Enterprise Resource Planning) software companies in designing a distributed version of their financial management and accounting system by using the techniques described in this paper. We have now started to implement a prototype of the system that may be integrated with mobile computing as described in this paper.

The paper is organized as follows:

Section 2 will describe an extended transaction model that provides semantic ACID properties. In section 3 we will describe the basic backup/replication designs that may be used in mobile computing. In section 4 we will integrate nested transactions in our transaction model and describe a transaction shell for implementing the atomicity property in mobile computing optimized for high performance and availability. Concluding remarks are presented in section 5.

Related Research: Different versions of the 1-safe and 2-safe backup designs are described in e.g. Garcia-Molina and Polyzois (1990), Polyzois and Garcia-Molina (1994), Gallersdörfer and Nicola (1995) and Humborstad et al. (1997). The 0-safe design has been used in practice for some years (Frank and Zahle, 1998). The 0-safe and the new 1/0-safe designs are described and evaluated in Frank (1998 and 1999c).

The transaction model described in section 2 is *The Countermeasure Transaction Model* from Frank and Zahle (1998) and Frank (1999a). This model owes many of its properties to e.g. Garcia-Molina and Salem (1987), Mehrotra (1992), Weikum and Schek (1992) and Zhang (1994). How to use our transaction model in banking systems is described in Frank and Zahle (1998). Frank (1999b) describes how to integrate the transaction model in systems for CSCW (Computer Supported Cooperative Work) and Frank (1999c) describes how to use the transaction model in electronic commerce.

2. The Transaction Model

A *multidatabase* is a union of local autonomous databases. *Global transactions* access data located in more than one local database (Gray and Reuter, 1993). In recent years, many transaction models have been designed in order to integrate local databases without using a distributed DBMS. The countermeasure transaction model, Frank and Zahle, 1998, has, among other things, selected and integrated properties from these transaction models in order to reduce the problems of the missing ACID properties in a distributed database not managed by a distributed DBMS. In The Countermeasure Transaction Model a global transaction consists of a *root transaction* (client transaction) and several single site *subtransactions* (server transactions). The subtransactions themselves can be nested transactions; i.e. a subtransaction may be a *parent transaction* for other subtransactions.

All communication with the user is managed from the root transaction, and all data is accessed through subtransactions. A subtransaction is either an execution of a *stored procedure* that automatically returns control to the parent transaction or an execution of a *stored program* that does not return control to the parent transaction.

All remote subtransactions are accessed through one of the following types of tools:

Remote Procedure Call (RPC).

From a programmer's point of view an RPC functions like a normal procedure call, except that the procedure call and the procedure itself are stored at different sites. RPCs have the following properties, which are important from a performance and an atomicity point of view:

- If a parent transaction executes several RPCs, the corresponding stored procedures are executed one at a time.
- A stored procedure managed from an RPC has only local ACID properties.
- The stored procedure automatically returns control to the parent transaction.

Update Propagation (UP).

Update propagation is here used in the general sense of propagation of any update (not just replicas). The UP tool works in the following way:

The parent transaction makes the UP "call" by storing a so-called *transaction record* in persistent storage at the parent location. The parent transaction id, the id of the subtransaction and the parameters of the subtransaction are stored in the transaction record. If the parent transaction fails, the transaction

record will be rolled back, and consequently the subtransaction will not be executed. When the parent transaction is committed, the transaction record is secured in persistent storage, and we say that the UP is *initiated*. After the initiation of the UP the transaction record will be read and sent by the UP tool (which uses a communication protocol) to the location of the corresponding subtransaction. If the subtransaction fails, the transaction record will be resubmitted until the subtransaction is committed. UPs may be implemented by using either *push* or *pull* technology. How different types of UPs have to be implemented is described in Frank and Zahle (1998). Some non-heterogeneous versions of these tools have been implemented in e.g. BD2 and Oracle DBMS software. UPs have the following properties, which are important from a performance and an atomicity point of view:

- If a parent transaction initiates several UPs, the corresponding, stored programs may be executed in parallel.
- A stored program initiated from a UP has atomicity together with the parent transaction, i.e. either both are executed or none are.
- The stored program does not automatically return control to the parent transaction.

Transaction Messages

A message send through a communication network can be used to start a stored program in a remote location. In the following this is called a *transaction message*. Transactions started by a transaction message has the following properties:

- If a parent transaction sends several transaction messages, the corresponding, stored programs may be executed in parallel.
- A stored program started by a transaction message has only local ACID properties.
- The stored program does not automatically return control to the parent transaction.

In this paper we use a nested version of The Countermeasure Transaction Model, and, therefore, we will give a broad outline of how semantic ACID properties are implemented in this model.

2.1 The Atomicity Property

An updating transaction has the *atomicity property* and is called *atomic* if either all or none of its updatings are executed. In The Countermeasure Transaction Model the global transaction is partitioned into the following

types of subtransactions that are executed in different locations:

1. The *pivot* subtransaction that manage the atomicity of the global transaction. That is, the global transaction is committed when the pivot subtransaction is committed locally. If the pivot subtransaction aborts, all the updatings of the other subtransactions must be compensated or not executed.
2. The *compensatable* subtransactions that all may be compensated. Compensatable subtransactions must always be executed before the pivot subtransaction is executed in order to make it possible to compensate them if the pivot subtransaction cannot be committed. Compensation is achieved by executing a *compensating* subtransaction.
3. The *retriable* subtransactions that are designed in such a way that the execution is guaranteed to commit locally (sooner or later) if the pivot subtransaction is committed. A UP tool is used to automatically resubmit the request for execution until the subtransaction has been committed locally, i.e. the UP tool is used to force the retriable subtransaction to be executed.

The global atomicity of ‘the single pivot transaction’ models is implemented by executing compensatable, pivot and retriable subtransactions in that order.

RPCs or corresponding transaction messages can be used to call/start the compensatable subtransactions and/or a pivot subtransaction, because the executions are not mandatory for these subtransactions from a global atomicity point of view. (If any problems occur before the pivot commit, we can compensate the first part of the global transaction).

After the commit decision of the global transaction, all the remaining updatings are mandatory. Therefore, UPs are always used to execute the retriable subtransactions, which are always executed after the global commitment.

If the pivot fails or cannot be executed the execution of all the compensating subtransactions are mandatory. Therefore, UPs are always used to execute the retriable compensating subtransactions.

Example 2.1

Let us suppose that an amount of money is to be moved from an account in one location to an account in another location. In such a case the global transaction may be designed as a root transaction that calls a compensatable withdrawal subtransaction and a retriable deposit subtransaction. Since there is no inherent pivot subtransaction, the withdrawal subtransaction may be chosen as pivot.

In other words, the root transaction executed at the user’s PC may call a pivot subtransaction executed at the bank of the user, which has a UP that “initiates” the retriable deposit subtransaction.

If the pivot withdrawal is committed, the retriable deposit subtransaction will automatically be executed and committed later. If the pivot subtransaction fails, the pivot subtransaction will be backed out by the local DBMS. In such a situation the retriable deposit subtransaction will not be executed.

2.2 The Consistency Property

A database is *consistent* if the data in the database obeys the consistency rules of the database. If the database is consistent when a transaction starts and also is consistent when the transaction is committed then the execution has the *consistency property*. Transaction *Consistency rules* may be implemented as a control program that rejects the commitment of transactions, which do not obey the consistency rules.

In The Countermeasure Transaction Model the global consistency property must be managed by the transactions themselves, e.g. local referential integrity may be managed by a local DBMS, while e.g. referential integrity between sites must be managed by the global transactions themselves.

2.3 The Isolation Property

A transaction is executed with the *isolation property* if the updatings of the transaction only are seen by other transactions after the updatings of the transaction have been committed. Often systems for mobile computing do not have the isolation property, because the transactions may be *long-lived* (Gray and Reuter, 1993) and it is not acceptable to lock data for a long period of time.

In The Countermeasure Transaction Model the global semantic isolation property is managed by using countermeasures against the isolation anomalies that occur when transactions are executed without the isolation property. If there is no isolation property, only the *lost update*, the *dirty read* and the *non-repeatable read* isolation anomalies may occur (Gray and Reuter, 1993 and Berenson et al., 1995). Therefore, The Countermeasure Transaction Model describes countermeasures that reduce the problems of these anomalies. In the following we only describe *The Pessimistic View Countermeasure*, because this countermeasure is used in the examples of the next section.

The Pessimistic View Countermeasure reduces or eliminates the dirty read anomaly and/or the non-repeatable read anomaly by giving the users a pessimistic view of the situation. In other words, the user cannot misuse the information. The purpose is to eliminate the risk involved in using data where long duration locks should have been used. A pessimistic view countermeasure may be implemented by using:

- compensatable subtransactions for updatings which limit the options of the users.
- retrievable subtransactions for updatings which increase the options of the users.

Example 2.2

When updating stocks, accounts, vacant passenger capacity, etc. it is possible to reduce the risk of reading stock values that are not available ("dirty" or "non-repeatable" data). These pessimistic stock values will automatically be obtained if the transactions updating the stocks are designed in such a way that compensatable subtransactions (or the pivot transaction) are used to reduce the stocks and retrievable subtransactions (or the pivot transaction) are used to increase the stocks.

2.4 The Durability Property

The execution of a transaction has the durability property, if the updatings of a transaction cannot be lost after the transaction is committed. The updatings of transactions are said to be *durable* if they are stored in stable storage and secured by a log recovery system. In case a global transaction has the atomicity property the global durability property will automatically be implemented, as it is ensured by the log-system of the local DBMS systems (Breibart et al., 1992).

2.5 The Atomicity Property of Nested Transactions

In our transaction model subtransactions may be nested, i.e. a subtransaction may call another subtransaction, etc.

In *The Open Nested Transaction Model* described in Weikum and Schek (1992), the subtransactions of a compensatable subtransaction must be compensatable. (Otherwise, the parent transaction cannot be compensatable). This idea has been generalized and integrated into our transaction model in the following way:

- Subtransactions of a compensatable subtransaction must be compensatable. Please

notice that sometimes the subtransactions of a compensatable subtransaction may also be designed as retrievable which simplifies the application program and reduce the response time.

- Subtransactions of a retrievable subtransaction must also be retrievable. (Otherwise, the parent transaction cannot be retrievable).
- Subtransactions of a pivot subtransaction must either be compensatable or retrievable in that order, because normally, compensatable subtransactions must be executed before the commit of the pivot subtransaction and retrievable subtransactions must be executed after the commit of the pivot subtransaction.

Furthermore, a non-committed order should be subject to changes in systems for electronic commerce. This implies that a retrievable subtransaction, which more or less compensates a compensatable subtransaction, may be executed before the commit of the pivot subtransaction.

3. Description of the Basic Backup/Replication Methods

In this section we will give a broad outline of the basic backup/replication designs described and evaluated in Frank (1998 and 1999c) in order to enable us to integrate the backup/replication designs in our transaction model.

3.1 The 1-Safe Design

In a 1-safe design (e.g. Gray and Reuter, 1993), the primary transaction manager goes through the standard commit logic and declares completion when the commit record has been written to the local log. In a 1-safe design, throughput and response time are the same as in a single-system design. The log is asynchronously spooled to the backup system/secondary copy.

In case of a primary site failure in a 1-safe system, production may continue in single database mode at the secondary location. When the failure in the old primary copy location has been repaired, the log records from this location cannot always be used to update the new primary location (former secondary copy), because the records in the new primary copy may have been updated. *Lost transactions* are defined as the transactions committed in the failed old primary copy and not in the new primary copy at the time production restarts in the new primary

location. The main problem of the 1-safe design is that the lost transactions must be reconstructed and re-executed before the recovery process has been completed. The problem may be prevented by discontinuing production in single database mode after a failure. Therefore, the single database mode of the 1-safe design is often only used in case of a disaster or a very serious failure.

3.2 The 2-Safe Design

The 2-safe design (e.g. Gray and Reuter, 1993) synchronizes two identical copies of a table. This design is not convenient in mobile computing, where the communication line often may fail.

3.3 The 0-Safe Design

Frank and Zahle (1998) defines the basic *0-safe* backup design as two databases where each transaction first will go to the nearest database location, where it is executed and committed locally. If the transaction is an update transaction, the transaction propagates asynchronously to the other database location, where the transaction is executed and committed once more. This means that both databases normally are inconsistent and not up to date under normal production. The inconsistency must be managed by using countermeasures against the isolation anomalies as described in The Countermeasure Transaction Model. From a performance and availability point of view, the 0-safe design is the best choice. The main problem with the 0-safe design is the development costs, because countermeasures that make the transactions commutative must be implemented.

3.4 The 1/0-Safe Design

If all the clients in a 0-safe design are connected to the same primary location, the design is also 1-safe and its properties will come from either the 1-safe or the 0-safe design. Frank (1998 and 1999c) defines this mixed backup design as the *1/0-safe* design. The 1/0-safe design does not have the high performance and capacity of the 0-safe design. On the other hand, in this design the isolation property may be implemented automatically as long as the primary copy does not fail. Normally, this makes it much cheaper to implement countermeasures against the isolation anomalies, because often it is only necessary to secure that “the lost updatings” are not lost in case of a primary copy failure.

4. Implementation of the Atomicity Property

In this section we will first illustrate how to use our nested transaction model in mobile computing used by salesmen. Next, we will describe our transaction model in details by using a transaction shell. Finally, we will describe a mobile computing transaction in more details.

4.1 Mobile Computing used by a Traveling Salesman

The following example will give a broad outline of how global mobile computing transactions may be designed by using our transaction model in a system for a traveling salesman.

Example 4.1

At first, we will give a broad outline of the table replication in a system for mobile salesmen. Next, we will describe a system for making orders and updatings of some of the most important files in a financial management and accounting system used by mobile salesmen. We have tried to optimize performance, availability and the possibility of recovery.

We will recommend that the Product table is 1-safe with a central primary copy and secondary copies in all the mobile locations of the salesmen. This may be implemented by using a standard “after image” UP tool which most DBMS suppliers can support. We do not think it is necessary to make the Product table 1/0-safe or 0-safe, because normally the file is only updated from the central location when product prices are changed or new products are introduced. Therefore, in case of a failure, it is possible to defer all updatings until the failure has been repaired, i.e. no lost updatings will occur. In other words, no risks are taken by using the cheap 1-safe design.

We recommend that the Customer table is fragmented with the 1/0-safe design. All Customer records are stored in a central location that is used in case of a local failure/disaster. The primary copies of the customer records are fragmented and each customer record are stored in the mobile database of the salesman with contact responsibility to the customer. This 1/0-safe design may be updated by using a pivot subtransaction that initiates a retrievable subtransaction that in turn updates the central secondary copy. If e.g. the balance of a customer is stored in the customer record, the customer record must be updated by the invoicing subtransaction that is described later in this example.

We recommend that the Order and Order-line tables are 0-safe and fragmented with a copy stored in the

mobile computer, where the corresponding sale took place. Other copies are stored in the server(s) of the store(s) that are going to deliver(s) the products, i.e. both the salesman and the store related to an Orderline may change an Orderline.

Suppose the Product-stock table has a record that includes the quantity of each product in stock at each store location. We recommend that the Product-stock table is 1/0-safe and fragmented with the primary copies in the locations of the corresponding stores. A central secondary Product-stock table may be used in case the server of the primary stock has failed.

When a salesman wants to make a new order the root transaction calls a compensatable subtransaction in the location of the mobile computer. This subtransaction creates an order record with relationship to the customer record in the mobile computer. Now, the salesman can make Orderlines. For each new Orderline the root transaction receives, it starts a compensatable subtransaction that creates an Orderline in the mobile computer. For each Orderline a compensatable sub-subtransaction updates asynchronously the remote stock of the product ordered in the Orderline. This will allow the salesman to make input for new orders without being connected to a store. If the first stock cannot fulfill the quantity ordered in the Orderline, another stock can be accessed by using another asynchronously compensatable sub-subtransaction. If an Orderline cannot be fulfilled, a compensatable sub-subtransaction must update the field 'quantity-delivered' in the Orderline. Please notice that only *short duration locks* (Gray and Reuter, 1993) are used, and, therefore, deadlock cannot occur. When the order form has been completed and the Orderlines asynchronously are confirmed by the servers of the remote store(s), the pivot subtransaction is executed at the mobile computer. If the credit limit of the customer is not violated, the pivot subtransaction marks the order as "committed", updates the balance of the customer and initiates a retrievable sub-subtransaction that update the central secondary copy of the customer record. Alternatively, the global transaction will be rejected or the salesman asked to reduce the amount of the balance in order to avoid violating the credit limit of the customer.

The amount of an Orderline can be reduced by executing a retrievable subtransaction that reduces the quantity ordered in the Orderline. For each reduced Orderline a retrievable sub-subtransaction increases the corresponding remote stock. Finally, the salesman can retry to execute the pivot subtransaction.

In the example above, it is possible to integrate the application into an ERP (Enterprise Resource Planning) system such as SAP R/3, Baan, Concorde Axapta, Navision, etc. We have analyzed how one of these major software companies can design a distributed version of their financial management and accounting system by using our transaction model. The Software Company has decided to implement a prototype of the enterprise-wide financial management and accounting system by using our transaction model. This is the first step in using our transaction model in mobile computing.

4.2 A Transaction Shell for Atomicity Implementation

In this section, we will describe a general transaction shell that can simplify the atomicity implementation. In order to implement the atomicity property the transaction shell must follow the rules of our nested transaction model described in section 2. After the presentation of the transaction shell, we will illustrate how to use the transaction shell in implementing the application of the traveling salesman from Example 4.1.

Tables 1-3 illustrate how the rules are applied to the activities of a global committing transaction. The distributed algorithm illustrated in Table 1 is a shell without application logic. However, all the necessary database access types of a global transaction are described. A global transaction can consist of up to four different main activities corresponding to read only accesses and the updatings by using compensatable, pivot or retrievable subtransactions. The programs corresponding to these main activities may have more than one entry. Therefore, the main activities may be divided into sub-activities or steps, which are numbered by the activity number followed by a serial number. Normally, each step of an activity is executed in another location than the previous step. In Table 1, we distinguish between the following types of locations:

- The "root location" that normally is the PC of the user.
- The "log location" where the parameters of possible compensating subtransactions are stored.
- The "pivot location" is the location where the pivot subtransaction is executed.
- The "other locations" are any location with a database server.

Inconsistency may occur if a compensating subtransaction is executed more than once. In the

following, this inconsistency is managed by storing the identification of a compensatable subtransaction in the location where the subtransaction is executed. In other words, all compensatable subtransactions must store its identification and in order not to execute a compensating subtransaction more than once, the stored identification must be deleted by the corresponding compensating subtransaction. In order to implement the atomicity property it is important that all updating transactions use the

transaction shell of Table 1. Therefore, the global compensating transaction of Table 2 and the global recovery program of Table 3 also use the transaction shell of Table 1. The global recovery program of Table 3 should always be used in case the root transaction fails, because the recovery program will tell the user the state of the global transaction. In case the global compensating transaction or the global recovery program fail, they may be restarted.

The read-only steps:	
1.1	The root transaction calls a subtransaction that only reads records. (In this way, the root transaction receives the data base information that is necessary for the user making the input for the global update transaction).
1.2	In a server location, a read-only subtransaction reads records by using short duration locks. As records may be stored in any location, this step may be executed at any location.
1.3	After the root transaction has received all the answers from the servers, it sends the information to the user.
1.4	The root transaction receives input from the user in order to execute either: <ul style="list-style-type: none"> • “The read only steps” • “The compensatable steps” • “The pivot step” • “The global compensating transaction” of Table 2
The compensatable steps:	
2.1	In the “log location” a parameter record is created for each of the compensatable subtransactions. A parameter record is committed after its creation if the corresponding compensatable subtransaction is stored in another location. Otherwise, the parameter record is committed together with the corresponding compensatable subtransaction. After the creation of a parameter record, the corresponding compensatable subtransaction is called.
2.2	A compensatable subtransaction is executed and the identification of the compensatable subtransaction is stored as described earlier. As records may be stored in any location, this step may be executed at any location. Any possible compensatable sub-subtransactions may be called, and, therefore, this step may be repeated according to the number of levels in the nested compensatable subtransactions.
2.3	The answers (“committed”, “cannot commit” or “no answer”) of the compensatable subtransactions are returned to the root transaction and displayed to the user, who can restart the root transaction in step 1.4.
The pivot step:	
3	In the pivot location, the pivot subtransaction is executed, and the identification of the subtransaction is stored in order to make it possible to see whether the global transaction has been committed or not. Any update propagations for retrievable subtransactions must be initiated before the pivot subtransaction can be committed.
The retrievable step:	
4	The servers execute the retrievable subtransactions asynchronously, and any possible retrievable sub-subtransactions may be initiated. This step may be repeated according to the number of levels in the nested retrievable subtransactions.

Table 1.

The read-only steps:	
1.1	The root transaction makes calls for reading the identification of the pivot subtransaction.
1.2	In the pivot location, a subtransaction tries to find the stored identification of the pivot subtransaction indicating that the global transaction has been committed.

1.3	If the pivot subtransaction has been committed, the restart is finished, because the global transaction has already been committed and the message sent to the user. Otherwise, step 1.1 is repeated as illustrated in the next step.
1.1	The root transaction calls a subtransaction in the log location for reading the parameter records of the compensating subtransactions.
1.2	In the log location a subtransaction tries to read possible parameter records corresponding to compensatable subtransactions that may be committed.
1.3	If no parameter records exist, the root transaction tells the user that the global transaction is aborted.
1.4	Otherwise, the root transaction calls the abort-pivot subtransaction.
The pivot step:	
3	The abort-pivot subtransaction is executed by initiating the UPs of the compensating subtransactions corresponding to the parameter records. The abort-pivot subtransaction may be executed in any location having DBMS where the UPs of the retrievable compensating subtransactions can be initiated.
The retrievable step:	
4	The servers execute asynchronously the retrievable compensating subtransactions if the identification of the corresponding compensatable subtransactions is stored. Any possible retrievable compensating sub-subtransactions may be initiated. This step may be repeated according to the number of levels in the nested retrievable compensating subtransactions.

Table 2: General activities for a global compensating transaction

The read-only steps:	
1.1	The root transaction makes calls for reading the identification of the pivot subtransaction.
1.2	In the pivot location a subtransaction tries to find the stored identification of the pivot subtransaction that indicates that the global transaction has been committed.
1.3	If the pivot subtransaction has been committed, the restart is finished, because the global transaction has already been committed and the message sent to the user. Otherwise, step 1.1 is repeated as illustrated in the next step.
1.1	The root transaction calls a subtransaction in the log location for reading the parameter records of the compensating subtransactions.
1.2	In the log location a subtransaction tries to read possible parameter records corresponding to compensatable subtransactions that may be committed.
1.3	The parameter records are sent to the user. (The user can then restart the global transaction in step 1.4 of Table 1).

Table 3: General activities for a recovery program

Example 4.2

In this example, we will use the transaction shell of Table 1 to illustrate how to implement the global invoicing transaction from Example 4.1. The root transaction is executed at the salesman's mobile PC. In this example, we also choose the server of the mobile salesman as the "log location" and the pivot location.

In steps 1.1-1.3, the offers of the seller are read in "the read only steps". When the root transaction in step 1.4 receives information to start a new global transaction it enters "the compensatable steps" and creates a compensatable order record in the mobile computer. In this example, we assume the Orderlines are confirmed by the salesman one by one. Therefore, "the compensatable steps" are executed once for each Orderline. After a

compensatable Orderline record has been created, a compensatable sub-subtransaction updates the remote Product-stock asynchronously. In this application, the log information with the parameters of the compensating subtransactions should be implemented as fields in the Customer, Order and Orderline records respectively, because this will reduce the number of updatings. If the salesman wants to correct errors, it is possible for the salesman to delete an existing Orderline by using the pivot and retrievable steps without committing the global transaction. That is, the compensating subtransaction of an Orderline is initiated in the pivot step and executed in the retrievable step. As the compensatable subtransactions are nested, the corresponding retrievable compensating subtransactions must be nested, too.

When the order form has been completed and the Orderlines confirmed, the salesman makes input to step 1.4 that starts “the pivot step, where the balance of the customer is updated, and the retrievable messages are initiated.

In case the root transaction fails, the recovery program of Table 3 can ensure recovery in the following way:

In “the read only steps” the recovery program reads all the log information, i.e. the Customer, Order and Order-line records. If the pivot subtransaction has updated the balance of the Customer record, the global transaction has already been committed. Otherwise, the compensatable Orderlines are displayed for the salesman and control is returned to him allowing him to restart the root transaction in step 1.4.

5. Conclusions

In this paper we have described in detail a new nested transaction shell (algorithm) designed for updating multidatabases with semantic ACID properties and we have illustrated how to use the transaction shell in mobile computing used by a traveling salesman.

References

Berenson, Hal and Phil Bernstein, Jim Gray, Jim Melton, Elizabeth O’Neil, Patrick O’Neil (1995). A Critique of ANSI SQL Isolation Levels, Proc ACM SIGMOD Conf., pp 1-10.

Elmagarmid, A. (ed.) (1992), *Database Transaction Models for Advanced Applications*, Morgan Kaufmann.

Frank, L. and Torben Zahle (1998), Semantic ACID Properties in Multidatabases Using Remote Procedure Calls and Update Propagations, Software - Practice & Experience, Vol.28.

Frank, L. (1998), ‘Evaluation Overview of the Replication Methods for High Availability Databases, Proceedings of Object-Oriented Technology, ECOOP’98 Workshop Reader, Springer-Verlag, pp 321-322.

Frank, L. (1999a), ‘Atomicity Implementation in Multidatabases with High Performance and Availability’, *Proc of the 2nd International Symposium on Cooperative Database Systems (CODAS’99)*, Springer-Verlag, pp 103-114.

Frank, L. (1999b), ‘Integration of Different Commit/Isolation Protocols in CSCW Systems with Shared Data’, *Andrei Ershov Third International*

Conference, Perspectives of System Informatics, Going to be published by Springer-Verlag.

Frank, L. (1999c), ‘Evaluation of the Basic Remote Backup and Replication Methods for High Availability Databases’, *Technical Report*, Department of Informatics, Copenhagen Business School.

Gallersdörfer, R. and Matthias Nicola (1995), Improving Performance in Replicated Databases through Relaxed Coherency, *Proc 21st VLDB Conf*, pp 445-455.

Garcia-Molina, H. and K. Salem (1987), ‘Sagas’, *ACM SIGMOD Conf*, pp 249-259.

Garcia-Molina, H., and Polyzois, C. A. (1990). Issues in disaster recovery. In IEEE Compcon. IEEE, New York, pp 573-577.

Gray, Jim and Andreas Reuter (1993). *Transaction Processing*, Morgan Kaufman.

Humborstad, Rune and M. Sabaratnam, S. Hvasshovd, O. Torbjornsen (1997). 1-Safe algorithms for symmetric site configurations, *Proc 23th VLDB Conf*, pp 316-325.

Mehrotra S., R. Rastogi, H. Korth and A. Silberschatz (1992), ‘A transaction model for multi-database systems’, *Proc International Conference on Distributed Computing Systems*, pp 56-63.

Polyzois, C. and Hector Garcia-Molina (1994). Evaluation of Remote Backup Algorithms for Transaction-Processing Systems, *ACM TODS*, 19(3), pp 423-449.

Weikum, G. and H. J. Schek (1992), ‘Concepts and Applications of Multilevel Transactions and Open Nested Transactions’, in: *Elmagarmid (1992)*, pp 515-553.

Zhang, A., M. Nodine, B. Bhargava and O. Bukhres (1994), ‘Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems’, *Proc ACM SIGMOD Conf*, pp 67-78.