A Framework for Analyzing Mobile Transaction Models

Roberto Hirsch, Andrés Coratella, Miguel Felder,* and Eduardo Rodríguez
Universidad de Buenos Aires, Argentina

*Also with Pragma Consultores

Currently, mobile technology is undergoing a high growth stage, allowing an increasing plethora of mobile devices (handheld PCs, handhelds, etc.) to daily access to distributed resources and information. This availability entails the requirement for transactional capabilities adapted to the specific characteristics of the mobile environment without losing the consistency and reliability guarantees of traditional OLTP systems. This paper surveys the definition and extension of transactional models to a mobile environment, starting with an explanation of this environment and a review of transactional systems applied to mobile computing. Afterwards, a framework for analyzing competing mobile models is defined. This framework allows for different constraints to be imposed to the most general "motion independence" requirement. Finally, existing mobile transaction proposals are assessed against the framework and classified, highlighting their relative strengths and weaknesses in different situations.

In a mobile computing environment, computers are connected to the network via a wireless interface, and move through different geographical areas trying to maintain their connections. Mobile users are potentially allowed to access remote data at any moment and place (Mazumdar et al., 1999). The current high growth of mobile technology increases the number of mobile users and available resources, thus motivating the need of using transactional systems that take into account the characteristic features of the environment. Although in essence a wireless network of mobile clients can be also considered a distributed system, there are some features that makes it unique (Lee et al., 1997; Satyanarayanan, 1996; Barbá, 1999; Mazumdar et al., 1999; Pitoura, 1998):

- Communications asymmetry: server-client communication bandwidth is much bigger than that of the client-server communication. In some systems, clients do not even have the capability to send messages to the servers. Therefore, it is better for the server to disseminate (broadcast) data to the clients instead of waiting for their request. This is called push-based dissemination. In this case, the clients frequently monitor received data and take what they need as they arrive through the communication channel. In the opposite scenario, called pull-based, the server receives explicit client requests, locates the corresponding data items and sends them back.

- Limited resource capability: Mobile computers have smaller capability resources (disk, memory, processing power) than those of static computers.

- Physical risks: There are more possibilities of theft, damages caused by crashes, loss of the computer, etc. These factors should be kept in mind when evaluating where critical or sensitive data will be stored or how it will be managed.

- Frequent disconnections: Mobile clients do not stay connected to the network as fixed computers do, since users turn on and off their computers regularly, or due to slowness of communication networks. Another effect of weak connectivity is the intermittent reception of messages (for example when passing under a tunnel). Also, mobile clients can roam, disconnecting themselves from a cell and then connecting to another one. These disconnections generate transactional systems' implementation problems, and also impact database consistency maintenance.

- Power limitations: Some mobile computers are severely limited by the energy they can use before having to recharge batteries. For instance, a mobile computer can turn off the communications subsystem only in order to save energy, even though it has no connection problem.
• **Screen size:** Some portable devices, such as Personal Data Assistants (PDAs) or cellular telephones, have very small screens that limit the design of application interfaces.

One of the objectives of this work was the study of transactional models applied to mobile environments. After analyzing the state of the art, we observed that none of the models covers all the necessary characteristics for current requirements, and therefore we propose a framework that allows us to capture and compare the main features to be taken into account in these models.

In this work, we present a revision of transactional systems applied to mobile environments. First, we define the mobile computing environment; after that, the basic transactional systems’ concepts are studied, together with a revision of traditional and non-traditional transactional models used in static distributed systems. Next, we describe some techniques that take advantage of the inherent features of mobile computing, and transactional models used in these environments. Then, a framework is presented, in order to analyze transactional models that were conceived keeping in mind mobile environment requirements and restrictions.

Finally, these models are evaluated using the defined framework, highlighting advantages and disadvantages of each one. We present the conclusions of our work.

**MOBILE COMPUTING**

In this section, we describe the mobile computing environment. This environment is made up of a set of devices that interact among them to give the user the possibility to stay connected while moving from one location to another. The way it was designed, together with current technology limitations, are the reasons for the previously mentioned characteristics.

**Environment**

A mobile computer system environment (Tewari et al., 1995; Varshney, 1998; Barbará, 1999; Yen et al., 1997) basically consists of a fixed network of computers with a wireless interface (MSS, Mobile Support Stations) that offers information and services support to mobile computers (MH, Mobile Hosts). Each mobile support station covers a logical or geographical area called cell, and conforms a wireless network together with all mobile computers connected to it (see Figure 1).

**Autonomy vs. Dependency**

Mobility increases the compromise between autonomy and dependency that is typical for distributed systems. As it was previously said, physical risks, energy limitations and performance problems entail the necessity of completely leaning on static servers. But, on the other hand, difficulties in communications take us to also think in the necessity of increasing the autonomy of mobile computers, requiring that the clients emulate server features at some moments. For these reasons, the distinction between clients and servers can be sometimes diffuse (this is called extended client-server model). For example, a mobile computer starts operating as a client, but while it is disconnected, it acts as a transaction server, and thus it must have the appropriate objects (data, registers, etc.), a transaction monitor, a locking and concurrency control system, etc.

There exist basically three strategies to support this duality, as mentioned in Satyanarayanan (1996):

- **Laissez-faire:** There is no mobile computing system support, so each application is completely responsible for taking the corresponding decisions.
- **Application-transparent adaptation:** The system is absolutely responsible for supporting mobile computing.
- **Application-aware adaptation:** It is a balance between the previously mentioned extremes.

**TRANSACTIONAL SYSTEMS**

Transactional systems are used to achieve a reliable and consistent information management. In this section we briefly summarize the non-traditional transactional models, and we also present a comparison of these models, taking into account the most important characteristics.

**ACID Properties**

We assume the standard ACID properties (atomicity, consistency, isolation and durability) as defined in Gray et al., (1993) and Bernstein et al. (1997).

---

*Figure 1: Mobile Environment.*
Non-Traditional Transactional Models

In the traditional transactional model (flat transactions), a transaction consists of a sequence of read or write operations, starting with a begin operation and ending either with a commit or an abort operation. The commit operation permanently applies all the changes to the database, making them visible to all the running transactions, while the abort operation cancels all the updates made, as if they were never executed.

Although this model strictly meets all the ACID properties, it is not adequate to model complex situations. That is the reason why a number of non-traditional transactional models have been defined, in order to take into account the specific requirements of advanced applications.

In the following sections, we briefly describe some of these models.

Flat Transactions with Savepoints

This model is similar to Flat, with the possibility to define checkpoints inside a running transaction (Gray et al., 1993). When such a checkpoint is defined, the system saves the transaction’s state, allowing to return to this state in the future, as long as the transaction has not committed or aborted.

Chained Transactions

This model is related to the previous one, in which each transaction can repeatedly save its state; but the difference is that it does so by calling a chain operation. This chain operation is similar to a commit since it saves the work done till that moment and sets it out of the scope of an abort operation. Additionally, another transaction is automatically created, preserving the locks already acquired, the open cursors, etc. (Gray et al., 1993).

Nested Transactions

A nested transaction is defined as a hierarchical tree of transactions where each node is called a subtransaction (Gray et al., 1993). A subtransaction can execute a commit (the results are not seen outside until the parent node commits) or an abort operation (the subtransactions and all its descendants abort).

Distributed Transactions

A distributed transaction is a flat transaction that executes in a distributed environment, partitioned in subtransactions that run in the network’s nodes where the referenced data is stored (Gray et al., 1993). This decomposition does not reflect the hierarchical or functional structure of the application, but rather the data distribution over the network. The decisions made by a subtransaction affect the entire transaction.

Multi-Level Transactions

Even though this type of transaction (Gray et al., 1993; Weikum, 1991) is a generalization of the nested model, when a subtransaction commits, any subtransaction can see the changes made, as long as it belongs to the same level. In addition, this model assumes the existence of a compensating transaction, which can semantically revert the modifications, in case the parent transaction decides to (or should) abort.

Sagas

A saga (Garcia-Molina et al., 1987) is defined as a long-lived transaction (LLT), i.e., a transaction whose execution takes a considerable amount of time, that can be partitioned in a collection of subtransactions, which can be interleaved with other transactions. Each subtransaction in a saga preserves the database consistency, but unlike other transactions, they are related and must execute as a non-atomic unit. Furthermore, it is not required that all the subtransactions see the same consistent state of the database. Compensating transactions must be defined if a saga aborts, in order to restore database consistency by semantically undoing the modifications, not necessarily taking the database to the same state it had at the time the saga was started.

A saga is similar to a nested transaction, except that:

- a saga allows only two nesting levels: the main transaction (saga) and the subtransactions;
- full atomicity is not provided at the main level. A saga can see the partial results generated by another saga. In addition, the isolation is relaxed, since there may exist interference between transactions.

Split-Transactions

This model (Puet al., 1988) was defined to support open-ended activities, characterized by its uncertain duration, its unpredictable execution, and its strong interaction with other concurrent activities. Thus, a new class of transactions is introduced, called split-transactions, which divide a running transaction in two serializable transactions. These can execute in a sequential or nested concurrent model.

When this splitting takes place, the read and written objects are distributed between the original and generated transactions. Each transaction executes independently, and one of them can commit before the other commits, thus freeing its objects and publishing the changes made so far. If the objects are accessed by both transactions (serial case), there should be an order of execution, which is not necessary if they access different objects (independent case).

Transactions can end by calling commit or join-transaction. By calling the latter operation, a transaction refers to another one, depending on it for freeing its objects.

Flexible Transactions

This model (Elmagarmid et al., 1990) was created to use transactions in a multidatabase system, where there are several independently created and maintained databases. A fun-
damental characteristic of a multidatabase system is the autonomy of the involved databases. This reduces the ability to support atomic transactions, and influences its performance as well.

The model named *Flexible Transactions* is basically a nested model with two levels. In the first level, the global transaction is defined. This transaction is partitioned in several subtransactions, which make up the second level. Some subtransactions may commit before the parent transaction, and thus it is necessary to define compensating transactions to revert their effects. Compensable and non-compensable transactions can be combined inside a single transaction (*mixed transaction*). Compensable transactions can commit before the parent commits, while non-compensable transactions must wait for a global decision. A transaction consisting solely of compensable subtransactions is called *saga*.

**Multitransactions**

In Buchmann et al. (1992), a transactional model intended to be used in distributed object systems with heterogeneous databases is presented. The model consists of basic building blocks from which more complex transactions can be defined, as needed by the application. Therefore, transactions can act as flat, nested, open-nested, or a mixture of these models.

The following is the list of transaction types supported:

- **Multitransactions**: Defined for long-lived transactions which can potentially access multiple databases. A multitransaction consists of transactions, each of which can be either a nested- (see below) or a multitransaction. It is assumed that these transactions execute in parallel, unless a precedence constraint is specified between some of them. Every transaction must end with a *commit* or *abort* operation for the multitransaction to end. In terms of visibility, these transactions can commit independently, allowing their results to be seen by other multitransactions. That is the reason why multitransactions must define compensating transactions.

- **Nested Transactions**: they are the roots of the closed nested transactions’ trees. A tree publishes its results when the root transaction (called *toptransaction*) commits. Any subtransaction of a tree can commit, but its effects are hidden from the outside world, and they are only seen by the parent and the siblings. It should be noted that nested transactions can be either specified by the user or fired by rules when a transaction executes. In the latter case, the structure depends on the coupling mode defined for the rule.

- **Compensating Transactions**: these have the opposite effect of its associated transaction, and are required in order to be able to use *open nesting* (in other words, to be able to publish their results before committing the whole multitransaction). When a multitransaction aborts, its active transactions are aborted, and the committed ones are compensated following the inverse order of commitment. It is worth noting that transactions that have been fired by rules are not directly compensated; instead, the system assumes they will be compensated when the user-defined transactions are compensated. Finally, note that compensating transactions should never abort and, if they do, a special recovery job must be done, generally requiring manual intervention.

- **Contingency Transactions**: a contingency transaction is executed when the associated transaction fails, and before it commits, and has the same mission as its related transaction. The failure of the original transaction is difficult to define, since it can be caused by different reasons (for example, business rules not met). It should be noted that a transaction can have one or more contingency transactions. In addition, different contingency transactions can be specified according to the type of failure of the original transaction.

- **Vital Transactions**: These transactions must commit for its parents to commit. On the other hand, non-vital transactions can abort without aborting their parents.

**Countermeasure Transactions**

The model described in Frank et al., (1998) tries to increase data availability in multidatabase systems using client/server technology, by relaxing ACID properties but still helping the user to maintain the correctness needed for each application (this is called *semantic ACID*).

A global transaction is defined as a root transaction (client side), which in part is formed by subtransactions (server side), each of them running on a single site. These subtransactions can also be parents for other subtransactions, thus creating a transaction hierarchy. The communication with the user is managed from the root transaction, and all the data is accessed through the subtransactions. Each subtransaction can commit locally using the DBMS of the site where it is running.

Subtransactions can be either *compensable*, *pivot* or *retirable* (this is based on Zhang et al., 1994). A retirable subtransaction is such that its execution is guaranteed to commit locally after a finite number of submissions. A subtransaction is said to be compensable if the effects of its execution can be compensated semantically after its local commitment (this is done by a *compensating subtransaction*).

Finally, a pivot subtransaction is defined as a subtransaction that is neither compensable nor retirable. There is only one pivot subtransaction in a global transaction, and the commitment of this subtransaction implies the commitment of the whole transaction.

So there is an implicit order of execution for subtransactions: in the first place, all the compensable subtransactions must run and commit locally; after that, the pivot subtransaction tries to commit. If it fails (or if the commitment of at least one compensable subtransaction fails),

---

*Journal of Database Management* July-Sept 2001
**Table 1: Non-Traditional Transactional Models Comparison**

<table>
<thead>
<tr>
<th>Transactional Model</th>
<th>Compensating Transactions</th>
<th>Sub-Tra Transactions</th>
<th>Pre-Commit</th>
<th>ACID Relax. (*)</th>
<th>No. of Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>Flat with Savepoints</td>
<td>x</td>
<td>x</td>
<td>x(1)</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>Chained</td>
<td>x</td>
<td>x</td>
<td>x(2)</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>Nested</td>
<td>x</td>
<td>v</td>
<td>x(3)</td>
<td>x</td>
<td>N</td>
</tr>
<tr>
<td>Distributed</td>
<td>x</td>
<td>v</td>
<td>x</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>Multilevel</td>
<td>v</td>
<td>v</td>
<td>v</td>
<td>ACI</td>
<td>2</td>
</tr>
<tr>
<td>Sagas</td>
<td>v</td>
<td>v</td>
<td>v</td>
<td>ACI</td>
<td>1</td>
</tr>
<tr>
<td>Split</td>
<td>v</td>
<td>v</td>
<td>v</td>
<td>ACI</td>
<td>2</td>
</tr>
<tr>
<td>Flexible</td>
<td>v</td>
<td>v</td>
<td>v(5)</td>
<td>ACI</td>
<td>N</td>
</tr>
<tr>
<td>Multitransactions</td>
<td>v</td>
<td>v</td>
<td>v</td>
<td>ACI (6)</td>
<td>N</td>
</tr>
</tbody>
</table>

**References:**

(*) In this column we show the letter of the ACID acronym that is relaxed or eliminated from the corresponding model. For instance, if A appears, it means that the Atomicity property is relaxed.

(1) Allows the system to save intermediate states of the transaction to enable partial aborts. Changes are not seen by the outside world.

(2) Although the Chained Transactions model allows pre-commitment, the objects remain locked by the transaction, and therefore the changes are not seen by the outside world.

(3) Only the top level and the siblings can access the changes made by a lower level commit operation.

(4) These subtransactions do not reflect the functional and hierarchical structure of the application, since they follow the data distribution over the network.

(5) If compensating transactions are used, it is possible to pre-commit in order to publish the changes made to the outside world.

(6) The isolation is relaxed, but countermeasures are provided to alleviate the anomalies that may arise.

the compensating subtransactions are run so as to undo the effects caused by the already committed subtransactions (note that the compensating subtransactions must be enforceable, that is to say, they must be retried till they commit). On the other hand, if the commitment of the pivot subtransaction is successful, the retriable subtransactions start running and will be committed later on.

It is clear that the transaction model just described does not guarantee global isolation. This is mainly caused by the use of short duration locks (they are released as soon as a subtransaction commits locally) instead of long duration locks (controlled by a Global Transaction Manager, and released when the whole global transaction commits). In order to alleviate this problem, a set of concurrency techniques, called countermeasures, are defined. These countermeasures, which are managed by the transactions themselves, help to manage the isolation anomalies that may arise (examples of anomalies are lost update, dirty read, etc.; see Frank et al., 1998, and Gray et al., 1993).

An example of a countermeasure is the Rereread Countermeasure. A transaction that uses this countermeasure reads a record twice, using short duration locks for each read. If another transaction has changed the record between the two readings, the transaction aborts itself after the second read. This is often used to prevent the lost update anomaly, in which a first transaction reads a record for update without using locks, then the record is updated by another transaction, and finally the update is overwitten by the first transaction. (Please note that this countermeasure does not always avoid the lost update anomaly, since another transaction can still update the record between the second read and the update).

Another example of countermeasures is the Semantic Lock Countermeasure. It is used to implement application-specific locking concurrency control. When a compensable subtransaction reads the data, it automatically marks the data with its signature and writes the record back to the database. When the global transaction is committed, a retrievable subtransaction removes the marks of the data. Between the marking and unmarking of the record, another concurrent transaction can access the marked data and decide if it will update the record or not, depending on the semantics of the application. For more examples of countermeasures, see Frank et al (1998).

**Comparison of Non-Traditional Transactional Models.** In this section we present a table which compares the main characteristics of the non-traditional transactional models studied.

**TRANSACTIONAL SYSTEMS APPLIED TO MOBILE COMPUTING**

Now that we have described transactional models as used in usual distributed systems, we continue our work by...
presenting different techniques that take into account the particular characteristics found in mobile computer environments, and we also describe some mobile transactional models. It should be noted that these techniques are not related to a particular model, and should be carefully considered at implementation and low-level design time.

Transactional Systems’ Requirements for Mobile Environments

Below we describe the requirements needed for a transactional system in order to be applied to a mobile environment (Tewari et al., 1995) (Dunham et al., 1997) (Lee et al., 1997).

- **Ability to distribute the transaction’s processing:** Due to memory, power processing, and battery limitations of the MHs, it may be necessary to execute certain sections of the transaction in the MSS.

- **Share the state and the partial results:** As it was stated in the previous item, parts of a transaction can be executed in the MH, while others run in the MSS.

- **Capture the movement of mobile transactions:** Due to the physical movement of the MHs, it is necessary to transfer a transaction’s control as it moves from cell to cell.

- **Support long-lived transactions:** This is required since some processes can take a considerable amount of time and, besides, the search for a computer that has physically moved from a cell can be a time-costly operation.

- **Support long disconnection periods:** Recall that disconnections can be caused by physical problems, or simply by the MH’s own decision. For a MH to continue operating despite being disconnected from the network, it may be necessary to maintain local copies of the data needed (caching techniques).

- **Support partial failures and provide different recovery strategies:** Battery problems, static electricity, accidental computer turn offs, etc can cause these failures.

ACID Relaxation

In some cases, it may be necessary to relax the ACID properties to support the use of transactional systems in a mobile environment (Pitoura et al., 1994). This can be seen, for instance, in (Mazumdar et al., 1999), where global consistency is reduced to local consistency. A reason not to completely support the ACID properties is the high probability of generating lots of aborts, due to the intrinsic nature of the environment. This could turn the system useless.

Alternatively, the system could be less strict, so that some transactions, which would abort on a conventional system, could still continue their execution and, eventually, commit the changes made (Dunham et al., 1997).

Techniques Used in Mobile Environments

There are common techniques used in transactional systems running in mobile environments that can be applied in the implementation of a mobile transactional model. As we stated before, they are not related to a particular model; they are tools that should be used when designing a model’s implementation, in order to exploit the features and restrictions of these environments.

These techniques include the use of an adapted version of the *Field Call* algorithm (Lee et al., 1997), designed to minimize the duration an object was locked, especially for frequently accessed objects; data dissemination techniques (Barbará, 1999; Lee et al., 1999) used, for example, to take advantage on the possibility of data *broadcasting* to help mobile computers in determining if the transactions they are executing should be aborted or not; data replication (Gray et al., 1996), allowing the disconnected MHs to locally continue transaction processing, etc. For a deeper discussion, see (Coratella et al., 2000).

Mobile Transactional Models

The use of transactions in a mobile environment differs substantially from their use in centralized or distributed systems. The main differences are the high disconnection frequency and the mobility of the transactions. Therefore, transactional models and commit coordination protocols must be revised to take into account the mentioned differences (Dunham et al., 1997).

It should be noted that there are models, which, although not designed for mobile environments, could be useful in those environments. Examples of these are the models based on pivot transactions. In these models, the use of retrievable subtransactions makes them well suited for mobile computing, because if a mobile user has missed an answer, it will be resubmitted until it is received and committed by the MH (Frank, 1999).

In the following sections, we describe some transactional models applied to mobile computing.

Kangaroo Transactions

*Kangaroo transactions* (Dunham et al., 1997) extend global transactions (which operate in multiple databases), allowing them to capture the natural movement produced in mobile transactions. This model basically works as follows:

When a mobile computer starts a new transaction (abbreviated KT, *Kangaroo Transaction*), the base station creates a mobile transaction and the only subtransaction that will run on that station (called JT, *Joey Transaction*). These subtransactions can contain other subtransactions, either local or global. When the mobile unit jumps from one cell to another, the KT’s control changes to the new base station, where a new subtransaction JT is created during the handoff process, by means of a *split* operation. The original JT commits independently of the new JT. Anyway, as soon as a JT fails, the whole KT is aborted. This is done by compensating all the previously completed JTs.

There are two processing modes: *Compensating Mode* and *Split Mode*. In the first one, the failure of any JT implies
that all the JT's must be aborted: the ones already committed are compensated by means of user-defined compensating transactions. In the latter mode, Split, when a JT fails no transaction is compensated, and the decision as to whether commit or abort active transactions is left to the DBMS. None of these modes guarantees serializability of the KT's. Isolation can be violated since the locks are held and released at the local transaction level. With the Compensating mode, however, JT's subtransactions are serializable, since it is assumed that JT \text{start}, is aborts executing when every JT, has already committed.

**Moflex Transactional Model**

The Moflex model (Ku et al., 1998) extends Flexible Transactions to support the requirements of a mobile computing environment.

This model allows for the definition of parameters that specify, in a very flexible way, the behavior of the transactions. A Moflex transaction is made up of subtransactions, which can be either compensable or not. It is possible to define the behavior these transactions will follow, such as the success or failure dependencies they must maintain with other subtransactions, conditions regarding the geographic location, communication cost taking into account the cell the MH is currently in, etc.

When a Moflex transaction faces a handoff, it can behave as specified by the user through a series of rules. These rules allow a transaction to either continue, restart or split between the old and the new cell (using Split Transactions).

For example, it is possible to define a transaction that, depending on the geographic location of the MH, could change its behavior in order to minimize communication costs (for instance, accessing the nearest server), or could restart a subtransaction since the environment's conditions change because of the handoff.

Formally, a Moflex Transaction \text{T} consists of a 7-tuple \((M, S, F, T, H, J, J_G)\) where:

- \(M = \{t_1, t_2, \ldots, t_j\}\) is the set of all subtransactions of \text{T}, where \(t_i\) can be compensable or not.
- \(S\) is the set of success dependencies between the transactions of \(M\). A subtransaction \(t_i\) has a success dependency on \(t_i \times t_j\) if \(t_i\) can be executed only after \(t_j\) has committed.
- \(F\) is the set of failure dependencies between the transactions of \(M\). A subtransaction \(t_i\) has a failure dependency on \(t_i \times t_j\) if \(t_i\) can be executed only after \(t_j\) has aborted.
- \(T\) is the set of external dependencies (referring time, cost or localization) between the subtransactions. Subtransaction \(t_i\) can be executed when the external predicates are satisfied.
- \(H\) is the set of control rules used for every subtransaction when a handoff takes place. The possibilities are: continue, restart, split resume and split restart. If continue is used, the subtransaction continues its execution after the handoff. If restart is specified, its execution is restarted at the new cell, aborting the operations previously made (this is used, for example, when the subtransaction depends on the geographical location of the mobile computer); if split resume is used, the subtransaction commits in the previous cell and continues its execution at the new cell, starting at the point the split was applied; similarly, when split restart is used, a commit is executed at the previous cell, but the execution at the new cell starts from the beginning of the transaction.
- \(J\) is the set of acceptable rules to join a subtransaction after it was split by a handoff (when split_resume or split restart was specified). That is to say, these rules allow to verify the correctness of the split subtransactions, in order to join them in a single subtransaction. The possible values are join(t) and user(t). The first value verifies that the concatenation of the partial results equal to the result of \(t\); the latter transfers this responsibility to the user.
- \(G\) is the set of acceptable goals for transaction \(T\). In other words, it specifies the final execution states where the user gets the results he wants. This means that it is not necessary for all the subtransactions to commit, since there can be a G state where some subtransactions fail.

**Isolation-Only Transactions (IOT)**

The IOT model (Lu et al., 1994; 1995) was designed to allow disconnected operations in mobile computers, detecting read/write conflicts based on limitations of the serializability. It was originally created for the reading and writing of UNIX files through mobile computers.

An IOT transaction guarantees the consistency property depending on the connection state of the system. It does not guarantee failure atomicity (a great number of resources are necessary to undo the effects of a transaction, and these are not available at the MHs), and only guarantees durability conditionally (see below).

It works as follows:

- When the client requests a transaction \(T\), it is executed by locally accessing files, using a cache in the MH. The partial results of the transaction are not visible from the outside world.
- When a transaction terminates, it enters a commit or pending state: the first one is entered if the transaction has a connection with the server, and the second if this connection is not present. In the commit state, if all the accessed files maintained a connection with the server (first-class transactions), the results are published. In the pending state (second-class transactions), results are only locally visible, and the transaction is validated at reconnection time. For this reason, durability is not guaranteed, since the results of a transaction could have been modified after its commitment. Durability is only guaranteed for first-class transactions.
- To validate transactions at reconnection time, the following consistency criteria are used:
  - Serializability (SR) for first-class transactions:
Serializability is guaranteed with all committed transactions.

- **Local serializability (LSR) for second-class transactions:** Serializability is guaranteed with all second-class transactions executed at the same mobile computer.

- **Global serializability (GSR) for second-class transactions:** One of the criteria used to validate pending transactions is that they should be GSR with respect to all the committed transactions, i.e., when reintegrating the client's local cache with the server, pending transactions should be serializable with all committed transactions. But since verification is done at reconnection time, it must be specified what should be done in case the verification fails. The possibilities are automatic or manual-resolution.

  The automatic-resolution possibilities are the following:
  
  - **Transaction re-execution:** The transaction is executed again, with all the updated data. This is the default option.
  
  - **Invocation of an application-specific transaction resolver (ASR):** Uses specific application knowledge to solve a problem by allowing the programmer to write a failure-correction routine.
  
  - **Transaction abortion:** Aborting a non-GSR transaction can be sufficient to recover consistency.

  In the case of a manual resolution, the user is notified to solve the problem.

- **Global Certification Order (GCO) for second-class transactions:** This scheme ensures that a pending transaction is serializable not only with the committed transactions, but also after those transactions. GCO has the same resolution options as GSR.

**Weak/Strict Transactions in Data Clusters**

In this model (Pitoura et al., 1995), transaction processing is allowed even if the mobile computer is disconnected. Since this can cause information inconsistency, data clusters are defined. These clusters group data based on some criteria (semantic, physical, etc.). Data consistency must be fully supported within a cluster, while there can exist various degrees of consistency among data replicated in different clusters.

In order to support local data processing, *weak read* and *weak write* operations are defined, while the traditional read/write operations are called *strict read* and *strict write* (they maintain inter-cluster consistency). The *weak write* operation writes a local copy of the data item, and this modification is not permanent until the computer recontacts to the network and the cluster consistency is validated. The *weak read* operation reads a local copy of the required data item, that is, the value written by the last *weak write* or *strict write* operation.

Consequently, two transaction types are defined:

- **Weak transactions:** formed by *weak read* and/or *weak write* operation.

- **Strict transactions:** formed by *strict read* and/or *strict write* operations.

The use of *weak write* operations should be restricted to compensable transactions since, as was previously explained, they could be finally not accepted if they provoke inconsistencies in the cluster.

Two intra-cluster consistency criteria are defined: in the first one, called **Weak Correctness**, weak transactions get consistent database views, but weak transactions from other clusters may get different consistent views. With the second criteria, called **Strong Correctness**, every weak transaction, from every cluster, obtains the same consistent database view.

**Reporting/Co-Transactions**

This model, defined in Chrysanthi (1993), is an extension of open-nested transactions (transactions that allow their partial results to be seen by the outside world), in which two new types of transactions are defined to satisfy the requirements of a mobile environment. In particular, sharing of partial results between active transactions is allowed, and transactions running in the MSSs migrate from one to another as the mobile computer moves from cell to cell, in order to minimize communication costs.

A mobile transaction S is defined as a set of transactions (components) \( \{t_1, t_2, ..., t_n\} \), compensables or not, which can be further decomposed in other transactions, thus generating a multilevel hierarchy. These components can be of different types (in the following explanation, a two-level transaction is assumed):

- **Compensable transactions:** This type of transactions can commit even before transaction S. If S aborts, it executes a compensating transaction.

- **Non-compensable transactions:** Since this type of transactions does not have a compensating transaction, they cannot publish their effects at commit time. Therefore, they delegate this responsibility to transaction S, which will publish these results at the time it commits, or will cancel them if it aborts.

- **Reporting Transactions:** A reporting transaction \( t_r \) can share its partial results with S, delegating them at any point during its execution. Furthermore, if \( t_r \) is not compensable, it will delegate all the operations not reported previously at commit time.

- **Co-transactions:** These components are reporting transactions that act like co-routines, where control passes from one transaction to another when the partial results are shared. In other words, a transaction suspends its execution at delegation time, and then resumes from this point. Co-transactions cannot be run concurrently since they maintain their state between executions.

**COMPARISON FRAMEWORK**

After having described a set of transactional models, each with its particular support for mobility, we present a guide which should allow us to capture and compare the main characteristics to be taken into account in mobile transaction
models. In the first place, we define this guide, and then we use it to compare the models described previously.

**Definition**

In this section, we describe the main characteristics that must be taken into account in order to define a transactional model capable of being applied to a mobile environment, trying to take full advantage of its peculiarities.

**Relating Physical Aspects**

- **Mobility support**: The system must maintain the transaction’s execution even though the computer moves from cell to cell.
- **Disconnection support**: It must have the ability to execute transactions even when the mobile computer is disconnected, using caching techniques.
- **Replication support**: It should support the replication of information to have a lower communication cost.

**Relating Transaction Execution**

- **Place of execution**: Some models execute transactions at the MSS, while others execute them at the MHs.
- **Compensable transactions**: Compensable transactions allow the system to partially commit changes, since they can be later reverted if the whole transaction aborts.
- **Conditions of execution**: Allow the specification of conditions to be evaluated before, during, and after the execution of the transaction. These conditions can be based on time, other transaction’s starting or finishing, etc.
- **Efficient concurrency handling**: Due to the inherent characteristic of mobile environments, there is a higher probability that a transaction becomes LLT, so the concurrency must be handled efficiently, in order not to lock an object for a very long time (for example, Field Call). To bring more flexibility to active transactions, a mechanism for explicitly interchanging information must be provided.

**Relating the Model’s Adaptability**

- **Take advantage of the geographical localization of the mobile unit**: A transaction could specify location-dependent conditions, in order to modify its behavior. Furthermore, a handoff may not be always a significant event for the execution of a transaction; for example, if a mobile unit is changing between cells located in Texas, it may be of no difference to it, until it changes to a cell located in another state.
- **Final states**: Allow the specification of acceptable final states for the transaction.

**Relating Implementation Issues**

- **Take advantage of mobile techniques**: The system should use techniques that take into account the mobile environment’s characteristics. For example, the use of broadcasting, the analysis of the connection’s quality service in order to adapt the application to more reliable or more efficient networks, etc.
- **Take into account physical limitations**: The limitations imposed by mobile devices, such as low power processing, small memory, etc., should be taken into account, since they can have an impact in the way certain tasks are executed (the way of executing a transaction, visualizing information, recovering failures, etc).
- **Take advantage of the particular characteristics of a transaction**: For example, if a transaction is read-only or not, interactive or not, etc., to increase system’s flexibility and performance. For instance, executing a transaction in an MSS if it is not interactive, thus reducing the overhead of the procedures associated with handoffs.

### Table 2: Characteristics Related to Physical Aspects

<table>
<thead>
<tr>
<th>Transactional Support</th>
<th>Mobility Support</th>
<th>Disconnection Support</th>
<th>Replication Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kangaroo</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Moflex</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>IOT</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Weak/Strict in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data clusters</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Reporting / Co</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Comparison of Transactional Models**

The mobile transactional models are compared here, using the previously defined framework. For each section in the framework definition, a comparison table is presented and then an analysis is carried out. It is important to note that the items related to implementation issues are not considered here, since their study exceeds the scope of this work.

Since mobility is one of the main characteristics to take into account (see Table 2), it is important that transactional models can continue the execution of a transaction even if the user moves from one cell to another (handoff). Disconnection support is also an important issue since it allows the user to continue working while the MH is disconnected, which is a very frequent situation in mobile environments. Another useful characteristic, which can lower communication costs, is the ability to access replicated data, because in this way a subtransaction can use data located in the local machine, or else in a neighbor computer, instead of a possibly-far-away centralized database.

The Kangaroo, Moflex, and Reporting/Co-Transaction models all support mobility, since, during a handoff, the original base station must migrate the transaction’s state to the target MSS in order to follow the physical movement of the MH. On the other hand, none of these models allow the user to continue working while the mobile computer is not connected to the network.

IOT and Weak/Strict do support disconnections by allowing the disconnected computers to work with local cop-
Table 3: Characteristics Related to the Execution of the Transactions

<table>
<thead>
<tr>
<th>Transactional Model</th>
<th>Place of Execution</th>
<th>Compensating Transactions</th>
<th>Execution Conditions</th>
<th>Concurrency Handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kangaroo</td>
<td>MSS</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Moflex</td>
<td>MSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IOT</td>
<td>MH</td>
<td></td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Weak/Strict in data clusters</td>
<td>MH</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Reporting / Co-Transactions</td>
<td>MSS</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
</tbody>
</table>

Finally, replication is supported by the Weak/Strict model because of the possibility of defining data clusters, where the replicas of different clusters may have dissimilar consistency degrees.

In Table 3, a comparison of the models is shown, taking into account characteristics related to the execution of transactions. These transactions can be executed either in the MH, MSS, or both (if the model allows this). The decision as to where a transaction should be executed depends on the processing capacity of the computer, if the transaction is interactive or not, etc. For instance, if the MH is a small telephone or electronic address book, it would be better to execute transactions at the current MSS.

Compensating transactions increase updated data availability, since it is not necessary to wait for the commitment of the whole transaction in order to access partially generated data, which is useful in mobile environments, where transactions can take a considerable amount of time to run.

Although nearly all the models allow the definition of compensable transactions, and therefore the sharing of information before the commitment of the whole transaction, there are differences in the way this facility is provided. The Reporting/Co-Transactions model allows a transaction to share its partial results in an explicit way, while this is not possible in the other models.

As can be seen from this table, the Moflex model is the only one that allows the specification of transaction execution conditions. It should be noted that these conditions (called external dependencies) are only evaluated before the execution of the transaction.

All the studied models provide facilities for concurrency management (which are very important since there is a high probability that transactions become LTT). Though each one does it in different ways: IOT allows publishing partial results within the mobile computer; Weak/Strict allows this at a cluster level; Reporting-Co-Transactions allows the explicit publishing of these results; the rest of the models expose their changes globally.

In Table 4, we describe those characteristics related to the adaptability of transactional models.

Since, by definition, in a mobile environment a MH can roam through different geographical regions, it is important for the model to be able to adapt a transaction’s execution to the changing conditions generated by this roaming. The main characteristic to consider regarding adaptability is the one related to geographical location, that is, the ability of a transaction to change its behavior depending on the position of the MH.

Here, Moflex is the only model that supports preconditions related to the geographical localization of the mobile computer, and the specification of final acceptable states. Earlier, where this model is defined, you can see the adaptability provided for different applications’ requirements.

Classification of Mobile Models

From all the issues analyzed so far, a classification of mobile models arise, based on the presence or not of specific features. This classification allows us to pinpoint the existing voids and to highlight the aspects that deserve further work to reach a more capable architecture (Figure 2).

According to how deep their mobility support is, the models can be categorized as providing:

- **Total Mobility**, reflecting the ideal model with both hand-off and disconnection support.
- **Hand-Off Support**, which provides cell-switching detection, but cannot function if disconnected.
- **Disconnection Support**, which is only able to work disconnected and do not take into account the switch of cells.

![Figure 2: Classification of Mobile Models](image-url)
A second difference is established between those adaptable models, which allow establishing locality conditions, or non-adaptable ones. From a third point of view, models allow transactions to be executed only at MSSs, MHs, or at both.

All models should exhibit an efficient handling of concurrency and LIT because of the specific nature of mobile environments.

From this classification, we observe the lack of models that support Hand-offs and Disconnections simultaneously. Moreover, no current model allows for adapting the site of execution (i.e. MH, MSS or combined). This analysis shows that the state of the art is not optimal and points out several promising areas for future research (for example, the extension of current models or the definition of new ones).

CONCLUSIONS AND FUTURE WORK

In this work, we have defined a framework to be used as a tool for studying transactional models that take the maximum advantage of the particularities of mobile computing environments. We consider it a valuable tool not only to compare existing models, but also to validate new transactional models to come.

As a practical demonstration of its usefulness, we have studied some transactional models applied to mobile environments, comparing them with the help of this tool.

From this comparison and the classification made, we can say that none of the studied models covers all of the characteristics described in the framework. We consider that the essential items every transactional model should meet are mobility support and disconnection support. It can be seen that none of the models covers both characteristics, which leads us to think of the need to extend some of these models to support those items, or define a new mobile transactional model.

Furthermore, we consider it is necessary to create a flexible and dynamic simulator, so arbitrary configurations can be easily defined and measured, in order to compare the different approaches of solving the problems present in mobile environments. This will allow us, in a further step, to develop concrete implementations to be tested in real environments. To do this, existing implementations must be studied, this being a complex task given the recent definition and development of these systems.

REFERENCES


Conference on Information and Knowledge Management, Gaithesburg, MD, 371-378.


Roberto Hirsch has a five-year degree in Computer Sciences from the Facultad de Ciencias Exactas y Naturales (University of Buenos Aires, Argentina) and is currently a PhD student at the University of Buenos Aires. His main research interests are advanced transaction models and mobile computing. He is a member of ACM.

Andrés Coratella has a five-year degree in Computer Sciences from the Facultad de Ciencias Exactas y Naturales (University of Buenos Aires, Argentina) and is currently a PhD student at the University of Buenos Aires. His main research interests are advanced transaction models and mobile computing. He is a member of IEEE.

Miguel Felder received the Licenciado en Informatica degree from Escuela Superior Latinoamericana de Informatica (ESLAI), Universidad de Lujan, (Argentina) in 1988; the Master in Information Technology degree from CEFRIEL - Politecnico di Milano (Italy) and the Ph.D. degree in Computer Engineering from Politecnico di Milano (Italy). He has published several papers in international conferences and journals. Since 1994 he is Professor of Software Engineering, and co-leads a research group on such area at the Universidad de Buenos Aires. His research interests include formal languages for specification and design of real-time and concurrent systems, analysis, validation and testing of such systems. He is a partner of Pragma Consuliores (Argentina). He is member of IEEE, ACM, and the New York Academy of Sciences.

Eduardo Javier Rodriguez has a degree in Computer Sciences from the Escuela Superior Latinoamericana de Informática (University of Luján, Argentina) and is currently completing his MBA at the University Torcuato Di Tella. He has been a Guest Researcher at the IBM Almaden Research Center and has worked at IBM Argentina and YPF Argentina. Currently he is a Database Professor at the University of Buenos Aires and also the Information Systems and Technology manager at Banco Líneos Sudamericanos. He has published several articles in international conferences and journals. His main research interests are advanced transaction models, multimedia databases and CAAD systems. He is a member of ACM and IEEE.