A version of this work has been published in the *Proceedings of ADBIS-DASFAA 2000*, September 2000. Springer-Verlag LNCS 1884.

Mobile Transaction Management in Mobisnap[*]

Nuno Preguiça°, Carlos Baquero↑, Francisco Moura↑, J.Legatheaux Martins°, Rui Oliveira↑, Henrique João°, J.Orlando Pereira↑, Sérgio Duarte°

↑ Departamento de Informática
Universidade do Minho
Largo do Paço, 4700 Braga
Portugal
{cbm,fsm,rco,jop}@di.uminho.pt

° Departamento de Informática
FCT, Universidade Nova de Lisboa
Quinta da Torre, 2845 Monte da Caparica
Portugal
{nmp,jalm,hj,smd}@di.fct.unl.pt

Abstract

To allow mobile users to continue their work while disconnected, mobile systems usually rely on optimistic replication techniques. In mobile database systems, mobile units cache subsets of the database state and allow disconnected users to perform transactions concurrently. These transactions are later integrated in the master database state. As concurrently performed transactions may conflict, it is usually impossible to determine the result of an update in the mobile unit. Moreover, this model differs from the traditional client/server model due to the fundamental fact that the user will usually not be connected to the system when the results of his transactions are finally determined - therefore, he can not immediately perform adequate alternative actions. In this paper we describe a transaction management system that takes into consideration the above-mentioned characteristics. Transactions are specified as mobile transactional programs, which allows the precise definition of operation semantics and the definition of alternative actions. Support for active user notification is also provided in the system. Finally, the system relies on a reservation mechanism to be able to guarantee the results of transactions in the mobile units.

Keywords

Transaction processing, operation-based update propagation, reservations, awareness.

1. Introduction

In recent years, advances in hardware and communication technology lead to a dramatic increase in the use of portable computers, ranging from small handheld personal digital assistants to powerful laptop devices. These computers have been used mainly for accessing the web and to manage personal or private information, such as address and date books, private documents, and electronic mail. However, the widespread availability of these devices opens opportunities for the development of applications where multiple mobile users independently access and modify shared information. Mobile computing inherent characteristics [13,14] lead to novel requirements that demand specific solutions. In particular, mobile applications have to face periods of disconnection that may arise due to economical factors, unavailable connectivity or application model. Mobile users expect to be able to continue their work even in these periods.

To face the above-mentioned problems, it is common to rely on optimistic replication techniques. In such approaches, shared data is replicated on mobile computers and users are allowed to continue their work while disconnected. Updates performed by disconnected users are logged and later propagated to servers. Several problems arise from such approaches: (1) updates performed by different disconnected users may conflict among them; (2) due to the previous problem, it is usually impossible to determine the result of an update in the mobile device; (3) mobile users may wish to perform operations over data that is not locally replicated. In this paper we present the Mobisnap [8] approach to tackle these problems in a relational database system.

The Mobisnap system is based on a central database server that holds the primary replica of all data items. Mobile clients replicate subsets of the database information and mobile users are allowed to update database information through the submission of "mobile transactions". These "mobile transactions" are specified in an extended subset of the PL/SQL language [11], allowing programmers to clearly state the intended semantics of

[*] This work was supported in part by PraxisXXI.

each operation - pre-conditions, post-conditions and different alternatives may be easily defined for each transaction. The final result of a "mobile transaction" is only determined when the transaction is performed in the central database. The Mobisnap system provides adequate linguistic and system support to allow mobile users to be notified of this final result - we believe that this is an important feature because, usually, mobile users will not be connected to the system when the final results of their transactions are determined.

If enough resources are available, mobile clients maintain two copies of the locally replicated data: a committed one that has been obtained from the database server, and a tentative one that reflects the updates performed locally to the committed state. This approach has been previously identified as interesting for such environments [15,5]. "Mobile transactions" submitted while disconnected are tentatively applied to the database tentative state. The result of the execution of one transaction represents the expected result of its execution in the central database. However, concurrent updates performed by other users may lead to a different result when the transaction is performed in the master database. To alleviate this problem, we have designed a reservation mechanism that allows mobile clients to make reservations upon database information. Therefore, mobile clients are able to determine the result of mobile transactions that only depend on reserved information. This mechanism combines leasing [2] with an extension of previously proposed escrow techniques [7].

The remainder of this paper is organized as follows: section 2 discusses some requirements that we consider important to support transaction processing in mobile computing; section 3 describes the Mobisnap transactional model; section 4 presents system design; section 5 discusses related work and section 6 concludes the paper with some final remarks.

2. Motivation

In this section we present the ideas that lead to the Mobisnap approach to mobile transaction processing. Details about the outlined mechanisms will be described in the following sections.

Mobile transactions should be conceptually simple

In database systems, transactions are used to coherently modify the state of the database as the result of some (usually external) operation - for example, in consequence of a money transfer two bank accounts are updated. A transaction processing system, such as a database management system, guarantees that the execution of transactions respects the ACID properties (atomicity, consistency, isolation and durability). These properties ensure that after the execution of any transaction the database remains in a consistent state, either reflecting all updates performed by the transaction or none. They also guarantee that each transaction is executed atomically without interference of any other concurrent transaction. The ACID properties provide a simple semantics for the computation of a transaction. This simplicity is one of the greatest strengths of transactions because they allow programmers to specify operations in a simple way. Therefore, we believe that mobile transactions should also have a simple semantics taking into consideration the mobile computer constraints.

In general, it is necessary to rely on optimistic replication to face disconnection. Therefore, as different disconnected users may concurrently issue conflicting operations, it is not possible to determine the result of transaction execution in mobile units. The problem lies in the fact that the local state of the database may have been modified and the values accessed during transaction execution may no longer hold. Transaction processing correctness is commonly enforced through the avoidance or detection of read/write and write/write conflicts. Although the same mechanism may be used for mobile transaction processing, it would impose a too restrictive model because it would only allow the execution of transaction that access elements of the database that have not been modified by any other transaction. Therefore we propose the definition of mobile transactions in an imperative language based on PL/SQL [11], allowing programmers to easily define the desired operations' semantics resorting on pre-conditions, post-conditions and different alternatives. The result of a mobile transaction is completely and safely determined by the execution of the transaction program in the server, i.e. on the primary copy. The programmer can reason about the mobile transaction as a mobile program sent from the client to the server holding the primary copy and executing later on that server. We believe that this approach offers a conceptually simple model, allowing simple operation definition. We also believe that this model allows a high degree of concurrency and scalability since the asynchronous nature of the client/server interaction and the semantics of mobile transactions only require short lasting locks in the database (no "long transaction" processing is required).

Awareness should be a first-class citizen

The usual client/server approach to transaction processing assumes that the user that issues a transaction is connected to the system when its execution is completed. Therefore, users can be immediately notified of the results of their transactions and may perform alternative actions if necessary - a typical example is the flight reservation system. The outlined mobile transaction model differs from this approach in a fundamental way - usually, users will not be connected to the system when the results of their transactions are determined. Therefore, we believe that a system that supports mobile transactions should integrate a simple and clean mechanism to actively notify users of the results of their transactions (additionally, a pull-based mechanism should also be provided). It should also allow the definition of alternative actions dependent on the current state of the database.

In the Mobisnap approach, we have defined a function that can be used to submit notification to users as part of the transaction execution. Therefore, it is very simple to notify users of the results of their transactions. As it has already been mentioned, the mobile transaction programs submitted by applications can be used to define alternative actions, thus introducing the possibility to perform alternative actions when some desired action is not possible. In figure 1 we present a simple example of a mobile transaction, where alternative schedules for a meeting are defined and notification to users are performed.

```
DECLARE
   cnt natural;
BEGIN
   SELECT count(*) INTO cnt FROM meetings WHERE date = '17-FEB-2000' AND hour = 10;
   IF (cnt = 0) THEN
      INSERT INTO meetings VALUES ('17-FEB-2000', 10, 'Mobisnap meeting');
      NOTIFY( 'SMTP', 'mobisnap@di.fct.unl.pt', 'Meeting scheduled for 17-FEB-2000 at 10:00.');
      COMMIT;
   ENDIF;
   SELECT count(*) INTO cnt FROM meetings WHERE date = '18-FEB-2000' AND hour = 9;
   IF (cnt = 0) THEN
      INSERT INTO meetings VALUES ('18-FEB-2000', 9, 'Mobisnap meeting');
      NOTIFY( 'SMTP', 'mobisnap@di.fct.unl.pt', 'Meeting scheduled for 18-FEB-2000 at 9:00.');
      COMMIT;
   END IF;
   ROLLBACK;
ON ROLLBACK
   NOTIFY( SMTP, 'mobisnap@di.fct.unl.pt', 'Impossible to schedule Mobisnap meeting');
END;
```

Figure 1 - Definition of a simple mobile transaction to perform the scheduling of a meeting. Two alternative time-slots are checked.

Guarantees are valuable

In the proposed mobile transaction model, as in others previously proposed in literature [15,5,3,12], the result of a transaction submitted in a mobile unit is only determined when the transaction is finally executed in the database server. However, the transaction is tentatively performed in the mobile unit to provide a hint of its final result. In some applications, the ability to provide a stronger hint about the results of transactions would be very valuable - for example, salespersons would like to immediately guarantee that they could meet some request. To this end, we have integrated a reservation mechanism in the Mobisnap system.

This reservation mechanism combines and extends ideas used previously in [7] (escrow techniques) and [2] (leases). We have defined four types of reservations:

1. Escrow. It is used to divide a partitionable resource. For example, different subsets of the available instances of a given product can be reserved by different salespersons.
2. Slot. It is used to reserve the right to insert a record with pre-defined values. For example, someone may want to reserve the right to schedule a meeting in a room in a defined period.
3. Value-change. It is used to reserve the right to change some values in the database. For example, someone may reserve the right to change the description of some product.
4. Value-use. It is used to reserve the right to perform transactions that use a given value for some fields. For example, a salesperson may reserve the right to sell some product for a given price, even if the price is updated.

Reservations held by mobile units are limited in time, thus guaranteeing that the system will be able to use the reserved values after a limited period of time even if the mobile unit becomes permanently disconnected. Due to this reservation mechanism, the result of a transaction can be correctly established in the mobile unit if it only depends on reserved values (assuming that the mobile transaction can be propagated to the server before the expiration of involved reservations). In the next section we detail the outlined mechanism and describe the global model for transaction processing in Mobisnap.

3. System Model

The Mobisnap system manages information structured according to the relational data model. Its architecture is based on the extended client-server model [4]. The server component is composed by a well-connected[1] server that holds the primary copy of all data items. The clients are devices that locally replicate a subset of the database state[2]. Clients are allowed to continue their normal operation even if they are disconnected from the server. Clients can be mobile or stationary computers. In most situations the server is a stationary computer but nothing prevents the server from being mobile, as long as the well-connected assumption holds.

Clients maintain two copies of the replicated data: committed and tentative. The committed version contains data received directly from the server and it reflects a possibly outdated database state. The tentative version is based on the committed one and it reflects the execution of previously submitted mobile transactions in the client unit. While disconnected, applications may access both database versions through different API function calls. Applications should reflect the possible weak consistency of data to users, and they should use the tentative data version to present the expected data evolution. If resources available in the client device are not enough to hold both data versions, a single version is maintained - by default, the tentative[3].

When clients interact with the server to fetch data copies, they can also request data reservations. In the previous section we have already defined the different types of reservations that the Mobisnap system can handle. For each specific database, the database designer should specify the associated reservation script. This script specifies the data elements that can be reserved. For each data element, it defines the type of reservation available. In escrow reservations, it also defines the number of instances that can be reserved by each client. Finally, the reservation script specifies for how long can a reservation be granted. It should be noted that the efficiency of the reservation mechanism depends highly on the adequate definition of the above parameters for each specific system. It is also important that clients request the adequate reservations for their operation.

As usual, users manipulate the database information using applications that run on client units. These applications display data and provide operations to modify the database state through a graphical user interface. In consequence of each performed operation that modifies the database state, the application creates a mobile transaction instantiating a previously defined template with the values specified by the user. This mobile transaction program is submitted to evaluation.

When submitted by an application, a mobile transaction is immediately executed in the client unit. The result of its execution in the client can be one of the following:

- Reservation commit. This result means that the mobile transaction code has executed successfully until a commit instruction and all tests performed during its execution are backed up by granted reservations. It should be noticed that the system only takes into consideration reservations that will expire after the next scheduled contact to the server. Application developers and users should note that this result only guarantees final commitment of the transaction if it correctly tests all dependencies and if it is propagated to the server before the expiration of the involved reservations.
- Tentative commit. This result means that the mobile transaction code has executed successfully until a commit instruction using the tentative database state. However, there is not enough reservations that back up this transaction.
- Tentative abort. This result means that the mobile transaction code has executed until an abort instruction using the tentative database state.

---

[1] By well-connected we mean that the server will be available to receive clients interactions most of the time.

[2] Clients store a subset of the database tables. For each table they store a subset of the table columns. They store a subset of the records that exist in each table.

[3] We are currently evaluating the possibility to recreate the tentative version, on demand, through the application of previous operations to the committed state

- Unknown. This result means that the currently cached data is not sufficient to evaluate the result of the transaction (e.g. a field or record that is referenced is not cached).

Mobile transactions are also stored by the system for later propagation to the server. By default, transactions that have been "tentatively aborted" are not propagated to the server. However, applications may request the propagation of all transactions. Transactions that have returned the result "reservation commit" are propagated to the server associated with references to the reservations used to guarantee them.

When the server receives a mobile transaction, it executes its transactional program. This execution may lead to the final commit or abort of the transaction. Besides propagating to users any messages defined in the mobile transaction, the server maintains a log recording the result of any executed transaction. Clients may access this log, if needed, to verify the result of any transaction. It should be noted that reservations are taken into consideration when transactions are executed[4]. Moreover, information about reservations used in clients by a transaction is also considered when it is executed in the server (in the next section we present our approach to tackle these problems).

The reservation mechanism may lead to the abortion of some transactions that could have been committed - for example, if a reservation that reduces the available instances of a given product is not used, any aborted transaction that have needed to use some of the reserved instances could have been committed. To face this problem we are evaluating the possibility to reevaluate aborted mobile transactions after the expiration of reservations that make their commitment impossible. However, this problem needs further investigation due to its possible influence in the reservation mechanism.

4. System Design

In this section we present the part of the Mobisnap system design that is related with transaction management and we describe how to implement the model defined in the previous section. In the following paragraphs we will explain how the system components depicted in figure 2 work in practice. Some solutions could be improved with a closer integration between the different Mobisnap components and the underlying relational database system.

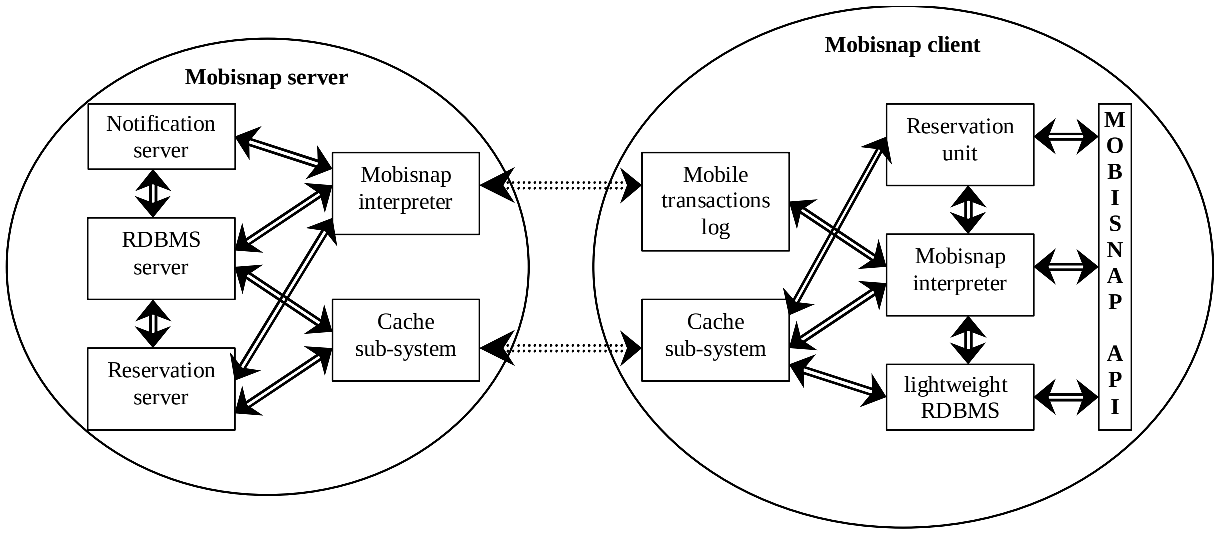

Figure 2 - Components of the Mobisnap system.

Obtaining data replicas and reservations

- In the server

The cache sub-system receives requests for copies of the database state and for reservations. Requests for reservations are propagated to the reservation server, where they are handled through the following process:

[4] For example, if for some product there are 10 instances in stock and some mobile client reserves 4 instances, only 6 instances are available for usage.

1. The validity of the request is evaluated taking into consideration the reservation script associated with the database (i.e. the allowed reservations defined by the database designer), the currently granted reservations, the current database state and the identity of the client.
2. If the reservation is granted, the database state is updated accordingly[5]:
   - For escrow reservations, the reserved instances are removed from the involved database fields.
   - For slot reservations, a dummy record is inserted in place.
   - For value change reservations, a trigger that aborts any transaction that updates the involved value is set.
   - For value use, nothing is done.
3. Besides the above updates, the reservation server maintains information about granted reservations. When reservations expire, it is responsible to "undo" the previous updates.

The requests for copies of the database state are obtained querying the database. The obtained values are then updated accordingly to the currently granted reservations in order not to reflect the granted reservations. Information about reservations and copies of the database state are propagated to the client unit.

- In clients

The cache sub-system receives from its server's counterpart the information about granted reservations and copies of the database state. The information about reservations is propagated to the reservation unit. The received subset of the database state is used to create or update the database cache which is stored in a local lightweighted database. The client database replica only contains the fields cached. However, information about all fields in each table is stored in the client for use in local operations, as it will be explained later. It is important to notice that the cache sub-system in the client unit is responsible to request a subset of the database state and associated reservations that are adequate for users operation. All problems related with reservation and cache definition, management and update are out of the scope of this paper and they will be presented elsewhere.

Processing mobile transactions

- In clients

Applications submit mobile transaction through the Mobisnap API, which forwards them to the Mobisnap interpreter. This component is responsible to execute the mobile transaction program, parsing and interpreting the PL/SQL constructions allowed in Mobisnap. SQL embedded statements are extracted and submitted to the local database for evaluation after being adequately manipulated. Any statement related with awareness information is ignored. The transaction program is executed using the following rules:

1. Every SQL embedded statement is manipulated to remove any reference to fields not available in the cache. Any query statement that can not be solved due to an unavailable field or variable with "unknown" state makes the state of the involved local variables "unknown". Any tests using a variable with value "unknown" lead to the local result of the transaction execution being "unknown".
2. "Select into" statements that involve reserved values (escrow, slot or value use) reflect only the previously unused granted reservations (for example, in a escrow reservation if the mobile client has a reservation for 4 instances of some product, but the cached value states 10 available instances, the query returns only 4 instances). The used reservation is associated with the variable.
3. Conditions that involve variables associated with reservations make those reservations associated with the current transaction.
4. Conditions that involve any variable not associated with a reservation make it impossible the transaction result "reservation commit". The transaction must be re-executed using only condition 1. If the transaction execution ends with an abort the result is "tentative abort". Otherwise, the result is "tentative commit"
5. Updates (Inserts) involving escrow (slot) reservations associated with the current transaction are interpreted to extract the reservation usage (for example, the statement "update product set stock=stock-1" uses a single instance of an existent reservation).

[5] We have decided to update the database state to allow concurrent database clients that do not use Mobisnap.

6. Updates involving "value change" reservations make those reservations associated with the current transaction.
7. If the transaction execution is aborted due to an abort statement, the transaction should be re-executed as in 4.
8. If the transaction execution succeeds, the result is "reservation commit". Reservations associated with the transaction are recorded for propagation to the server and for local available reservations update.

After being locally executed, transactions are forwarded to the mobile transaction log where they are stored until they are propagated to the server (as it has been previously mentioned, "tentatively aborted" transactions are only propagated upon explicit request).

- In the server

Mobile transactions received from clients are processed in the Mobisnap interpreter. For any transaction that does not include any associated information about used reservations, its execution is a simple interpretation of the allowed PL/SQL instructions (SQL embedded statements are extracted and submitted to the database server for execution). If the client remains connected to the server while its mobile transactions are executed, its currently unused reservations are made available to these transactions (e.g. if the mobile client remains with reservations for 3 instances of a given product and the master database has 2 unreserved instances, mobile transactions have 5 available). When there are associated reservations, the reservation server is called to verify the validity of used reservations and to update their current state. The interpretation of the mobile transaction is preceded by the update of the database state accordingly to the used reservations - these actions are performed atomically.

Notifications defined in the execution of mobile transactions are forwarded to the notification server. This server is responsible to propagate the notifications through the defined mechanism - e-mail, SMS or other (to cope with temporary impossibilities of propagation, several retries may be necessary). The Mobisnap server maintains in the RDBMS server a system table in each database containing the state of all transactions received from clients during a certain period of time (i.e. if the transaction has been committed or aborted). Any problems that may have occurred in the propagation of notifications are also stored.

Processing SQL statements in clients

In our current design the two data versions are implemented using two independent copies of each involved table (a predefined prefix identifies which data version is stored in the database table). Therefore it is rather simple to modify the SQL statements to make them manipulate the expected data version - only the names of the tables need to be modified. To allow applications to access both data versions, the Mobisnap API provides two alternative query functions. In our current design mobile transactions always access the tentative version. This design imposes an unnecessary resource expenditure that we expect to solve in a future design - it is not intrinsic to the proposed model.

5. Related Work

Research in optimistic replication schemes for large-scale distributed systems has been conducted for a long time [1,6]. The advent of mobile computing has lead to a renewed interest in optimistic replication and reconciliation mechanisms to cope with the common periods of disconnection.

Oracle Lite [10] is a mobile database product that provides support for mobile database operation. Mobile clients cache data snapshots from the central database. These snapshots contain a subset of the database state that can be read-only or updateable. Changes preformed to the snapshots are propagated to the server where the valid transactions are applied to the master database state. Validity of transactions is checked through conflict detection - write/write, uniqueness and delete conflicts are detected. Conflict resolution functions can be associated with different database tables (or table fields). This approach does not guarantee the serializability of transaction execution - write/read conflicts are not detected. Moreover, the conflict resolution strategy does not allow the use of semantic information associated with the updates. This problem lies in its state-based approach to conflict resolution - conflicts are solved based on the conflicting data versions.

Bayou [15] is a replicated database system to support data-sharing among mobile users. The Bayou architecture is composed by a group of servers that replicate databases using epidemic techniques and by clients that access these replicas. Bayou updates (writes) include information to allow generic automatic conflict detection and resolution through dependency checks and merge procedures. Bayou data presents two values: tentative and committed. A primary replica scheme is used to fasten update commitment. One of Bayou's main strengths is its update definition. However, we believe that it lacks an integrated notification mechanism. Moreover, no guarantees about the result of an update can be provided in the mobile units.

In [3] the authors propose a two-tier replication algorithm that includes connected base nodes and disconnected mobile nodes. Mobile nodes propose tentative update transactions to modify objects mastered in base nodes. When a mobile unit connects to a base node, the tentative transactions are propagated and reapplied to the master copy. An acceptance rule can be specified to verify the validity of transaction execution. If any transaction is considered not valid it is aborted and a diagnostic message is returned to the base node. This approach can not provide any guarantees in clients. Moreover, we believe that a more "aggressive" error-report mechanism should be provided to allow direct user notification.

In [12] the authors propose a transaction management system for mobile databases. This system is based on optimistic replication and it allows mobile users to perform concurrent transactions while disconnected. The client stores the read and write sets (and the values read and written) for each transaction. Moreover, for each transaction two functions should be specified: a conflict resolution and a cost function. Mobile transactions are propagated to the server to be reintegrated in the database (the reintegration algorithm serializes transactions based on the cost and conflict resolution functions). The conflict resolution function is always executed in the server and it can capture not only the actions of client transactions but can extend them to capture additional semantics on the server. Although this approach is very general, we believe that a single semantics for the client and the server transactions is more adequate - it presents a simpler conceptual model that it is easier for programmers and users to understand. Moreover, we think that the reintegration algorithm is too complex to be useful in practice. No mechanisms to provide guarantees to mobile users are presented.

The use of escrow techniques to increase the degree of concurrency in transaction processing has been proposed for a long time [9]. The idea is to divide the total number of available instances of an item among concurrent transactions - therefore, concurrent transactions are allowed to proceed if enough instances of an item are available. The same idea can be applied to manage replicated data - instances are divided among different sites. In [7] the authors describe a model to allow mobile units to independently guarantee the results of mobile transactions based on escrow techniques. A reconfiguration protocol is also described. In our work, we propose a reservation mechanism that includes these ideas and allows mobile units to guarantee the results of mobile transactions in new circumstances.

In [16] the authors generalize the usage of escrow techniques by exploiting object semantics. The idea is to split large and complex objects into smaller fragments with certain constraints. These fragments can be modified independently since the constraints are honored. These modified fragments can be later merged using the semantic information available. This approach can be used only with some data types and it is more adequate to object oriented databases.

6. Final Remarks

Mobile systems have to face a set of problems that arise from the inherent characteristics of mobile environments. To face disconnection, optimistic replication techniques with an update anywhere-anytime model must be used. Clients cache subsets of the database state. Updates are stored and later propagated to the server. This approach leads to several problems that must be tackled by the mobile data management system. In particular, the system must handle concurrent updates that may conflict among them. Moreover, it has to take into consideration that the operational model differs from the traditional client/server model in one fundamental aspect: the final result of transaction execution may only be determined in the server. In this moment the users will usually not be connected to the system.

In this paper, we have presented the Mobisnap mobile transaction management model. In this model, mobile transactions performed by applications are defined in a language based in PL/SQL. These "transaction programs" are propagated and executed in the server, thus allowing the validation of transaction execution based on application-specific semantics. As mobile users are not usually connected to the system when the final result of a transaction is determined and therefore they can not immediately perform alternative actions if it aborts, these programs may contain a set of alternative actions to be executed depending on the database state. Moreover, our model provides explicit mechanisms to provide awareness information to mobile users - a mobile transaction may specify the propagation of messages containing the result of its execution to mobile users using different mechanisms (e-mail, SMS/pager,…). One important aspect of our model is the reservation mechanism. It allows mobile units to guarantee the commitment of mobile transaction in some circumstances. We propose the definition of different types of reservations.

We have also presented a system design to implement the proposed model. One important aspect of the design is its independence of the underlying relational database system - it can be implemented using any database that supports SQL. Moreover, a different database system may be used in the server and in clients. The presented design is composed by several components that cooperate to implement the defined model - the relational database is one of these components. We are currently implementing the Mobisnap system using the presented design. A preliminary prototype with some limitations is already operational - slot and value-change reservations are still under implementation. It has been implemented using the Java language and the JavaCC parser generator. A rudimentary mobile sales application has already been used to test the prototype. We expect to implement more complex mobile applications to further validate and refine the proposed model.

7. References

[1] S. Davidson, H. Garcia-Molina, D. Skeen. Consistency in Partitioned Networks. *ACM Computing Surveys*, C-31, 1982.

[2] C. Gray, D. Cheriton, Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, 1989.

[3] J. Gray, P. Helland, P. O'Neil, D. Shasha. The dangers of replication and a solution. In *Proceedings of the ACM SIGMOD'96*, 1996.

[4] J. Jing, A. Helal, A. Elmagarmid. Client-server computing in mobile environments. *ACM Computing Surveys*, 1999.

[5] A. Joseph, A. DeLespinasse, J. Tauber, D. Gifford, M. Kaashoek. Rover: A Toolkit for Mobile Information Access. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, 1995.

[6] L. Kawell Jr., S. Beckhardt, T. Halvorsen, R. Ozzie, I. Greif. Replicated Document Management in a Group Communication System. In *Proceedings of the 2nd ACM Conference on CSCW*, 1988.

[7] N. Krishnakumar, R. Jain. Escrow techniques for mobile sales and inventory applications. *Wireless Networks*, 3, 1997.

[8] Mobisnap team. Mobisnap - Managing database snapshots on a mobile environment. *Technical report*, 1999.

[9] P. O'Neil. The escrow transactional model. *ACM Transactions on Database Systems*. 1986.

[10] Oracle. Oracle8i Lite replication Guide - release 4.0. 1999.

[11] Oracle. PL/SQL User's guide and reference - release 8.0. June 1997.

[12] S. Phatak, B. Badrinath. Multiversion reconciliation for mobile databases. In *Proceedings of ICDE'99*, 1999.

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

[14] M. Satyanarayanan. Fundamental Challenges in Mobile Computing. In *Proceedings of the 15th ACM Symposia on Principles of Distributed Computing*, 1996.

[15] D. Terry, M. Theimer, K. Petersen, A. Demers, M. Spreitzer, C. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, 1995.

[16] G. Walborn, P. Chrysanthis. Supporting semantics-based transaction processing in mobile database systems. In *Proceedings of the 14th Symposium on Reliable Distributed Systems*, 1995.