Mobile Transaction Supports for DBMS: An Overview

Patricia Serrano-Alvarado[*], Claudia Roncancio, Michel Adiba
LSR-IMAG Laboratory
BP 72, 38402 St-Martin d'Hères, France
e-mail: Firstname.Lastname@imag.fr

Abstract

In recent years data management in mobile environments has generated a great interest. Several proposals concerning mobile transactions have been done, however, it is very difficult to have an overview of all these approaches. In this paper we analyze and compare some proposals, we focus on the effect of mobile transactions on the ACID properties and on the execution model. In addition, we analyze approaches that deal particularly with the mobile nature of mobile hosts; in these proposals the ACID properties are not compromised because transactions are executed at multidatabase systems on the wired network. Based on our analysis, we introduce our ongoing research: the definition, design and implementation of a Mobile Transaction Service.

Keywords: Mobile transactions, databases, mobility, atomicity, consistency, isolation, durability, commit processing

1. Introduction

Distributed information systems are evolving in several directions which generate new challenges. Advances in computer and network technologies have made mobile computing a reality but generate new kinds of problems [10], due, for instance, to the mobile nature of mobile clients and to frequent disconnections.

Data management in mobile environments is gaining a great attention today with the emergence of mobile computing environments. To that extent, database system architecture should be revisited [17]. Concerning arising data management problems, solutions in distinct areas have been proposed [32] [18] [25]. The notion of

[*]Supported by the CONACyT scholarship program of the Mexican Government

transaction has also been revisited and several models have been introduced. These works indicate how current transaction technologies are not suitable for mobile information systems and propose different extensions which are difficult to compare [4].

In this paper we propose a deep analysis and comparison of previous mobile transaction proposals. The literature on the subject is important and some attempts to analyze proposed models have been made [11][19]. However, we think that it is necessary to make an extensive analytical comparison of these models. Additionally, we identify relevant issues that influence the construction of a Mobile Transaction Service (MTS). Thus, the last part of the paper concerns our ongoing research which is the definition, design and implementation of a MTS.

We are considering a mobile computing environment with a network consisting of stationary and mobile hosts (SH, MH). Shared data are distributed over several database servers executing on SH. Mobile hosts could be of different nature ranging from PDA to personal computers. Each MH acts as a client and initiates queries/updates considered as transaction operations. Here, we make no specific hypothesis about the database model (relational, object) or the DBMS type which are used. We consider only that the database deals with a collection of shared data and transactions operate on them. An MH changes its location and network connections while transactions are being processed. While in motion, an MH retains its network connections through the support of SH which act as Base Stations (BS). Both MH and BS may have storage capabilities and DBMS modules for running operations on behalf of a given transaction. However, MH are assumed to have limited storage capacity (cached data) and power (battery life).

Informally, *a transaction is a set of operations that translate a database from a consistent state into another consistent state*. Transaction managers offer ACID properties by implementing commit protocols, obtaining serializable executions, controlling visibility of non-committed transactions, supporting recovery, etc. Although, very often ACID properties are not appropriate and several models relaxing these properties have been proposed [22][21][26].

In the context of mobile computing, there exist several interpretations of mobile transactions. For us, *a mobile transaction is a transaction where at least one mobile host takes part in its execution.* In any case, the participation of an MH introduces dimensions inherent to mobility such as: movement, disconnections and variations on the quality of the communication. As we will see in the following, transaction managers (TM) supporting mobile transactions have to adapt their functionalities to deal with these dimensions.

In the scope of this paper we will focus on systems with a client-server architecture where clients are MH having stockage/processing capacities and where the server is on the wired network. The server provides resources and transaction management. We consider the system in a *connected mode* if the MH and the server are connected; otherwise it is in a *disconnected mode.* Whenever we use the term “local”, as in local transactions and local processing, we refer to MH.

Section 2 presents a survey of analyzed proposals and their execution models. In section 3, the influence of mobile transactions on ACID properties is analyzed and compared. Proposals that deal particularly with mobility and disconnection are analyzed in section 4. Section 5, discusses issues on the definition of a Mobile Transaction Service, our ongoing research. Finally, section 6 concludes the paper.

2. Mobile transaction survey

In this section we begin our analysis by introducing each analyzed model with a brief overview. In section 2.2 we analyze and compare their respective execution model; we also identify their transaction types and their principal characteristics.

2.1. Proposals survey

Clustering proposal [23] [24] assumes a fully distributed system and is designed to maintain database consistency. The database is dynamically divided into clusters, each one groups together semantically related or closely located data. A cluster may be distributed on several strongly connected hosts. When an MH is disconnected it becomes a cluster by itself. For every object two copies are maintained, one of them (*strict version*) must be globally consistent, and the other (*weak version*) can tolerate some degree of inconsistency but must be locally consistent. MT are either *strict* or *weak*. Weak transactions access only weak versions whereas strict ones access strict versions.

Two-tier replication [16] considers both transaction and replication approaches for mobile environments where MH are occasionally connected. A master version for each data and several replicated versions (copies) exist. Two types of transactions are supported: *base* and *tentative transactions*. Base transactions are executed accessing master versions (lazy-master replication scheme) whereas tentative transactions are executed accessing tentative versions (local copies). Tentative transactions may perform updates on the MH in a disconnected mode. When the connection is established the tentative transactions are re-executed as base transactions.

Pro-motion [30][29] is a mobile transaction processing system that supports disconnected mode. *Compacts* are introduced to allow local executions on MH. Necessary information to manage the compact is encapsulated in it. To improve autonomy and to increase concurrency, object semantics are used in the construction of compacts whenever possible. Compacts are the basic unit of caching and control.

Reporting [4] analyzes nested transactions [22] and open-nested transactions (such as sagas [21], split transactions [26] and multitransactions [27]) showing their limitations for mobile environments. [4] considers a mobile database environment as a special multidatabase system with specific requirements, where transactions on MH are considered as a set of subtransactions. They propose an open-nested transaction model that supports *atomic*, *non-compensatable transactions* and two additional types: *reporting* and *co-transactions* [3][6]. While in execution, transactions can share their partial results and partially maintain the state of a mobile subtransaction (executed on the MH) on an BS.

Semantics-based [5] focuses on the use of object semantics information to improve the MH autonomy in disconnected mode. This contribution concentrates on *object fragmentation* as a solution to concurrent operations and to limitations of MH storage capacity. This approach uses objects organization and application semantics to split large and complex data into smaller manageable fragments of the same type. Each fragment can be cached independently and manipulated asynchronously. Fragmentable objects can be aggregate items, sets, stacks and queues.

Prewrite [20] tries to increase data availability on MH by introducing a *prewrite* operation in addition to standard writes. A prewrite makes data value visible at *precommit* before the commit of the mobile transaction. Permanent updates on the database are performed later by the write operation at commitment. Two variants of data are maintained: *prewrite* and *write*. Prewrite variant reflects future data state but may be structurally slightly different from the corresponding write value e.g., in an object of type document the prewrite is an abstract and the write is the complete document.

KT [11] (Kangaroo Transactions) proposes a mobile transaction model that focuses on the MH movement during the execution of transactions. Mobile transactions are generated at MH and are entirely executed at a Multi-database System (MDBS) on the wired network. KT proposes to implement a Data Access Agent (DAA) on top of existing Global Transaction Managers (GTM). This agent will be placed at all BS and will manage mobile transactions and the movement of MH.

MDSTPM [31] (Multidatabase Transaction Processing Manager architecture) proposes a framework to support transaction submissions from MH in a multidatabase environment. The contribution concerning MH disconnections is the implementation of the Message and Queuing Facility (MQF) that manages the message interchange between MH and the wired multidatabase system. An MDSTPM is assumed at each host (MH/SH) on top of existing local DBMS. Local processing is the responsibility of the local DBMS. The MDSTPM coordinates the execution of global transactions, it generates scheduling and coordinates commitments.

2.2. Execution model

In Clustering, strict transactions are executed when hosts are strongly connected and weak transactions when MH are disconnected. Two kinds of operations are introduced *weak reads* and *weak writes*. Strict transactions

Proposal	**Transaction type**	**MT request**	**Execution at MH**	**Execution at wired network**
Clustering	Strict and weak transactions	MH	Weak transactions and *local commit* in disconnected mode. Participation in the execution of strict transactions in connected mode	Strict transactions and *commit* of weak transactions (synchronization, permanents updates)
Two-tier replication	Base and tentative transactions	MH	Tentative transactions in disconnected mode. Participation in the execution of base transactions in connected mode	Base transactions
Promotion	Long-lived nested-split transactions	MH	The compact agent executes entirely the MT and makes *local commit*	The compact manager is in charge of compact construction, *commit* of *locally committed* transactions (synchronization, permanents updates)
Reporting	Open-nested transactions with atomic, non-compensatable, reporting and co-transactions	MH/SH	Subtransactions and even global transactions	Global transactions and subtransactions
Semantics-based	Long-lived transactions	MH	MT and *local commit*	In answer to MH requests, objects fragmentation (split) is made by the database server and also updates reintegration (merge)
Prewrite	Long-lived (nested, split) transactions	MH	MT and *local commit*	Lock management and *commit* of *locally committed* transactions (write operations)
KT	Open-nested and split transactions	MH		Coordination and execution of the entire transaction
MDSTPM	Multitransactions and local transactions	MH	Local transactions	Coordination and execution of multitransactions

Table 1. Summary of execution models

contain standard reads and writes (strict operations), whereas weak transactions contain weak operations. When reconnection is possible (or when application consistency requires it) a synchronization process, executed on the database server, allows the database to be globally consistent.

Two-tier replication uses two types of atomic transactions (base and tentative) that differ in the version of data they access. After a disconnected execution, the BS will re-execute tentative transactions as base transactions to reach global consistency. This re-execution is the way to make local updates persistent.

Both, Clustering and Two-tier replication require a transaction manager on the MH to provide local transaction execution, concurrency control, log management and recovery.

Pro-motion uses nested-split transactions [4] [27] as its infrastructure. It considers the entire mobile system as one extremely large long-lived transaction executed on the server. Resources needed to create compacts are obtained by this transaction through usual database operations. Compacts construction is responsability for the *compact manager* at the database server. The management of compacts is performed by a *compact manager*, a *compact agent* at the MH and a *mobility manager* at the BS. The *compact manager* will act as a front-end for the database server and appears to be an ordinary database client executing a single, large long-lived transaction. On each MH, the *compact agent* is responsible for cache management as well as for transaction processing, concurrency control, logging and recovery. The *mobility manager* is in charge of transmissions between agents. MH transactions are executed locally even in connected mode. A synchronization process is executed by the compact agent and the compact manager at reconnection. This process checks compacts modified by locally committed transactions. If the compacts preserve global consistency, then a global commit is performed.

In Reporting, a mobile transaction is structured as a set of transactions, some of which are executed on the MH. They consider that limitations on MH make necessary the use of SH e.g., to store part of the state of the computation or to perform part of the computation. Open-nested transactions with subtransactions of the following four types are proposed: *atomic transactions* have standard abort and commit properties. *Non-compensatable transactions* at commit time delegate to their parent all operations they have invoked. *Reporting transactions* report to another transaction some of their results at any point during execution. A report can be considered as a *delegation of state* between transactions. *Co-transactions* are reporting transactions where control is passed from the reporting transaction to the one that receives the report. Co-transactions are suspended at the time of delegation and they resume their execution when they receive a report.

In Semantics-based, mobile transactions are invoked at the MH, and for the database server point of view they are long-lived because of communication delays. No assumptions are made about the transaction structure. MH fragment request includes two parameters: *selection criteria* and *consistency conditions*. The selection criteria indicates data to be cached on the MH and the required fragment size. The consistency conditions specify constraints to preserve consistency on the entire data. Data fragmentation executed on the sever allows fine-grain concurrency control. Exclusive master copies of fragments are given to the MH and transactions can be entirely executed on it. A reconciliation process is executed by the server when reconnection occurs. This model may be used with different transaction types.

In Prewrite, the main idea is to divide the transaction execution between the MH and the database server. The TM on the MH executes the transaction, but permanent updates are made at the database server by a data manager (DM). Prewrite ensures that, by delegating the responsibility of write at the database, transaction processing is reduced on the MH. Three operations (*prereads, prewrites* and *precommit*) that will be executed by the TM are proposed. Ordinary reads and permanent writes are made by the DM. The BS has logging capacities and maintains close relationship with the DM. The transaction execution is divided in two parts, first, the TM requests to the BS necessary locks. The BS acquires locks from the DM. When the TM finishes the transaction by a local commit (precommit), prewrites are sent to the BS. In the second part, the DM makes prewrites permanent and commits the mobile transaction. This model considers that mobile transactions are long-lived and implementation can be made with nested and split transactions.

In KT, preserving the ACID properties is the responsibility for each DBMS. The transaction model is built using concepts of open-nested [14] and split transactions [26]. The mobile transaction execution (actually a global transaction) is coordinated by the BS to which the MH is currently assigned. When one MH hops from a cell to another (consequently from BS to BS) the coordination of the mobile transaction also moves. This mobility is captured by splitting the original transaction into two transactions (called Joeys transactions, there exist one Joey transaction per BS). The split only concerns the coordination of the transaction. Thus, if the MH hops from BS-1 to BS-2, BS-1 will just coordinate the operations that were executed during the stay of the MH in the BS-1 cell.

In MDSTPM, as in KT, the manner ACID properties are enforced depends on each DBMS at each site. Each MDSTPM is responsible for coordinating its global transactions. For MH, because of disconnections, a SH coordinator is designated in advance. Therefore, once a MH submits a global transaction, it may disconnects and performs some other tasks without having to wait for the mobile transaction to commit. The coordinator host will manage the mobile transaction on behalf of the MH.

All proposals but Reporting assume that mobile transactions are requested from MH. In Reporting, transactions can be requested by any host.

In table 1 we summarize all the execution models and their principal characteristics. We recall that local transactions are transactions executed at MH. MT makes reference to mobile transactions. Whenever there is an empty cell in the table that means that we have not enough information to fill it.

3. Analysis of ACID properties

We consider that it is essential to know how mobile transactions deal with the ACID properties and how they are executed. In 3.1, 3.2, 3.3 and 3.4 we compare the models and identify common points regarding ACID properties. In this part of the analysis KT and MDSTPM are not included because they do not propose new solutions with respect to ACID properties. These proposals (analyzed in section 4) are oriented toward managing movement and disconnection properties. They assume that transactions will be executed by MDBS on the wired network.

3.1. Atomicity

Except for Reporting and Semantics-based, transaction validation is done in two steps. The first one is realized on MH (local commit) and the second one (commit) at the BS/Database server. Clustering, Two-tier replication, Pro-motion and Prewrite execute local commit, each one with specific characteristics:

- Clustering and Two-tier replication make local commit only in disconnected mode using special transaction types. In connected mode an atomic commit protocol is used (e.g., two phase commit) and it includes participation of several clusters/hosts.
- Pro-motion and Prewrite do not differentiate connected and disconnected mode. Local commit is performed using an atomic commit protocol.

At the second step of the validation process, locally committed transactions execute commit to make updates permanent on the database server. Transaction commitment can involve reconciliation mechanisms or transactions re-execution.

- Reconciliation in Clustering is made syntactically where weak transactions are aborted or rolled back if their weak writes conflict with strict transactions.
- In Two-tier replication, if base transactions (when they are the re-execution of tentative transactions) fail, even by taking into account the *acceptance criteria* (attached to each tentative transaction), then the tentative transactions are aborted.
- In Pro-motion, compacts involved in locally committed transactions are checked. If some compacts are no more valid, then mobile transactions are aborted and a *contingency procedure* (attached to each local commit) is executed to obtain semantic atomicity.
- In Prewrite, neither reconciliation nor re-execution are made. By the transaction processing algorithm and locking protocol, Prewrite ensures that locally committed transactions will commit at the database server.

The approach is different in Reporting where each subtransaction is atomic but this does not prove the atomicity of the global mobile transaction. Except for*non-compensatable subtransactions*, compensatable transactions can be associated to subtransactions so (semantic) atomicity is guaranteed. In *Non-compensatable transactions*, *reporting* and *co-transaction* delegation does not affect atomicity because it does not require the invoking transaction of an operation to be the transaction who either commits or aborts the operation. A transaction is *quasi atomic* if all operations that it is *responsible* for are committed or none at all.

In Semantics-based, transactions are considered long-lived. As MH are responsible of local transaction commit it would be possible to support atomic or not atomic transactions.

Conceptually, Semantics-based, Pro-motion, Prewrite and Reporting consider transactions as long-lived ones. If these transactions are executed on multidatabase systems, global atomicity depends on the autonomy of each database system [2]. If some DBMS cannot participate in a global atomic commit protocol, then atomicity is hard to be guaranteed.

Cascading aborts may occur in Clustering, Two-tier replication and Pro-motion. Nevertheless, local committed transactions modify local data, consequently, only aborts of local transactions are provoked. In addition, these aborts concern only weak and tentative transactions because local results are exclusively available for these types of transactions.

Table 2 shows the validation process of compared proposals. We underline the fact that generally mobile transactions make validation in two steps, where *local commit* is done at MH and *commit* is done at the BS/Database server.

3.2. Consistency

Clustering and Two-tier replication maintain consistency of replicated data with two versions. Both versions are located on the MH, one of them (weak/tentative) is used to support data evolution in disconnected mode. The second one (strict/master) must always be consistent but sometimes it will contain old versions (in disconnected mode). Consistency in strict/master versions is preserved using one-copy serializability methods e.g., quorum consensus, master copy. Some particularities are:

- In Clustering, semantic information is used to specify the degree of inconsistency for weak versions. This degree may be bounded limiting the number of local commits, the number of transactions that can operate on inconsistent copies, the number of copies that can diverge, etc. There exist also a *function h* that controls this degree by projecting strict operations on weak versions. Full consistency is achieved by merging different copies of the same data located at different clusters (reconciliation).

- In Two-tier replication, tentative data versions are discarded at reconnection since they are completely refreshed from master versions.

Proposal	**Validation process**	
	First step at MH	**Second step at BS/DB server**
Clustering	Disconnected mode: *local commit* of weak transactions. Connected mode: 2PC for strict transactions	*Commit* involves syntactic reconciliation with abortion and rollback in solution conflict
Two-tier replication	Disconnected mode: *local commit* of tentative transactions. Connected mode: atomic commit protocol for base transactions	Tentative transactions are re-executed taking into account their acceptance criterion
Promotion	*local commit* of all local transactions	A synchronization process checks compacts involved in local transactions. In case of conflicts, local transactions are aborted and contingency procedures are executed
Prewrite	*local commit* of all local transactions	Local updates are made permanents by the write operations
Semantics-based	*local commit*	Updates reintegration (merge). As fragments are exclusive copies and they have attached consistency conditions there not exist conflicts in reintegration.
Reporting	All subtransactions are atomic and they are able to commit independently of the parent transaction. Except for non-compensatable subtransactions, in case of abortion compensating transactions can be associated to subtransactions	

Table 2. Summary of validation process

It seems to us that weak/tentative transactions have drawbacks with respect to strict/base transactions, in the resynchronization process (reconciliation in Clustering and re-execution in Two-tier replication).

Pro-motion and Semantics-based exploit semantic information to construct compacts and fragments:

- For Pro-motion the *compact* represents an agreement between the database server and the MH. The compact manager and the database server encapsulate in compacts: *data, type specific methods, state information, consistency rules,* and *obligations.* If the compact agent and compact manager respect all these conditions, the use of compacts will not affect database consistency. The compact designer can determine correctness criteria and concurrency control methods per compact.

- In Semantics-based, to preserve consistency, objects must carefully support *split* (to make fragments) and *merge* (to reconciliate fragments) operations. Another restriction to preserve consistency is to provide *consistency conditions* (supplied by applications) on the entire object. These conditions include allowable operations, constraints of their input values and conditions on the object state.

In Reporting, new ways to achieve consistency are not proposed, but subtransactions can be related to compensating transactions (except for non-compensatable) in order to maintain semantic consistency in case of abortions.

Proposal	Underlying concepts	Use of semantic information
Clustering	2 versions of data: strict (one-copy serializability), weak (degrees of inconsistency, data evolution in disconnected mode)	Definition of the function h and degrees of inconsistency
Two-tier replication	2 versions of data: master (one-copy serializability), tentative (local data evolution in disconnected mode)	Acceptance criteria
Promotion	Compacts including type specific methods, consistency rules and obligations	Compacts construction and contingency procedures
Reporting	Multitransaction approach	Delegation, compensating transactions
Semantics-based	Objects fragmentation (consistency conditions and split/merge operations)	Fragmentation
Prewrite	Serializability is based on the local commit order of mobile transactions	Definition of data variants (prewrite/write)

Table 3. Summary of consistency properties

Prewrite assures that the transaction processing algorithm along with the lock-based protocol, produce only serializable histories. This serializability is based on the local commit order of mobile transactions.

It is important to notice that semantic information of objects is essential to guarantee consistency in mobile applications. All analyzed models exploit objects semantics in different ways. Clustering defines degrees of inconsistency based on the application semantics. Two-tier replication manages an acceptance criteria between tentative and base transactions. Pro-motion uses semantic information to construct compacts and Semantics-based to split objects. Reporting makes delegation based on semantic requirements, and Prewrite defines semantically identical data variants (prewrite/write objects).

Table 3 summarizes the main concepts used to preserve consistency. It is also emphasized the importance of semantic information to offer more flexibility in consistency support.

The next issue, isolation, is strongly related to consistency because *the execution of a transaction in isolation preserves database consistency.*

3.3. Isolation

Isolation is not strictly enforced by all proposals, some of them allow *visibility* of intermediate transaction results.

Clustering, Two-tier replication, Pro-motion and Semantics-based give visibility of local committed results to local transactions on the same MH. On the other hand, Prewrite at local commit makes the results public to all hosts. In Reporting, visibility is permitted in atomic, reporting and co-transactions but not in non-compensatable transactions. An atomic transaction can commit its execution even before the commit of its parent, and its modifications to the database become visible for others transactions. In reporting and co-transactions the objective is precisely to allow visibility of partial results while in execution.

Proposal	**Visibility**	**Concurrency control protocol**
Clustering	*local committed* transaction results are visible to local weak transactions on the same MH	2PL, 4 conflict tables and new lock types are proposed
Two-tier replication	*local committed* transaction results are visible to local tentative transactions on the same MH	Locking mechanisms
Promotion	*local committed* transaction results are visible to local transactions on the same MH	2PL
Reporting	with subtransactions *atomic, reporting and co-transactions* visibility is allowed before the commit of the global transaction	
Semantics-based	*local committed* transactions results are visible to local transactions on the same MH	2PL to control access to locally cached fragments
Prewrite	*local committed* transactions results are visible to all hosts	2PL extended, one conflict table and new lock types are proposed

Table 4. Summary of isolation aspects

Taking Pro-motion and Reporting as open-nested transactions, global isolation is not respected since subtransactions are not executed isolately. After the synchronization process, Pro-motion splits its long-lived transaction. All operations that have been successfully synchronized form a separate transaction that is committed on the database server. Results of this split (committed) transaction will be visible for all the database environment.

To manage isolation (restraint visibility) Clustering and Prewrite propose new conflict solution tables.

- Clustering uses strict two phase locking and proposes four lock types that correspond to weak and strict operations (WR, WW, SR, SW). Four conflict tables for lock compatibility are proposed. The projecting *function h* utilizes conflict tables to reflect strict operations on weak versions depending on the application consistency requirements. For example, strict consistency requires translating a strict write on an object into strict writes on all its copies (strict and weak ones). Consequently, a SW lock is non compatible with any other lock. Weak transactions release their locks at local commit and strict transactions at commit.

- As Clustering, Prewrite uses a two phase locking protocol and the conflict operation table includes preread and prewrite operations (PR, PW, R, W). As prewrite and preread locks are managed at TM level and read and write locks at DM level, there exist no conflict between prewrite/preread and write/read locks. To make prewrites permanent the prewrite lock must be converted into a write lock so that the DM can write and commit the mobile transaction. Preread locks are released at local commit time whereas prewrite/write/read locks at commit time.

In our opinion, Prewrite approach is interesting in applications using objects that can have two variants (write/prewrite value) as *design objects* (the prewrite represents a model of the design) or *document objects*. In these object types, prewrites are different from writes and availability is improved with two variations of the "same object". Otherwise, using simple objects prewrites are identical to writes and the algorithm behaves as using relaxed two phase locking.

Since in Pro-motion the compact designer can determine correctness criteria and concurrency control methods per compact, they propose to use a ten level scale. Levels are characterized based upon the degrees of isolation defined in the ANSI SQL standard as extended in [1]. Level 9 represents a serial execution of transactions and level 8 a serializable execution. Each succeeding level represents a lesser degree of isolation. At level 0 there is no guarantee about isolation. Because the arbitrary use of isolation levels can lead to inconsistencies, Pro-motion proposes simple rules:

1. Transactions impose a minimal level for write and read operations.
2. Each operation is associated to a level.
3. None of the write operation level is lower than the write level of the transaction.
4. None of the read operation level is lower than the read level of the transaction.
5. The lowest level of any read operation is greater than or equal to the highest level required by any write operation.

In Semantics-based, to ensure serializability, local transactions have access to cached fragments by conventional concurrency control protocols e.g. two phase locking.

In table 4, we remark the importance of visibility at local commit. Having local data availability conduces to some kind of autonomy, consequently, local process at MH will not be blocked when disconnection occurs. Moreover, the table makes evident that two phase locking (2PL) is the concurrency control protocol most utilized by the analyzed works.

3.4. Durability

Clustering, Two-tier replication and Pro-motion cannot guarantee durability before commit. Pro-motion with compacts can give some guarantees of durability, but they may exist conditions that could not be respected because of disconnections e.g., there is a deadline (in the compact) that could not be reached, consequently, durability is hard to obtain in the synchronization process. In Reporting, subtransactions are durable if the parent transaction commits. Semantics-based and Prewrite models guarantee durability since local commit. The first one reduces fragments availability because it can hold fragments by an undefined period of time. The second one uses many message exchanges to get locks from the BS. In the Prewrite algorithm, if a mobile transaction makes a local commit, it is sure to commit, Prewrite does not permit a local committed transaction to abort.

Proposal	**Durability guarantees**	**Drawbacks**
Clustering	Yes, after *commit* (resynchronization)	*Locally committed* transactions can be rolled back due to resynchronization conflicts
Two-tier replication	Yes, after *commit* (re-execution)	*Locally committed* transactions can be rolled back due to resynchronization conflicts during re-execution
Promotion	Yes, after *commit* (resynchronization)	*Locally committed* transactions can be rolled back due to resynchronization conflicts
Reporting	Yes, if the parent transaction *commits*, subtransactions are durable	
Semantics-based	Yes, after *local commit*	Reduction of fragments availability at database server
Prewrite	Yes, after *local commit*	Many message exchanges between MH and BS

Table 5. Summary of durability property

Note that logging issues are not discussed here. It seems that in the proposals we have analyzed these issues are not clearly studied, but we are currently investigating these aspects.

Table 5 shows the moment when durability is insured and some drawbacks.

4. Movement and disconnection analysis

In this section we are concentrated on movement and disconnection issues. Previous analyzed proposals do not give details about management of MH mobility. Only Pro-motion includes in its architecture a *mobility manager* that is in charge of communication between the MH and the database server; but there is no details about its functioning. Therefore, in this section we propose a complementary analysis for section3.

As we mentioned before, in KT and MDSTPM the ACID properties are not affected by mobility because transaction execution is the responsibility for DBMS located at SH. Although, as transactions are requested from MH, mobility and disconnection must be managed.

In KT, to support MH mobility and disconnection, the Data Access Agent (DAA) tracks MH movement by maintaining a linked list of all the BS that have been coordinators of the mobile transaction. This list will be used in case of cascading aborts. There exist also structures (*transaction status table* and *local log*) that store information of mobile transactions like: global transaction ID, status (active, commit, abort), Joey transaction ID, subtransactions that are included in the Joey transaction, compensating transactions (if they exist), etc.

MDSTPM The principal idea of Message and Queuing Facility (MQF) is an asynchronous message interchange, where messages are of types: Request, Acknowledgment, and Information. With MQF the MH can submit global transactions and switch to disconnected mode. In MH and coordinator hosts there exist tables and logs that record the overall state of the MH as well as information on global transactions (*Message Queue, Transactions Queue, Global Log, Global Transaction Table, Site Status Table*). At any moment, the MH can request information about its global transactions.

Both approaches are very similar, they propose to add a layer in existing multidatabase architectures to manage transactions requested by MH. The main difference is that in MDSTPM the coordination of the mobile transaction execution is centralized, that means that the SH coordinator is fixed in advance and it will not change during the transaction execution. Unlike MDSTPM, in KT, the coordination is distributed along all the BS that the MH visit. Hence we note that KT deals with the mobile nature of MH, not only with respect to disconnections. Distributed coordination reduces communication cost during execution, however, in case of cascading aborts communication is highly incremented. In contrast, with a centralized coordination as in MDSTPM, cascading aborts will be easer and cheaper, however, in case of high mobility, communication will be expensive.

To manage global transactions MDSTPM implements strict two phase locking for concurrency control and two phase commit for atomic commitment. We consider that if MDSTPM contemplates transaction execution at MH, these two mechanisms are not suitable because they lead to many message exchanges (MH with coordinator) and to undefined locking time of data (because of disconnections).

In [13] it can be found a good analysis about the impact of mobility on transactions requested from MH and executed at DBMS on the wired network. They analyzed three possible approaches for transaction coordination: (1) fixed at the MH, (2) fixed at a centralized site, (3) moving from BS to BS. In other respects, [12] proposes a mobile transaction definition dedicated to Location Dependent Data (LDD). They analyzed the impact of mobility on LDD and their effect on the ACID properties.

5. Towards a Mobile Transaction Service

The NODS project (Network Open Database Services) aims at defining an open, adaptable architecture that can be extended and customized on a per-application basis [7]. Our approach is characterized by a service oriented view of database functionality [9]. All DBMS and related tasks are unbundled into services (e.g. a persistence service) and applications use services as needed. In the design of services, particular attention is payed on their adaptability.

The Mobile Transaction Service (MTS), we are working on, provides support to mobile transactions and will cooperate with other services such as the replication, persistence [15] and event services [8] [28]. This section discusses some important issues about the MTS definition.

Overall functionalities As we have seen in previous sections, mobile transaction support includes standard TM functions, extra functions and particular implementations.

The most important and new feature the MTS must support is mobility management. This includes MH movements and disconnections.

Concerning function implementation, we have already identified two important points that has to be modified because of mobility: transaction validation process and consistency management. We consider that it is crucial to perform transaction validation in two steps that corresponds to *local commit* and *commit* introduced in 3.1. Disconnections make consistency management more complicated. It is necessary to adopt particular concurrency control protocols and synchronization process to offer some kind of serializability.

Transaction Processing Scenarios Considering the context introduced in section 1, the execution of mobile transactions may be performed in accordance with one of the following scenarios:

1. Mobile transaction is entirely executed on the database server.

2. Mobile transaction processing is distributed between the MH and the database server.

3. Mobile transaction is executed on the MH.

Each one of these execution strategies has special characteristics and demands particular capabilities. In (1), it is necessary the MTS provides mobility management (utilizing some techniques like in KT or MDSTPM). That execution can be "traditional" but the MTS should be aware of MH position and connectivity state to deliver results.

In (2), besides mobility management, MTS must be able to perform distributed executions where participants could not communicate during executions. In this scenario, a two steps validation process would be appropriated. Further, consistency must be guaranteed with special concurrency control protocols and synchronization methods.

In (3), MTS should ensure global consistency comprising MH updates. The MH has some freedom to manage the data locally but updates have to be incorporated in the database server. As in (2), concurrency control and synchronization methods must be adapted.

Consistency and Durability We emphasize the importance of avoiding application blocking at MH in disconnected mode. To achieve this goal, local availability of consistent objects is necessary. As we have noticed, semantic approaches are well adapted to manage consistency in mobile context. Moreover, local commit is necessary to obtain visibility of transaction results that are not already committed at the database server (in disconnected mode). Another important property to consider is durability of mobile transactions. Frequently, at resynchronization time, local committed mobile transactions have lower priority than non-mobile transactions, for mobile applications this is a great disadvantage. It is important to remark, that due to all changes introduced by mobility, also logging has to be adapted. Logging increases in importance because in addition to recovery purposes it is utilized to perform synchronization processes.

Transaction model In our opinion, the support of one single transaction type is not enough for mobile environments. Different transaction types are needed depending on the execution strategy. For example we could use for (1) flat transactions, for (2) open-nested transactions and for (3) long-lived transactions. Open-nested transactions by their structure can support some kind of local commit (allowing data evolution, application blocking is reduced) and parallel processing (when execution is distributed between the MH and the MTS). Long-lived transactions are ad-hoc for the third execution strategy because of undefined disconnection time. The long-live transaction could be a simple transaction or an open-nested one.

Architecture The analyzed models showed that BS can be a significant support for the MTS. Besides establishing connection with MH, the BS can have server capabilities as logging, data caching, resynchronization process, concurrency control mechanisms, etc. Delegating functionalities to the BS allows the MTS to save communication costs and to improve response time because the MH is closer to the BS than to the MTS. Consequently, for the MTS we will consider a three-tier architecture as client/agent/server, where the client is the MH, the agent is on the BS and the server on the MTS. This architecture and the specification of a prototype environment are part of our future work.

6. Conclusions

In this paper we analyzed different proposals that deal with mobile transactions. We organized our analysis in three parts, in the first one, we examined and compared the execution models. In the second part, we discussed the way ACID properties are preserved, pointing out common features and proposing summary tables. In the last part, we considered proposals oriented to MH movement and disconnection issues. In these proposals the ACID properties are not compromised because the transaction execution is made at MDBS on the wired network. In addition, we discussed the design of a Mobile Transaction Service which is the subject of our ongoing research.

References

[1] H. Berenson, P. Bernstein, and J. Gray et al. A Critique of ANSI SQL Isolation Levels. *SIGMOD (ACM Special Interest Group on Management of Data)*, 2(24):1–10, May 1995.

[2] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of Multidatabase Transaction Management. In *VLDB*, October 1992.

[3] P. K. Chrysanthis. *ACTA, A Framework for Modeling and Reasoning about Extended Transactions.* PhD thesis, Departament of Computer and Information Science, University of Massachusetts, Amherst, September 1991.

[4] P. K. Chrysanthis. Transaction Processing in a Mobile Computing Environment. In *Workshop on Advances in Parallel and Distributed Systems*, pages 77–82. IEEE, October 1993.

[5] P. K. Chrysanthis. Supporting Semantics Based Transaction Processing in Mobile Database Applications. *14th IEEE Symp on Reliable Distributed Systems*, September 1995.

[6] P. K. Chrysanthis and K. Ramamritham. Synthesis of Extended Transaction Models Using ACTA. Technical Report 93-05, University of Pittisburg, 1993.

[7] C. Collet. The NODS project : Networked Open Database Services. *ECOOP*, 2000.

[8] C. Collet, G. Vargas-Solar, and H. Grazziotin-Ribeiro. Open Active Services for Data-Intensive Distributed Applications. In *IDEAS*, Yokahama-Japan, September 2000.

[9] K. R. Ditrich and A. Geppert. *Component Database Systems.* Morgan Kaufmann Publishers, 2001.

[10] M. H. Dunham and Abdelsalam Helal. Mobile Computing and Databases: Anything New? *ACM SIGMOD Record*, 4(4), December 1995.

[11] M. H. Dunham and Abdelsalam Helal. A Mobile Transaction Model that Captures Both the Data and the Movement Behavior. *ACM/Baltzer Journal on special topics in mobile networks and applications*, 2:149–162, 1997.

[12] M. H. Dunham and Vijay Kumar. Defining Location Data Dependency, Transaction Mobility and Commitment. Technical Report 98-CSE-01, Southern Methodist University, Dallas, February 1998.

[13] M. H. Dunham and Vijay Kumar. Impact of Mobility on Transaction Management. In *Proceedings of the international workshop on data engineering for wireless and mobile access*, pages 14–21. SIGMOBILE, August 1999.

[14] A. K. Elmagarmid. *Database Transaction Models for Advanced Applications.* Morgan Kaufmann, 1992.

[15] L. García-Bañuelos and C. Collet. Towards an Adaptable Persistence Service: The NODS Approach. *TOOLS 2001 Workshop on Object-Oriented Databases*, March 2001.

[16] J. N. Gray, P. Helland, P.O'Neil, and D. Shasha. The Dangers of Replication and a Solution. In *Conference on Management of Data*, pages 173–182, Canada, June 1996.

[17] J. Jing, A.S. Helal, and A. Elmagarmid. Client-Server Computing in Mobile Environments. *ACM Computing Surveys*, 31(2), 1999.

[18] G. Jomier and A. Doucet, editors. *Bases de Données.* Hermes Science Publications, to appear on June 2001.

[19] S. K. Madria. Transaction Models for Mobile Computing. *15th IEEE International Conference on Distributed Computing Systems*, June 1995.

[20] S. K. Madria and B. Bhargava. A Transaction Model for Improving Data Availability in Mobile Computing. *Distributed and Parallel Databases*, 2001.

[21] H. Garcia Molina and K. Salem. SAGAS. *ACM SIGMOD International Conference on Management of Data*, pages 249–259, May 1987.

[22] J. E. B. Moss. *Nested Transactions: An approach to Reliable Computing*. PhD thesis, MIT, April 1981.

[23] E. Pitoura and B. Bhargava. Maintaining Consistency of Data in Mobile Distributed Environment. In *15th Int. Conference on Distributed Computer Systems*, Vancouver Canada, May 1995.

[24] E. Pitoura and B. Bhargava. Data Consistency in Intermittently Connected Distributed Systems. In *Transactions on Knowledge and Data Engineering*, Nov 1999.

[25] E. Pitoura and G. Samaras. *Data Management for Mobile Computing*. Kluwer Academic Publishers, 1998.

[26] C. Pu, G. Kaiser, and N.Hutchinson. Split Transactions for Open-Ended Activities. In *Proceedings of the Fourteeth International Conference on Very Large Databases*, pages 26–37, September 1988.

[27] K. Ramamritham and P. K. Chrysanthis. *Advances in Concurrency Control and Transaction Processing*. IEEE Computer Society Press, 1996.

[28] G Vargas-Solar. *Service d'Evénements Flexible Pour l'Intégration d'Applications Bases de Données Réparties*. PhD thesis, Université Joseph Fourier, December 2000.

[29] G. D. Walborn and P. K. Chrysanthis. PRO-MOTION: Management of Mobile Transactions. In *11th ACM Annual Symposium on Applied Computing*, San Jose Ca, March 1997.

[30] G. D. Walborn and P. K. Chrysanthis. Transaction Processing in PRO-MOTION. In *14th ACM Annual Symposium on Applied Computing*, San Antonio Tx, February 1999.

[31] L. H. Yeo and A. Zaslavsky. Submission of transactions from mobile workstations in a cooperative multidatabase processing environment. In *Conference on Distributed Computing Systems*, 1994.

[32] T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, Englewood Cliffs, New Jersey, 2nd edition, 1999.