

# Adaptable Mobile Transactions

Patricia Serrano-Alvarado\*, Claudia Roncancio, Michel Adiba, Cyril Labbé  
e-mail: Firstname.Lastname@imag.fr

LSR-IMAG Laboratory BP 72, 38402 St. Martin d'Hères, France

## Abstract

Mobile environments are characterized by high instability (e.g., variable bandwidth, disconnections, different communication costs) as well as by limited mobile host resources. Such characteristics lead to high rates of transaction failures and variable execution costs. We argue that to raise the rate of success of transactions and to have a minimal control on resources consumption, both application design and transaction management should be environment *aware*. This paper introduces an Adaptable Mobile Transaction model (AMT) which allows to define transactions containing several *execution alternatives* associated to a particular context. When an AMT is launched, the appropriated execution alternative is initiated depending on the current environment state. The goal is to *adapt* transaction execution to context variations. Our model relaxes atomicity and isolation properties but preserves conflict-serializability. An analytical study of AMTs shows that they can increase commit probability of transactions according to expected costs. This study can be a guideline to transaction designers. We describe also a middleware, called TransMobi, that supports mobile transactions involving several mobile and fixed hosts. TransMobi provides environment awareness and implements the AMT model through suitable protocols. Among these protocols we propose CO2PC to guarantee semantic atomicity.

\*Supported by the CONACyT scholarship program of the Mexican Government

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the 29th VLDB Conference,  
Berlin, Germany, 2003

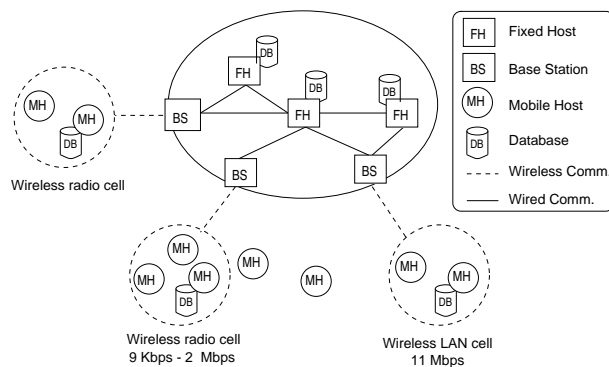


Figure 1: Mobile environment global architecture.

## 1 Introduction

The omnipresence of mobile devices such as cell phones, PDAs, smartcards, sensors and laptops, together with the development of different kinds of networks (local, wireless, ad-hoc, etc.) leads to a true mutation in the use, design and development of future information systems. Our work is related to the topics of ubiquitous and pervasive computing where technological improvements allow users to access data and perform transactions from any type of terminal and from anywhere using a wired or a wireless network. Applications that we have in mind cover a wide area. If we consider a client/server approach, where mobile and fixed hosts can be clients or servers, applications might be personal ones, where clients want to access public data (e.g., weather forecast, stock exchange, road traffic) or professional ones, where mobility is inherent (e.g., mobile vendors, health services, mobile offices, transport).

We consider a mobile computing environment with a network consisting of fixed and mobile hosts (FH, MH) [12], see Figure 1. MHs could be of different nature ranging from PDAs to personal computers. Shared data are distributed over several database servers running, generally, on FHs. MH may run database management system (DBMS) modules and may provide some services to other hosts. While in motion, an MH may retain its network connection through a wireless interface supported by some FHs

which act as Base Stations (BS). The geographical area covered by a BS is a cell. Each MH communicates with the BS covering its current cell. The process during which an MH enters into a new cell is called *hand-off*.

We make no specific assumptions about the database model (relational, object) but we place ourselves in a multidatabase environment assuming that data are managed by autonomous DBMSs.

We consider that applications in mobile environments are confronted to the limitations imposed by hardware such as: low and variable bandwidth, frequent disconnections, high communication costs, limited displays, battery autonomy, processing power and small data storage. All these limitations could lead to a lot of potential failure modes that can affect transaction processing. However, in some cases, these situations must be handled as “normal” or “weak performances” situations and not as failures. All these limitations lead to a lot of potential failure modes that can affect data management process (e.g., queries, replication, caching, transactions, etc.) [19]. However, in some cases, these situations must be handled as “normal” or “weak performances” and not as failures.

In this paper, we are particularly interested on mobile transaction management. Focusing on this notion, we adopt a quite general definition: *a mobile transaction is a transaction where at least one mobile host takes part in its execution*. For us, mobile transactions are long lived ones because of the probability of disconnections.

As an example, let us consider an e-shopping application that allows people to browse products in an e-mall, to select, to book and to buy items. We assume also that secured e-payment is available based on credit cards or *e-money*. Application execution, as a set of transactions, will not be the same if they are launched from a (fixed) terminal office, from a PDA while traveling in a train or from home using a laptop. Thus, under this context:

- transactions may succeed but with different execution times (bandwidth capacity is highly variable) and communication costs (prices vary among wireless network providers/technologies);
- energy consumption is affected by low bandwidth (more battery is consumed);
- transaction failure may occur due to unexpected disconnections or battery breakdown.

In traditional environments, application designers do not care about host and network characteristics. Nevertheless, in mobile applications we claim the necessity of being *environment aware* to overcome the infrastructure instability. To achieve this goal, it is necessary that application designers provide particular information concerning the mobile environment and the adequate policies to deal with it. This will help

transaction managers to adapt and improve transaction execution.

New models for non-traditional transactions have been proposed [8]. Several works concerning mobile transactions have also been introduced [11, 18, 23, 6, 13]. In the analysis made in [22, 21], we found out that the majority of these proposals are particular solutions oriented to specific application context. Moreover, most of these works do not take into account the importance of mobile environment variability and therefore environment awareness.

The contribution of this paper is twofold, we propose an Adaptable Mobile Transaction Model (AMT) and the middleware TransMobi to support it. The general idea of the AMT model is to define mobile transactions with several *execution alternatives* associated to a particular mobile environment state. When an AMT is launched, the appropriated execution alternative will be initiated depending on the current mobile environment. We argue that adapting transaction execution improves communication costs, response times and application availability, in short, the application’s quality of service. We made an analytical study that shows how AMT can increase transaction commit probability according to expected costs. This study can be a guideline to AMT designers and can be used as a base to build assistance tools for application developers.

The TransMobi middleware coordinates the execution of AMTs and is aware of the variations of the mobile environment state (e.g., bandwidth rate, communication cost, MH disconnections, MH available energy, etc). It uses suitable protocols to guarantee AMT properties. Among these protocols we propose (CO2PC) a Combination of an Optimistic approach with 2PC [10] to ensure *semantic atomicity*.

This paper is organized as follows: Section 2 presents the adaptable mobile transaction (AMT) model and related concepts. Section 3 gives an analytical study of AMT, whereas, Section 4 introduces the TransMobi middleware. Section 5 discusses related work and Section 6 concludes this article.

## 2 Adaptability for Mobile Transactions

### 2.1 Overview

Traditionally, transactions are defined independently of execution infrastructure. This approach is well suited for centralized and distributed systems where the execution characteristics have acceptable, predictable and controlled state variations. As the mobile environment is highly variable, transaction execution can be unpredictable.

We consider that in order to optimize resources, both transaction design and management should be made taking into account environment awareness. Our proposal is done along these lines. For each mobile

	Dimension	Values	Unit
WN	connection-state	<i>connected, disconnected</i>	
	bandwidth-rate	<i>high, medium, low</i>	kbytes/s
	communication-cost	<i>free, cheap, expensive</i>	Euros/time
MH	available-battery	<i>full, half, low</i>	hh:mm:ss
	available-cache	<i>full, half, low</i>	kbytes
	available-persistent-memory	<i>full, half, low</i>	kbytes
	processing-capacity	<i>high, medium, low</i>	Mhz/s
	estimated-connection-time	<i>t</i>	hh:mm:ss

Table 1: Mobile environment characteristics.

transaction we ask the application programmer to give execution alternatives suitable to a particular environment context. For example, a transaction distributed over a fixed and a mobile host will be launched if a good connection is available, whereas a centralized execution will be preferable if there is no connection, or if only a very low bandwidth is available.

To allow context aware transaction execution we propose an Adaptable Mobile Transaction (AMT) model. This model offers concepts to design mobile transactions, called AMTs. Generally speaking, the AMT is composed of at least one *execution alternative* involving one or several mobile or fixed hosts. Execution alternatives may be semantically equivalent. The successful execution of one of them, represents a correct execution of the AMT. An AMT also contains *environment descriptors* which express the state of the mobile environment required to execute each alternative. When an AMT is launched, the mobile environment state is checked and the appropriate execution alternative is chosen. If the environment state does not allow the execution of any alternative, the AMT execution may be deferred. So, an execution alternative will be triggered as soon as an acceptable environment state will be detected.

## 2.2 Mobile Environment Awareness

From our point of view, the mobile environment includes the wireless network (WN) and mobile hosts involved in mobile transactions. As said before, the instability of such an environment may affect transaction execution and impact resource consumption. The environment descriptors introduced here reflects the execution context and its potential state variations. Thus, transaction designers who know the application characteristics, declare the execution context and different execution alternatives<sup>1</sup>.

We define the environment descriptor (ED) as follows.

**Definition 1** *The Environment Descriptor*  $ED = \{dimension=value(s)\}$  contains a set of dimensions with their respective values at a given instant.

<sup>1</sup>This might put the burden on the application designer. Future work should be oriented to develop computer aided environments for application developers (see Section 3).

We consider the set of dimensions introduced in Table 1. In our opinion, they reflect the characteristics of the mobile environment that can affect transaction execution.

**Example 1**  $ED = \{connection-state = connected, bandwidth-rate = high, communication-cost = free, available-battery = half, available-cache = low, available-persistent-memory = low, processing-capacity = medium, estimated-connection-time = 00:40:00\}$ .

The set of dimensions may vary and user-defined dimensions (e.g. location indicator, specific “quality” of data) can be added. Notice that if several hosts are involved in a transaction, the ED may or may not specify the required state for each one.

## 2.3 The AMT Model

This model allows to describe mobile transactions having one or more *execution alternatives* (EA) associated each to an ED. EAs may take the form of any of the following execution types: the mobile transaction (1) is initiated by an MH and entirely executed on FHs, (2) is initiated by an MH/FH and entirely executed on an MH, (3) execution is distributed among MHs and FHs, and, (4) execution is distributed among several MHs. So, a wide variety of mobile transactions is addressed.

An EA contains a set of *component transactions* which must respect ACID properties. They can be traditional *flat*, *distributed* or *close nested* transactions. *Compensating transactions* may be associated to component transactions. They will be executed in case of failures. The EAs and the AMT are coordination units, data access is made only by component transactions. Making the analogy with multidatabases, component transactions are *local transactions* participating into *global transactions*.

Next, we present a semi-formal definition of the AMT model.

**Definition 2** *An adaptable mobile transaction is a triplet*  $AMT = (T, CT, \langle EA_k \rangle)$  where:

- $T = \{T_i\}$ ,  $1 \leq i$ , is a set of component transactions.
- $CT = \{CT_j\}$ ,  $1 \leq j \leq i$ , is a set of compensating transactions.
- $\langle EA_k \rangle$ ,  $k \geq 1$ , is a list of execution alternatives for AMT where  $EA_k$  has higher priority than  $EA_{k+1}$ .
- $EA_k = (ED_k, EP_k)$ , an execution alternative has an execution plan  $EP_k$  to be executed if the actual mobile environment satisfies the environment descriptor  $ED_k$ .
- $ED_k$  describes the appropriate environment state for the suitable execution of  $EP_k$ .

- $EP_k = \{(T_{ki}, HostId)\}$ ,  $T_{ki} \in T$ , is a set of pairs introducing a component transaction and the host where it has to be executed.

Let  $\mathcal{RD}$  be a relationship dependence over  $EP_k$ , such that,  $\forall(T_i, HostId_x) \in EP_k$ ,  $\forall(T_i, HostId_y) \in EP_k$ ;  $(T_i, HostId_x)\mathcal{RD}(T_i, HostId_y)$ .

HostId indicates a database and the MH/FH responsible of the execution. Such host must execute only one component transaction per execution alternative. Using  $\mathcal{RD}$ , several types of distributed transactions can be defined. Dependencies can be of different types (Chapter 10 of [8]). For simplicity reasons, we only consider here parallel execution and *begin on commit* (precedence constraint) dependencies.

Compensating transactions are semantically equivalent to physical rollbacks and are defined to undo already committed transactions. They recover semantically the database and avoid cascading aborts.

An analytical study of the AMT model is presented in Section 3.

### AMT example

Continuing with the example introduced in Section 1, consider an MH client with storage capacity, and an e-mail with two servers on the wired network: CatalogS and PurchaseS. The first site allows to query the store catalog whereas the second one takes purchase orders and payments.

We define the AMT AMTshopping with the following component transactions,  $T_i$ s:

- **GetCatalog** allows clients to get a catalog;
- **SelectItems** allows clients to select items from a local copy of the catalog;
- **Order-Pay** allows clients to send the purchase order and payment to the store;
- **AutoPay** allows clients to pay on the MH (with e-money) without contacting other host;
- **Order** allows clients to send a purchase order (no payment included);
- **Select-AutoPay** = **SelectItems** + **AutoPay**

In Table 2, AMTshopping proposes three execution alternatives to be triggered according to the environment state. In this example, the main dimensions determining the choice of an alternative concern the wireless network: connection availability, bandwidth and communication cost. The presence of the catalog on the MH is an application defined dimension. It takes the values *missing* (the catalog is not available on the MH) *present* (a version, probably out of date or incomplete, is on the MH) and *uptodate* (an up to date version is on the MH). Dimensions not appearing

EA	ED	EP
1	{catalog-state=uptodate}	{(SelectItems, MH), (Order-Pay, PurchaseS)}
2	{connection-state=connected, bandwidth-rate=high, medium, communication-cost=free, cheap} catalog-state=present, missing,	{(GetCatalog, CatalogS), (SelectItems, MH), (Order-Pay, PurchaseS)}
3	{connection-state=connected, bandwidth-rate=low, catalog-state=missing}	{(GetCatalog, CatalogS), (Select-AutoPay, MH), (Order, PurchaseS)}

Table 2: AMTshopping example.

in environment descriptors have no influence on the execution.

In this example, the priorities between EAs are determined by the communication cost. Executing  $EA_1$  is cheaper than executing  $EA_2$ . In  $EA_1$  the MH has an up to date catalog, this allows saving communication messages.  $EA_1$  may be launched even in disconnected mode and **Order-Pay** can be deferred until reconnection.  $EA_2$  will be launched provided that the communication quality is acceptable. In  $EA_3$ , the AMT is executed even under bad communication rates. The advantage of this alternative is that **Select-AutoPay** can be made in disconnected mode because the payment is in the MH. **Order** will be launched at reconnection.

Defining compensating transactions for this example can be easy. They would mainly include operations to cancel orders and refundings.

### 2.4 AMT Properties

An AMT can be considered at three different levels: the AMT itself, the execution alternatives and the component transactions (called transactions in the following). For the last ones we assume that ACID properties are guaranteed, nevertheless, as we will see later, durability is conditioned by the success of their corresponding execution alternative.

An execution alternative (actually the associated  $EP_k$ ) is a kind of *sagas* [16] containing a set of transactions which execution may be distributed among mobile and fixed hosts. The  $\mathcal{RD}$  defined inside the alternative describes the possibility of a parallel or/and sequential execution of component transactions. Global integrity constraints (IC) involving several  $T_{ki}$ s are not considered, consequently, consistency is not compromised.

#### Atomicity and isolation for EAs

Considering the restrictions of mobile environments, we propose to relax the atomicity property adopting semantic atomicity [15] (as in sagas).

**Definition 3**  $EA_k$  preserves semantic atomicity if either:

- All  $T_{ki}$ s defined in  $EA_k$  commit.

Property	$T_i$	$EA_k$	AMT
Atomicity	✓	Semantic atomicity	Semi-atomicity
Consistency	Verification of IC	Not considered	
Isolation	✓	Relaxed (local commits)	
Durability	✓ conditioned	Underlying DBMS	
Correctness	Serializability	Global serializability	

Table 3: Summary of AMT properties.

- All  $T_{ki}$ s defined in  $EA_k$  are compensated and/or rolled back.

The goal is to avoid blocking participant hosts and to allow MH disconnections. This is obtained with *local commits* where *partial* results are shared before the EA commit. The durability of locally committed transactions is conditioned to the commit of the EA. In case of an EA abortion, compensating transactions are used.

To address critical applications with non-compensatable transactions, resources are blocked. That is, when transactions terminate, resources are retained until a global decision (EA commits/aborts) is made. Hence, compensating transactions are not needed. If no participant commit locally, compensating transactions will be unnecessary and traditional atomicity is obtained.

The criteria used to control the correctness of concurrent execution alternatives is *global serializability*<sup>2</sup>. Global serializability states that transactions of each EA must have the same relative serialization order in their corresponding underlying DBMS.

### Atomicity and Isolation for AMTs

We mentioned in Section 2.3 that EAs defined in an AMT may be semantically equivalent. An AMT is correctly executed if one of its EAs is successfully executed. *Semi-atomicity* [25] is guaranteed for AMTs as follows.

**Definition 4** *AMT preserves semi-atomicity if either:*

- All  $T_{ki}$  defined in  $EA_k$  commit and all attempted  $T_{is}$ , of another alternative  $EA_l$  in the same AMT, are aborted or have their effects compensated.
- No partial effects of component transactions ( $T$ ) of the AMT remain permanent in the underlying DBMSs.

Serializability of AMTs is offered through the serializability of execution alternatives. Regarding the durability property, once the corresponding EA commits (and consequently the AMT), durability of component transactions is provided by the underlying DBMS. Table 3 summarizes AMT properties at all levels ( $T_i$ ,  $EA_k$ , AMT).

<sup>2</sup>In this paper, serializability is actually *conflict-serializability*.

Section 4 introduces the appropriated protocols to ensure these properties.

## 3 Analytical Study of AMT

The purpose of this section is to provide an analytical study of the capabilities provided by the AMT model. Using a probabilistic model, we consider several configurations which can lead to different execution alternatives. For each alternative, we evaluate its probability to be chosen and its execution cost.

The benefit expected from environment awareness and AMT adaptable facilities is highlighted. In addition, a method to guide AMT designers is provided. Those guidelines allow the designer to define the best AMT according to the quality of service required: low consumption without care of success probability or at the opposite success any time at any cost.

Firstly, a model of the environment is introduced, then a cost model is adopted and finally performance parameters such as the probability of successful initiation of an AMT and its cost are defined.

All along this section we use the *AMTshopping* example as an illustration of the analytical study.

### 3.1 Environment Parameters

As mentioned in previous section, mobile environment awareness allows to choose an  $EA_k$  if the environment state corresponds to the associated  $ED_k$ . Remember that ED is composed of a set of acceptable values for each dimension.

**Definition 5** *Let  $p_{ij}$  be the probability of the dimension  $i$  to be in the state  $j$ .*

In the resulting matrix  $P = (p_{ij})$  we have  $\forall i, \sum_j p_{ij} = 1$ . This matrix depends on the properties of an MH and of its environment.

**Example 2** *In AMTshopping the environment is seen as a set of four dimensions:*

	$j = 1$ Good	$j = 2$ Medium	$j = 3$ Bad
$i = 1$ Connection state	connected		disconnected
$i = 2$ Bandwidth rate	high	medium	low
$i = 3$ Communication cost	free	cheap	expensive
$i = 4$ Catalog State	Uptodate	present	missing

*The probability of being connected is given by  $p_{11}$  and the probability of having an uptodate catalog is given by  $p_{41}$ : Here is an example of such a matrix:*

$$P = (p_{ij}) = \begin{bmatrix} 0.8 & 0 & 0.2 \\ 0.7 & 0.2 & 0.1 \\ 0.5 & 0.25 & 0.25 \\ 0.2 & 0.3 & 0.5 \end{bmatrix}$$

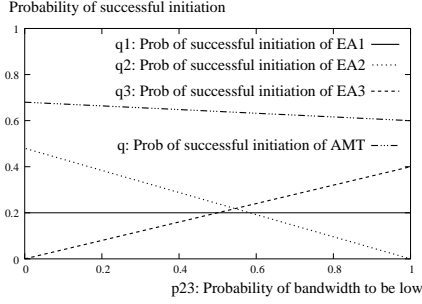


Figure 2: Successful initiation vs Probability of bandwidth to be low.

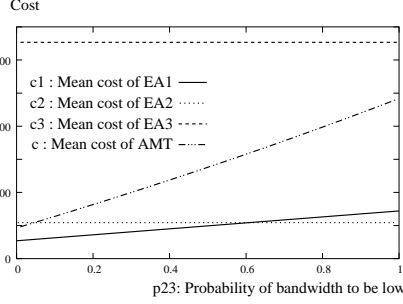


Figure 3: Mean cost vs Probability of bandwidth to be low.

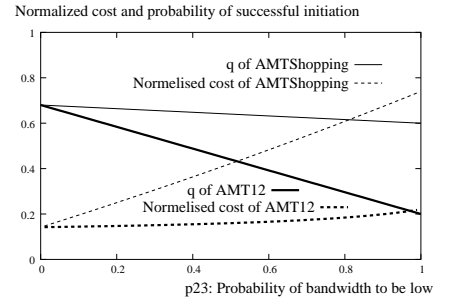


Figure 4: Successful initiation and mean cost vs bandwidth.

### 3.2 Descriptor Matrix

**Definition 6** For each  $EA_k = (ED_k, EP_k)$  we denote by  $\Delta^k$  the boolean matrix where:

$$\delta_{ij}^k = \begin{cases} 1 & \text{if the state } j \text{ of dimension } i \\ & \text{is acceptable for } EA_k \\ 0 & \text{otherwise} \end{cases}$$

If a dimension  $i$  does not appear in  $ED_k$  then, all its values are suitable for  $EA_k$ , so if  $i \notin ED_k$  then  $\forall j, \delta_{ij}^k = 1$ .

$EA_k$  has a higher priority than  $EA_{k+1}$  this allow us to assume without lost of generality that:

**Property 1** if a state of the environment is suitable for an  $EA_k$  it should not be suitable for an  $EA_{k'}$  of the same AMT. That is to say:  $\forall(k, k'), k \neq k', \exists i$  such that  $\forall j, \delta_{ij}^{k'} \neq \delta_{ij}^k$

**Example 3** The  $\Delta^k$  matrix for the three proposed alternatives of the AMTshopping example (see Table 2) are :

$$\Delta^1 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \Delta^2 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \Delta^3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

### 3.3 Cost Matrix

**Definition 7** Let  $C^k$  be the matrix where  $c_{ij}^k$  is the cost associated to the  $EA_k$  execution in the state  $j$  of the dimension  $i$ .

**Example 4** In the example we focus on communication cost. During execution of  $EA_1$  three wireless logical messages are necessary. First, a message for requesting the transaction (kind of login), then the purchase order message (component transaction Order-Pay) and finally, a message is received by the MH to confirm the order (acknowledgment). During execution of  $EA_2$  and  $EA_3$  an extra message asks for the catalog which is received through another message (GetCatalog).

Three different types of messages may be identified: small messages (login and ack), medium ones (Order-pay or Order) and large ones (GetCatalog).

To evaluate communication cost, we assume that:

- a large message is two times bigger than a medium one which is ten times bigger than a small one.
- if communication is free, sending a message has no cost. Sending a small message in the state cheap (resp expensive) has the cost 1 (resp 2).
- if bandwidth is high there is no extra cost. In the medium (resp low) state the cost doubled (resp multiply by four).

With respect to these assumptions, if the execution plan of  $EA_k$  sends  $n_s$  small,  $n_m$  medium and  $n_l$  large messages we have:

$$C^k = \begin{bmatrix} n_s + 10n_m + 20n_l & 2(n_s + 10n_m + 20n_l) & 4(n_s + 10n_m + 20n_l) \\ 0 & n_s + 10n_m + 20n_l & 2(n_s + 10n_m + 20n_l) \\ 0 & 0 & 0 \end{bmatrix}$$

And then:

$$C^1 = \begin{bmatrix} 0 & 0 & 0 \\ 12 & 24 & 48 \\ 0 & 12 & 24 \\ 0 & 0 & 0 \end{bmatrix} \quad C^2 = C^3 = \begin{bmatrix} 0 & 0 & 0 \\ 33 & 66 & 132 \\ 0 & 33 & 66 \\ 0 & 0 & 0 \end{bmatrix}$$

### 3.4 Mean cost of an AMT

If the execution plan of  $EA_k$  has to be launched, this would be done in the state  $j$  of the dimension  $i$  with the probability:

$$\frac{\delta_{ij}^k p_{ij}}{\sum_j \delta_{ij}^k p_{ij}}$$

So, the mean cost due to the dimension  $i$  associated to the execution of  $EA_k$  is given by:

$$c_i^k = \frac{\sum_j \delta_{ij}^k c_{ij}^k p_{ij}}{\sum_j \delta_{ij}^k p_{ij}}$$

**Example 5** With previous  $P$ ,  $\Delta^k$  and  $C^k$  we can compute:

$$c_2^3 = \frac{132 * 0.1}{0.1} = 132$$

$$c_3^3 = \frac{0 * 0.5 + 33 * 0.25 + 66 * 0.25}{0.5 + 0.25 + 0.25} = 24.75$$

**Definition 8** We define  $c^k$  the mean cost of an  $EA_k$  execution, as a function of all mean costs associated to dimensions:  $c^k = f(c_1^k, \dots, c_i^k, \dots)$ .

This function  $f$  may be chosen by the AMT designer in regards to its concern: communication time, battery or CPU consumption, etc.

**Example 6** If  $c_2^k$  is a time duration (seconds) and  $c_3^k$  money per time duration (cents/seconds), we can compute  $c^k$  in money by choosing  $f(c_1^k, \dots, c_i^k, \dots) = c_2^k * c_3^k$ . And so,  $c^1 = 162$  is the mean cost of an execution of the first alternative whereas  $c^2 = 444.3$  and  $c^3 = 3267$ .

### 3.5 Probability of successful initiation

**Definition 9** Let  $q^k$  be the probability of  $EA_k$  successful initiation. Because the probability that dimension  $i$  has an acceptable state for  $EA_k$  is given by  $\sum_j \delta_{ij}^k p_{ij}$ , we have:

$$q^k = \prod_i \left( \sum_j \delta_{ij}^k p_{ij} \right)$$

So the execution plan of an  $EA_k$  has a chance to be initiated ( $q^k > 0$ ) if and only if:  $\forall i, \exists j$  such that  $\delta_{ij}^k = 1$ .

**Definition 10** Let  $q_{AMT}$  be the probability of successful initiation of the AMT. Thanks to property 1 we have:  $q_{AMT} = \sum_k q^k$

**Definition 11** Let  $c$  be the mean cost of the AMT execution. Under the assumption that the state of the environment is stable during an AMT execution,  $EA_k$  is initiated with the probability  $q^k$  so the cost of the whole AMT is  $c^k$  with the probability  $q^k$ . Hence:

$$c = \frac{\sum_k c^k q^k}{\sum_k q^k}$$

**Example 7** As an example, we study the mean cost and the probability of a successful initiation of the AMTshopping in regards to the probability the bandwidth to be in the low state ( $p_{23}$ ):

$$P = \begin{bmatrix} 0.8 & 0 & 0.2 \\ (1 - p_{23})/2 & (1 - p_{23})/2 & p_{23} \\ 0.5 & 0.25 & 0.25 \\ 0.2 & 0.3 & 0.5 \end{bmatrix}$$

Figure 2 shows that the probability of a successful initiation never reaches 1. This is due to the fact that for certain states no alternative are defined. For example in the state *disconnected* with a *missing* catalog the transaction is not initiated.  $q^1$  is constant as it depends only on the probability for the catalog to be *uptodate*. This figure also shows that the probability of successful initiation of a single  $EA_k$  is less than the probability of successful initiation of the whole AMT.

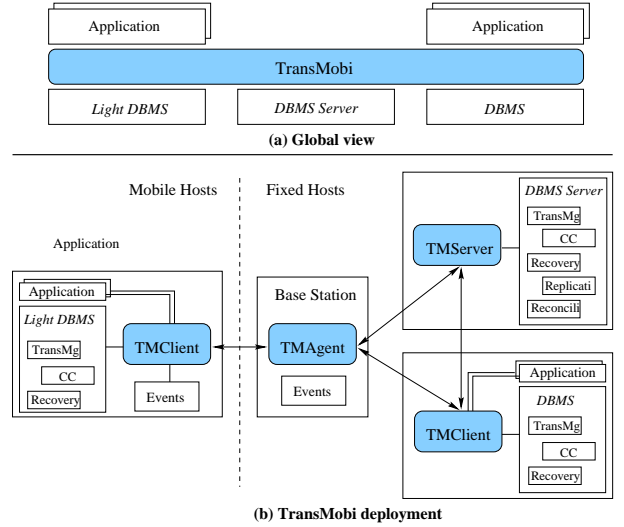


Figure 5: TransMobi global architecture.

Figure 3 shows that the mean cost of the AMT is growing as the probability of the bandwidth to be *low* is going up. This is due to the fact that (as seen in figure 2)  $EA_3$  is more often executed than  $EA_2$ .

Figure 4 shows performance parameters for AMTshopping and the same AMT without  $EA_3$ . This new AMT is called AMT12. It can be seen in this figure that AMTshopping has a better probability of successful initiation whereas AMT12 has a better mean cost.

## 4 TransMobi Middleware

We designed the TransMobi<sup>3</sup> middleware which supports adaptable mobile transactions (AMT). This section presents the architecture of TransMobi, its functionalities, the environment awareness and the protocols to ensure AMT properties.

### 4.1 Overview and Architecture

As a middleware between application code and existing DBMSs, TransMobi coordinates the execution of AMTs, see Fig. 5a. We assume the existence of DBMS functionalities on fixed and mobile hosts. TransMobi relies on them for the execution of component and compensating transactions ensuring ACID properties. To preserve DBMS autonomy, TransMobi uses standard interfaces.

TransMobi has the client/agent/server architecture showed in Fig 5b. A TMClient acts as an intermediate between the mobile application and underlying DBMS. It may run on FHs, called *fixed TMClients* or on mobile hosts, called *mobile TMClients*.

<sup>3</sup>Preliminary ideas on a Mobile Transaction Service (MTS) were introduced in [20].

TransMobi Agents (TMAgent) are on BSs<sup>4</sup> or on some FH able to communicate with MHs. TMAgents play a particularly important role, they (1) facilitate the interaction between mobile and fixed hosts acting as a representative (e.g., storing communication messages addressed to disconnected MHs, keeping track of particular MH information), (2) store transactional coordination information (e.g., transaction state, participating hosts), (3) follow MH moves (in case of hand-off, the *old* TMAgent sends the related MH information to the *new* TMAgent), and in particular, (4) manage network awareness.

A TMServer does some kind of global control, for instance, it maintains a global serializability graph. Furthermore, it interacts with database servers requesting for transaction execution on behalf of mobile applications. Besides transaction execution, servers may provide replication and reconciliation features, but it is not our objective to address these issues here.

Generally speaking, AMT application requests are intercepted by a TMClient. It chooses the best  $EA_k$  to be initialized according to the mobile environment state and distributes the  $T_{ki}$ s to the appropriate hosts. Distribution is made according to relationship dependencies over  $EP_k$  (see Section 2.3). Mobile TMClients send the  $EA_k$  to their TMAgent which takes in charge the distribution.

## 4.2 Adaptability and Environment Awareness

TransMobi takes advantage of the adaptability potential of AMTs (see Section 2). TMClients take in charge the comparison of the actual mobile environment state with environment descriptors. This comparison is made in the order defined by the EA list.

Environment awareness can be implemented with events. Table 4 defines the event types that reflect the variations of the mobile environment. Related information can be attached to events. Notifications are made in *pull* and *push* manners as indicated in Table 4. This approach facilitates immediate and deferred triggering of AMTs.

The environment state concerns the wireless network and MHs. A mobile TMClient knows its state (see Table 1) and perceives the WN one. For *fixed TMClients* the state of the mobile environment is provided by TMAgents which perceive the WN state and the state of MHs in their area. An MH may publish its state completely or partially. It may include physical characteristics but also particular processing functions and available data. Corresponding events can be pushed by the MH or pulled by a TMAgent.

Environment events may be provided directly from the operating system, (e.g., module *Odyssey* in [17])

<sup>4</sup>Generally, wireless network providers do not allow to install software on their BSs, we believe that this will change in the future.

	Event type	Attached information	Notification mode
WN	<i>e-connection</i>	MH id	Immediate push
	<i>e-disconnection</i>	MH id	
	<i>e-hand-off</i>	New BS id	
	<i>e-bandwidth-rate-changes</i>	Bandwidth rate	
	<i>e-communication-cost-changes</i>	Communic. cost	
MH	<i>e-available-battery-changes</i>	Available battery	Pull before choosing an $EA_k$
	<i>e-available-cache-changes</i>	Available cache	
	<i>e-available-persistent-memory-changes</i>	Available persistent memory	

Table 4: Event types for environment awareness.

or be handled through an event service (e.g., ADEES [4]).

As said in section 2.2, environment descriptors can include application defined characteristics. This allows adaptability to a large set of *dimensions*.

## 4.3 TransMobi Protocols

This section presents the protocols to provide the properties of AMTs (see Section 2.4), particularly, semantic atomicity and global serializability.

### 4.3.1 Atomicity Protocol

#### CO2PC Protocol

Our objective is to provide semantic atomicity for execution alternatives (EAs), by allowing participants to perform either optimistic local commit or non-optimistic commit. We attempt to increase the flexibility of participants (particularly MHs), thus compensable transactions can be committed locally in an optimistic manner, whereas non-compensable ones have to wait for the global  $EA_k$  decision.

We propose CO2PC, a Combination of an Optimistic approach and 2PC [10]. CO2PC works as follows: all participants send a *vote (commit/abort)* to the coordinator which takes the global decision and sends it to all participants. An unanimous vote for commit leads to a commit decision, otherwise the decision is aborted.

Participants are TMClients or TMServers working with a local DBMS. The CO2PC coordinator must be a participant FH. If there is no FH participant then a TMAgent will play this role. MH should not be coordinator because they are subject to frequent disconnections and have restrained resources for logging and communication.

A participant making optimistic commit (called *opt-participant*) executes its transaction  $T_{ki}$  and commits/aborts it unilaterally. Then, it sends its *vote* to the coordinator. If the global decision is *commit*, *opt-participants* are done; if it is abort they have to launch compensating transactions. Fig. 6a illustrates CO2PC with two *opt-participants*.

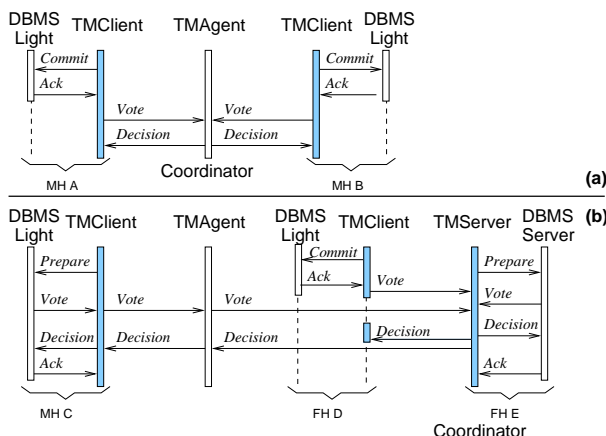


Figure 6: CO2PC examples.

A participant making non-optimistic commit (called *non-opt-participant*) executes the 2PC protocol locally<sup>5</sup>. The TMClient (or TMServer) acts as coordinator for this 2PC and the unique participant is the underlying DBMS. In Fig. 6b, MH C and FHE are *non-opt-participants*. The TMClient/TMServer initiate the 2PC with their respective DBMSs (*prepare* message). Each DBMS sends its *vote* to the TMClient/TMServer which forwards it to the CO2PC coordinator. If the vote is *abort*, the participant aborts the  $T_{ki}$  unilaterally, otherwise  $T_{ki}$  enters into a *prepared* state. When the decision of the CO2PC coordinator arrives to participants, the corresponding *commit/abort* is executed on the underlying DBMS. Notice that the 2PC part of the protocol is made on the same host and does not require messages through the wireless network.

Fig. 6b shows CO2PC with one *opt-participant* and two *non-opt-participants*. Only *vote* and *decision* messages are transmitted over the wireless network (2 messages per participant).

In order to tolerate MH disconnections, to limit undefined blocking and to break eventual deadlocks, CO2PC uses timers. A timeout is assigned to each  $T_{ki}$  and to  $EA_k$ s. Commits must be done before timeouts expire. Timeouts could be renegotiated for example, when an MH disconnection period is known in advance. When a participant *votes commit*, it deactivates its timer and has to wait for the global decision. If the  $EA_k$  timeout expires before the coordinator has all *votes*, the  $EA_k$  is aborted or compensated (consequently the AMT).

2PC is used by CO2PC because for non-compensable transactions, resources must be retained until a global commit/abort. 2PC provides this state, besides it ensures atomicity, produces *recoverable* systems, *avoids cascading aborts* (when it is combined with two-phase locking) and its interface is supported by the majority of DBMSs.

<sup>5</sup>A non-compensable transaction cannot be executed on a DBMS not supporting 2PC.

Hosts	Log information
Opt participant	<i>Commit, Acknowledge</i>
Non-opt participant	<i>Start2PC, Prepare</i>
All participants	<i>Vote, Decision</i>
Coordinator	<i>StartCO2PC, Vote, Decision</i>

Table 5: Logging during CO2PC.

Notice that 2PC is not used for the global coordination of AMT atomicity. In our opinion 2PC is not the best choice for that because it requires a high rate of messages (four per participant) and it blocks the entire global transaction for an undefined period of time, until a global *decision* is made. This is particularly constraining in the mobile context. Moreover, light MHs may not support the 2PC interface or may not have enough resources to spend on it. Thus, trying to address several kind of transactions (non-compensable as well as compensable ones) and different types of MHs (with limited and unlimited resources) we propose CO2PC.

## Recovery

To provide recovery, CO2PC records its progressing steps in the coordinator and participant logs. Since failures and disconnections can occur at any moment, logging information should be forced to be written (i.e., flushed into a stable storage) before sending messages. For MHs, CO2PC information will be logged in the TMClient log as well as in the corresponding TMAgent. Physical storage of MH is not considered stable because MHs are subject to lost, damages and undefined disconnections. If the MH is disconnected, logging in the TMAgent will wait for a reconnection.

Table 5 shows information to be logged by the participants and the coordinator. In order to preserve DBMS autonomy, we make no assumptions about the logging policy used by underlying DBMS. Besides, logs of transaction operations are not shared.

Once compensating transactions are initiated they should complete successfully. This characteristic is called *persistence of compensation* [16] and is ensured by resubmitting compensating transactions until they commit. If persistence of compensation is guaranteed there is no need to use an atomic commit protocol to obtain the atomicity of the set of  $CT_{ki}$ s of an  $EA_k$ .

### 4.3.2 Serializability Protocol

Concurrency control algorithms synchronize concurrent transactions by ordering their conflicting operations such that a serialization order can be maintained. At a global level ( $EA_k$ ), determining conflicts is not easy, particularly when DBMSs do not share local management information. Two global transactions without direct conflicts may however conflict if there exists local transactions that do not belong to  $EAs$ . These *indirect conflicts* are known at global level only

if DBMS autonomy is relaxed. In our context, DBMS autonomy is respected.

To provide global serializability two conditions should hold. (1) local DBMS should produce serializable and recoverable schedules. (2) each local DBMS should maintain the subtransactions relative order determined at global level.

Thanks to the assumptions that we made (underlying DBMSs preserve ACID properties), the first condition holds. To guarantee the second condition, we propose to use the Optimistic Ticket Method (OTM) [9]. Several reasons motivate our choice:

- OTM maintains global serializability.
- Autonomy of underlying DBMS is preserved.
- No extra messages are necessary, this is important for mobile environments.
- Concerning traditional approaches, using homogeneous concurrency control protocols does not guarantee global serializability. Exception is using *strong 2PL* [1], however this approach does not allow early unlocking (i.e., optimistic commit) and is therefore not suitable for mobile environments.
- Sequential execution of global transactions preserves global serializability only when it is combined with *timestamp ordering* [1]. This approach is not suitable because it reduces concurrency, and applied to mobile environments may generate undefined waiting periods of time for the entire system.

The main idea in OTM is that indirect conflicts are converted into direct ones. For this, each underlying DBMS has a special data item, called *ticket*. Each global subtransaction reads the ticket at the host where it executes (there is one ticket by host), increments it, and writes it into the database. At global level, a *global serialization graph* (GSG) is maintained.

TransMobi uses OTM at  $EA_k$  level and combines it with CO2PC. The GSG is maintained by a global *serializability manager* (SM), located on a TMServer. When an  $EA_k$  is initiated, the SM creates an  $EA_k$  node in the GSG. Each  $T_{ki}$  increments the ticket and when the execution terminates the participant sends the *vote* and the ticket value to the CO2PC coordinator. The latter sends the ticket value to the SM which inserts edges between  $EA_k$  and recently committed  $EA_j$ s. Edges must reflect the relative ticket order [9]. As soon as a cycle is generated, the SM advises the coordinator which aborts the  $EA_k$ . If no cycle is generated, after all edges have been inserted, the coordinator commits  $EA_k$ .

#### 4.4 Disconnection Management

A key property of mobile computing is to support disconnections as a normal state of the system. For this,

TransMobi gives support to disconnections at a transactional level. That is, during the transaction execution, participant MHs may disconnect. There are two kinds of disconnection, the programmed one and the unexpected one. The former is requested by the user and the latter is a consequence of some mobile environment variations (e.g., MH is out of the area covered by the wireless network, MH battery is exhausted).

TransMobi protocols tolerate both kind of disconnections. In CO2PC, after the *vote* is sent, an MH may disconnect temporally. In that case, the coordinator *decision* is logged in a TMAgent and sent to the MH when reconnection occurs. In the same way, the timeout assigned to each transaction ( $T_i$ ) allows the MH to disconnect provided that the *vote* is sent before the timeout expires. Both, *opt* and *non-opt-participants* may disconnect. Nevertheless resources of *non-opt-participants* will remain blocked until reconnection (their 2PC is not finished).

Concerning the OTM protocol, disconnections are not a problem because the serialization order is dictated by the ticket updating. This is completely independent of the MH connection state. The unique requirement is to send the ticket value to the coordinator. Nevertheless, the disconnection duration has an important consequence. The GSG is constructed in the commit order of the execution alternatives (actually the AMTs). Thus, if there is a conflict, the alternative trying to be serialized will be aborted. That is, the abort/success probability is related to the time that a transaction takes to be serialized.

## 5 Related Work

The AMT model was inspired from DOM [3] and Flex [7] where the general idea is to define *equivalent* transactions for being executed in case of failures. The DOM transaction model allows *close* and *open nested* transactions. Compensating transactions as well as contingency ones can be specified for being executed if a given transaction fails. In the Flex transaction model, contingency transactions are defined in terms of *functionally equivalent* transactions. A failure order is defined where the execution of a transaction depends on the failure of another one. Unlike the AMT, DOM and Flex transactions are not defined to deal with mobile environments and the notion of context awareness is not considered.

The panorama of mobile transactions is vast. A detailed analysis of several works is given in [22, 21]. Like in KT [6], MDSTPM [24], Pre-serialization [5] and Moflex [13], our work addresses mobile transactions as multidatabase ones. Unlike AMT, none of these propositions consider execution on MHs. Besides, in general, the adaptability vision is very limited, almost all studied systems adapt their behavior to support disconnections. Only Moflex addresses execution adaptability when hand-off occur. On the contrary,

our proposition can take into account any defined environment characteristics.

Concerning atomicity protocols for mobile environments we identify UCM [2] and TCOT [14]. [2] defines an Unilateral Commit Protocol (UCM) supporting off-line executions (on MHs) and disconnection. Several assumptions are made, for instance, all participants are constrained to use strict 2PL and at commit time the effects of all local transactions are logged on stable storage. UCM guarantees atomicity and durability, nevertheless, transactions are blocked until global commit and logs of updates must be flushed (on a FH) at each transaction commit. In TCOT, participants may commit locally before the global commit using timeouts. If global commit fails, compensating transactions are executed. To be able to commit independently, a mobile participant must send its log of updates to the coordinator. This protocol does not take disconnections into account. The protocol proposed here (CO2PC), makes no assumption about concurrency control techniques used by the participants. Furthermore, CO2PC reduces the use of the wireless network (2 messages by participant are necessary) and allows cohabitation of compensable and non-compensable transactions.

## 6 Conclusion and Future Work

This paper focused on mobile transaction management. We proposed an adaptable transaction model (AMT) and also an appropriate middleware (TransMobi) to support it. A first prototype of TransMobi has been developed. It uses the virtual machine Personal Java and PointBase (as DBMS) on MHs (of type Ipaq H3850 of Compaq), over a WLAN 802.11b as network. This prototype allowed to test some features but experiences in a more variable environment should be done.

The AMT model, introduces environment awareness in the design of mobile transactions. The selection of an *execution alternative* is based on the current mobile environment state. Execution alternatives may involve several DBMS on mobile or fixed hosts. We developed an analytical study that showed how AMTs can increase commit probability according to expected costs. This probabilistic approach can be the basis to build assistance tools for mobile application developers. To guarantee semantic atomicity and global serializability of AMTs, we proposed the CO2PC commit protocol and adopted the OTM serializability protocol. These protocols preserve DBMS autonomy, save message exchanges over the wireless network and allow mobile hosts disconnections.

In our proposal, environment awareness can be implemented with events. Dimensions included in *environment descriptors* (ED) are characteristics of the mobile environment and/or application defined ones. This approach is quite general and can be applied to

other variable contexts. In particular, applications needing location dependent data/process could be addressed by including in the ED the appropriate dimension (e.g., City, values = {Grenoble, Lyon, Paris}).

Our future work addresses several topics. The first one concerns durability. Our current solution does not force logging transfers from MHs to FHs. This limits the fault tolerance of our approach. We will study this point in order to provide a good trade-off between autonomy, durability and performances. The second topic concerns execution adaptability. The idea is to allow a dynamic change of execution strategy avoiding global abortions and reusing work already done.

## Acknowledgments

We wish to thank the members of the NODS project (<http://www-lsr.imag.fr/Les.Groupes/STORM/>) for their feedbacks all along this research, particularly to Stéphane Drapeau, Edgar Benitez-Guerrero and Nagapraveen Jayaprakash for their valuable remarks on earlier versions of this article.

## References

- [1] P. A. Bernstein, V. Hadjilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.
- [2] C. Bobineau, P. Pucheral, and M. Abdallah. A Unilateral Commit Protocol for Mobile and Disconnected Computing. In *12th Conference on Parallel and Distributed Computing Systems (PDCS'00)*, Las Vegas US, August 2000.
- [3] A. Buchmann, M. T. Ozsu, M. Hornick, D. Georgakopoulos, and F. A. Manola. A Transaction Model for Active Distributed Object Systems. In *Database Transaction Models for Advanced Applications*, 1992.
- [4] C. Collet, G. Vargas-Solar, and H. Graziotin-Ribeiro. Open Active Services for Data-Intensive Distributed Applications. In *IDEAS*, Yokohama-Japan, 2000.
- [5] R. A. Dirckze and L. Gruenwald. A Pre-Serialization Transaction Management Technique for Mobile Multidatabases. *Mobile Networks and Applications*, 5(4), 2000.
- [6] M. H. Dunham and A. Helal. A Mobile Transaction Model that Captures Both the Data and the Movement Behavior. *ACM/Baltzer Journal on special topics in mobile networks and applications*, 2, 1997.
- [7] A. Elmagarmid, Y. Leu, and M. Rusinkiewicz. A Multidatabase Transaction Model for INTERBASE. In *International Conference on VLDB*, August 1990.
- [8] A. K. Elmagarmid. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann Publishers, 1992.
- [9] D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth. Using Tickets to Enforce the Serializability of Multidatabase Transactions. *IEEE Transactions on Knowledge and Data Engineering*, 6(1), February 1993.
- [10] J. Gray. Notes on Database Operating Systems. *Operating Systems: An Advanced Course*, LNCS, Springer Verlag, 60, 1978.

- [11] J. N. Gray, P. Helland, P.O'Neil, and D. Shasha. The Dangers of Replication and a Solution. In *ACM SIGMOD Conference on Management of Data*, Canada, 1996.
- [12] T. Imielinski and B. R. Badrinath. Wireless Mobile Computing: Solutions and Challenges in Data Management. Technical report, Rutgers University, New Brunswick, 1993.
- [13] K. Ku and Y. Kim. Moflex Transaction Model for Mobile Heterogeneous Multidatabase Systems. In *10th International Workshop on Research Issues in Data Engineering*. IEEE, 1998.
- [14] V. Kumar, N. Prabhu, M. H. Dunham, and A. Y. Seydim. TCOT- A Timeout-Based Mobile Transaction Commitment Protocol. *IEEE Transactions on Computers*, 51(10), 2002.
- [15] H. G. Molina. Using Semantic Knowledge for Transaction Processing in a Distributed Database. *ACM Transactions on Database Systems*, 8(2), June 1983.
- [16] H. G. Molina and K. Salem. SAGAS. *ACM SIGMOD International Conference on Management of Data*, May 1987.
- [17] B. D. Noble and M. S. et al. Agile Application-Aware Adaptation for Mobility. In *Sixteen ACM Symposium on Operating Systems Principles*, Saint Malo, France, 1997.
- [18] E. Pitoura and B. Bhargava. Data Consistency in Intermittently Connected Distributed Systems. In *Transactions on Knowledge and Data Engineering*, November 1999.
- [19] E. Pitoura and G. Samaras. *Data Management for Mobile Computing*. Kluwer Academic Publishers, 1998.
- [20] P. Serrano-Alvarado. Defining an Adaptable Mobile Transaction Service. In *Post-proceedings of the EDBT Workshops, Selected Papers, LNCS*, volume 2490. Springer, 2002.
- [21] P. Serrano-Alvarado, C. Roncancio, and M. Adiba. A Survey of Mobile Transactions for DBMS. *Submitted paper, 35 pages*, 2003.
- [22] P. Serrano-Alvarado, C. L. Roncancio, and M. Adiba. Analyzing Mobile Transactions Support for DBMS. In *International Workshop on Mobility in Databases and Distributed Systems in DEXA*, Munich, Germany, September 2001.
- [23] G. D. Walborn and P. K. Chrysanthis. Transaction Processing in PRO-MOTION. In *14th ACM Annual Symposium on Applied Computing*, San Antonio Tx, 1999.
- [24] L. H. Yeo and A. Zaslavsky. Submission of Transactions from Mobile Workstations in a Cooperative Multidatabase Processing Environment. In *Conference on Distributed Computing Systems*, June 1994.
- [25] A. Zhang, M. Nodine, B. Bhargava, and O. Bukhres. Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems. In *ACM SIGMOD*, May 1994.