A mobile transaction model that captures both the data and movement behavior

Margaret H. Dunham[a,*], Abdelsalam Helal[b,**] and Santosh Balakrishnan[c]

[a] Department of Computer Science and Engineering Southern Methodist University, Dallas, TX 75275, USA
[b] MCC, 3500 West Balcones Center Dr., Austin, TX 78759-6509, USA
[c] Department of Computer Science, University of Texas at Arlington, Arlington, TX 76019, USA

Unlike distributed transactions, mobile transactions do not originate and end at the same site. The implication of the movement of such transactions is that classical atomicity, concurrency and recovery solutions must be revisited to capture the movement behavior. As an effort in this direction, we define a model of mobile transactions by building on the concepts of split transactions and global transactions in a multidatabase environment. Our view of mobile transactions, called Kangaroo Transactions, incorporates the property that transactions in a mobile computing system hop from one base station to another as the mobile unit moves through cells. Our model is the first to capture this movement behavior as well as the data behavior which reflects the access to data located in databases throughout the static network. The mobile behavior is dynamic and is realized in our model via the use of split operations. The data access behavior is captured by using the idea of global and local transactions in a multidatabase system.

1. Introduction

Recent advances in hardware technologies such as portable computers and wireless communication networks have led to the emergence of mobile computing systems. Examples of mobile systems that exist today include travelers carrying portable computers that use cellular telephony for communications, and transporting trains and airplanes that communicate location data with supervising systems. In the near future, it is expected that tens of millions of users will carry a portable computer and communicator that uses a wireless connection to access a worldwide information network for business or personal use. Access by users with mobile computers to data in the fixed network will involve transactions. However a transaction definition in this environment varies from that within a centralized database or even a distributed environment. In addition, limitations of the wireless and nomadic environment place a new challenge to implementing efficient transaction processing using classical transaction models. Disconnection is the major obstacle. Moreover, the long-lived nature of transactions issued by mobile users exposes transactions to a larger number of disconnections. Such frequent disconnections give rise to reliability (instead of availability) being the primary system requirement for transaction processing in the mobile environment. A new transaction model is therefore needed to cope with this new requirement.

In addition to differences in the environment, a mobile transaction executes differently from a distributed transaction. The former hops through a collection of visited sites, while the latter communicates with a collection of remote sites, and starts and ends at the same originating site. The implication of this difference is that protocols for coordination of transaction termination (be it for commit or abort) must be revisited. In the distributed transaction case, it suffices for any remote site to send to (and maybe receive acknowledgment from) the originating site to participate in the termination protocol. In the mobile environment, however, the originating site could be different from the site at which the transaction finishes. And even networking protocols that provide relocation transparency (like Mobile-IP) may not be able to guarantee delivery to the originating site. This is because the mobile host may be disconnected due to roaming off the area, damaged, or stolen.

A new transaction model for the mobile environment is needed. It should address the movement behavior of transactions. As mobile transactions "hop" from a stationed host to another, the state of the transaction and its progress must also move, transparently. This transaction model must still support concurrency, atomicity, and recovery. It should also handle frequent disconnection failures and maintain mutual consistency among replicated data.

The new model is not expected to be purely ACID. Not that ACID is not enforceable, but because it is expected that ACID will be enforced using too many aborts, resulting in a system that is perfectly consistent, but that gets only a small fraction of useful work done. To cope with all the disconnections, and to accommodate for different network characteristics, the new transaction model should provide a spectrum of correctness criteria ranging from ACID to unrestricted access. The case for the usefulness of unrestricted access can be easily made in the mobile environment. For

* This research was partially supported by the National Science Foundation under Grant Number INT-9417907 and by a Massive Digital Data System (MDDS) effort sponsored by the Advanced Research and Development Committee of the Community Management Staff. Part of this research was performed while Margaret Dunham (then Margaret Eich) was on sabbatical at the University of Queensland in Brisbane, Australia.
** This research was partially supported by Hughes Research Laboratories under Grant Number 906356.

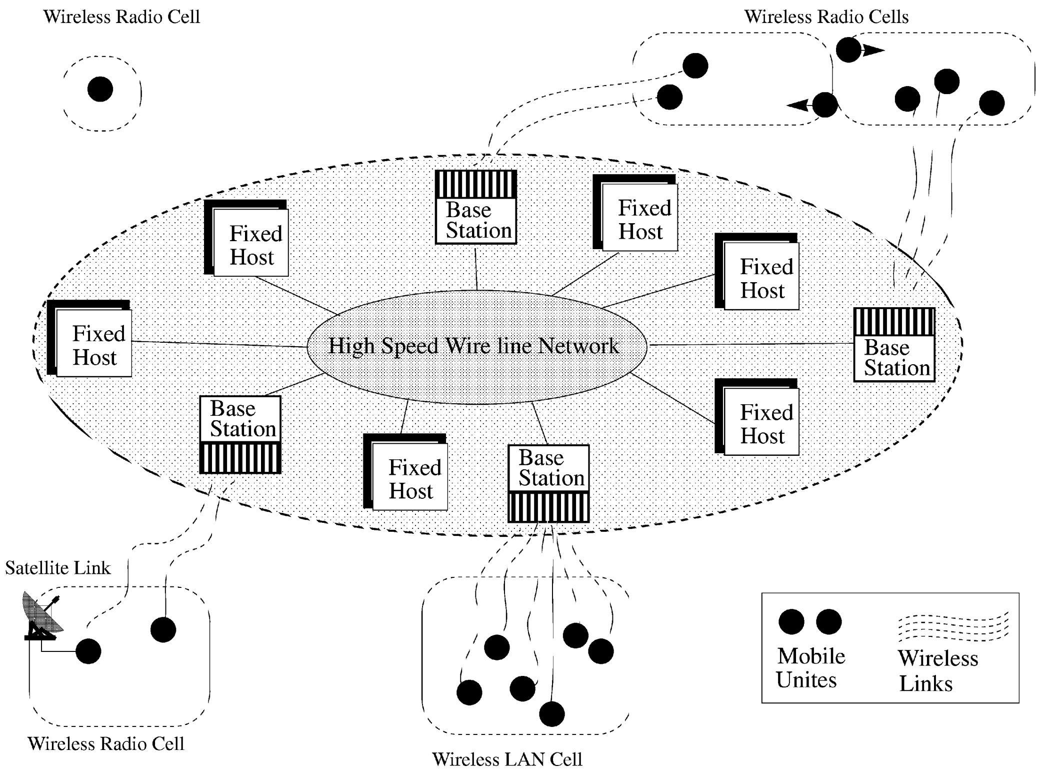


Figure 1.

example, given the scarcity of resources in the mobile environment, a conflicting access to a data item (caused by a concurrent update) should be allowed. Returning "dirty" data tagged with appropriate warnings is much more useful than returning an ABORT message, especially, that bandwidth and delays will be incurred anyway.

In this paper, we present a transaction model that can cope with the hopping behavior of the mobile user.

We build on two previously proposed transaction models: global transactions in a multidatabase environment [2,3,23], and split transactions [17]. In the next section we provide an overview of the mobile computing environment. In section 3 we examine the requirements which a mobile transaction should satisfy. Section 4 introduces our model of mobile transactions. In section 5 we briefly review related transaction research, and finally in section 6 we summarize the paper and list future research.

2. Mobile database environment

Figure 1 shows the existing (and widely accepted) architectural model of a system that supports mobile computing [7,11–15,21,24]. The model consists of stationary and mobile components. A *Mobile Unit* is a mobile computer which is capable of connecting to the fixed network via a wireless link. A *Fixed Host* is a computer in the fixed network which is not capable of connecting to a mobile unit. A *Base Station* is capable of connecting with a mobile unit and is equipped with a wireless interface. They are also known as *Mobile Support Stations*. Base stations, therefore, act as an interface between mobile computers and stationed computers. The wireless interface in the base stations typically uses wireless cellular networks. Ericson GE Mobidem is an example of a cellular network that uses packet radio modems. The wireless interface can also be a local area network, of which NCR WaveLan is an example. Current cellular technology offers a limited bandwidth in the order of 10 Kb/s (Mobidem offers 8 Kb/s), whereas current wireless LAN technology offers a bandwidth in the order of 10 Mb/s (WaveLan offers 2 Mb/s). While these numbers are most likely to change in the future, it is safe to assume that the network bandwidth will remain a major limitation and a performance bottleneck for nomadic system design in the near future. In addition, cost is also a factor in the development of non-LAN applications. In this paper, we use the existing mobile computing model in figure 1 as a foundation ground on top of which we define our mobile transaction model.

We have previously defined a reference model for mobile database systems [10]. Our reference model consists of three layers: the source system, the data access agent, and the mobile transaction. Table 1 provides an overview of this reference model.

The *Source System* represents a collection of registered systems that offer information services to mobile users. Examples of future services include white and yellow pages services, mail-enabled applications, public information systems including weather, navigation, stock quotes, and company-private database/information systems. In our model, a system in this layer could be any existing stationary system that follows a client-server or a peer-to-peer

Table 1
Reference model layers.

Layer	Location	Purpose
Source System	Fixed Host Base Station Mobile Unit	Provide services defined by specific software
Data Access Agent	Base Station	Coordinate access to data in source system and facilitate recovery. Manage mobile transaction
Mobile Transaction	Base Station Mobile Unit	Grouping of operations needed to perform user request initiated at a Mobile Unit

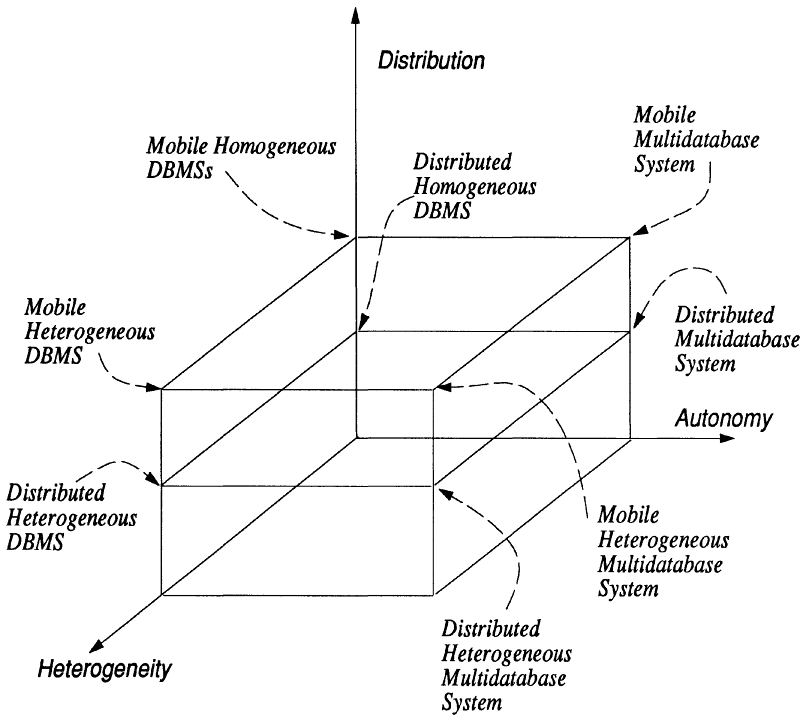

Distribution
Mobile Homogeneous DBMSs
Distributed Homogeneous DBMS
Mobile Multidatabase System
Mobile Heterogeneous DBMS
Distributed Multidatabase System
Autonomy
Distributed Heterogeneous DBMS
Mobile Heterogeneous Multidatabase System
Heterogeneity
Distributed Heterogeneous Multidatabase System

Figure 2.

design. A distributed database system is a perfect example. Data in the source system(s) is accessed by the mobile transaction through the *Data Access Agent* (DAA). Each base station hosts a DAA. The DAA does not maintain location information of the mobile user, even though it will need to use this information. When it receives a transaction request from a mobile user (we call this a mobile transaction), the DAA forwards it to the specific base stations or fixed hosts which contain the needed data and source system component. In the simplest case, the DAA would forward all requests to one multidatabase host system which would then determine where to execute the subtransactions. A more general situation would be where each DAA contains location tables which indicate, by transaction or subtransaction, the correct MDBS (Multidatabase System) or DBMS to process the request. When the mobile user is handed over to another base station, the DAA at the new station receives transaction information from the old base station. The mobile user accesses data and information by issuing transactions. We define the *Mobile Transaction* as the basic unit of computation in the mobile environment. It is identified by the collection of sites (base stations) it hops through. These sites are not known until the transaction completes its execution. A major function performed by the DAA is management of the mobile transaction. We call this component of the DAA the *Mobile Transaction Manager* (MTM). Its responsibilities include keeping track of the execution status of all mobile transactions concurrently executing (or previously executed at this site but not yet committed), logging recovery information, and performing needed checkpointing.

We view a mobile DBMS computing environment as an extension of a distributed system. Özsu and Valduriez have provided an excellent classification for distributed DBMSs based on the system characteristics of autonomy, distribution, and heterogeneity [16]. Figure 2 extends their classification to include mobile DBMS systems. To their classification we have added an extra point on the distribution axis. This is because a mobile computing system must include a fixed network (see figure 1) which is a distributed system. A mobile computing system can thus be viewed as a dynamic type of distributed system where links between nodes in the network change dynamically. These changing links represent the connection between the mobile units and the base stations to which they are connected. Based on this categorization we could conceivably have a mobile system with no autonomy or heterogeneity among the component DBMSs. However, we view this as being highly unlikely. We target our transaction model to the most general system: a Mobile Heterogeneous, Multidatabase System. By building on transaction models designed for the most general distributed environment, Distributed Heterogeneous Multidatabase System, we develop a transaction model for the most complicated mobile computing environment. Our model as will be detailed in section 4 captures such generalization by observing the constraints and requirements of heterogeneous and multidatabase systems, including local autonomy, and global serializability constraints.

3. Requirements of a mobile transaction

There have been many previously proposed transaction models and many of these are applicable for a distributed database system [8]. In fact, we will build our mobile transaction model using the concepts of Open Nested Transactions [23] and Split Transactions [17]. However, these models alone do not capture the complete behavior of a mobile transaction. In this section we justify this claim and provide motivation for our mobile transaction model.

One of the primary requirements which we would like to have for a mobile transaction is to build on top of the existing infrastructure in the fixed network. The software at the DAA will be used primarily to manage the mobile transactions. The source system software will be responsible for accessing the data. Since we target our model for a Mobile Heterogeneous Multidatabase System, we assume complete autonomy of the component DBMSs and the various multidatabase systems. Figure 3 shows our view of how mobile transaction management is related to multidatabases. The DAA serves as a mobile transaction manager built on top of existing Global Database System (GDBS) and DBMS software residing on the fixed network. Its relationship to the GDBS is similar to the GDBS relationship to local DBMS

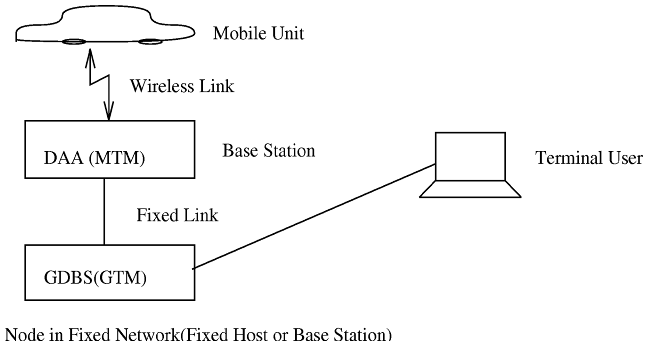


Figure 3.

systems. A GDBS assumes that the local DBMS systems perform the required transaction processing functions including recovery and concurrency. The DAA's view of the GDBS is similar to that seen by a user at a terminal in the fixed network. As a matter of fact, as seen in figure 3 there may be terminals in the fixed network issuing the same transactions from the same GDBS. The GDBS is not aware of the mobile nature of some nodes in the network and the DAA is not aware of the implementation details of each requested transaction.

One aspect to mobile transaction management which differs greatly from that in distributed transaction management is the movement of the transaction through the network. In order to properly manage a mobile transaction, this movement must be documented by the transaction controlling mechanism. This movement is as much a part of the transaction behavior as is the data access. When a mobile transaction hops to a new cell, the control of the transaction may move or may remain at the originating site. If it remains at the originating site, however, messages would have to be sent from the originating site to the current base station any time the mobile unit requests information. To avoid this message overhead, we assume that the transaction management function moves with the mobile unit. Each DAA has log information for that portion of the transaction that is executed from its site.

A comprehensive example is now provided which will serve to illustrate the requirements of a mobile transaction model. This example is an extension of one found in [5]. In an automobile insurance agency, an insurance adjuster must physically examine each damaged automobile and provide a cost estimate for needed repairs. To provide this estimate she may need to access information about the automobile, any police reports concerning the car accident in which the vehicle was damaged, information about previous insurance claims from this individual, and the current value which the automobile has as found in the "Blue Book". The ultimate purpose of the database adjustment transaction is to provide a report including insuree information, accident information, damage information, cost estimate for repairs, and any recommendation from the adjuster concerning whether the insurance company should pay for the damage and whether follow up actions are needed (cancel insuree, attempt to recover from any third party involved in accident, etc.). We first examine how this Adjustment Transaction would be accomplished in the world of today. We then examine how it could be performed in the mobile world of 2005. Figure 4 illustrates this example. In the world of 1995, the adjuster receives the request to perform the adjustment transaction either in written form from her supervisor or in electronic form when logging into her PC/terminal in her office/home. Prior to examining the car, the adjuster must gather information about the insuree, automobile, and report. Some of the information is gathered via phone calls, some must be requested in person at the appropriate location, and others by requesting it from internet sites. The adjuster then drives to the location of the automobile and

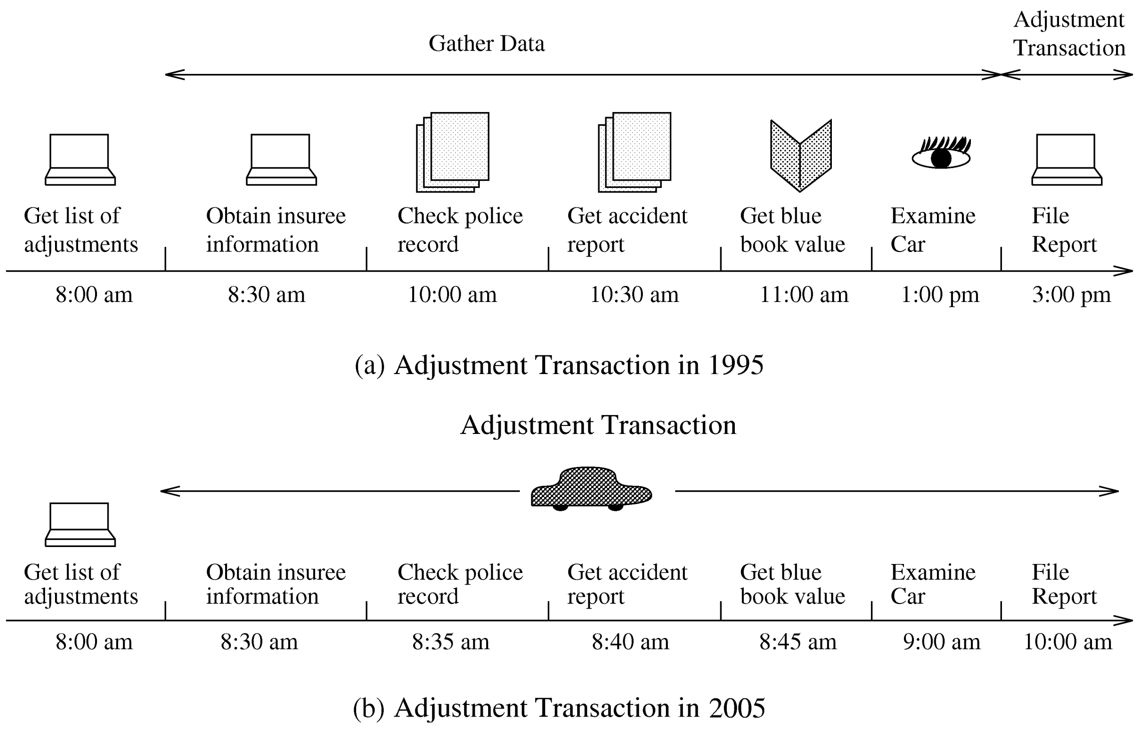


Figure 4.

physically examines it. As she does so, written notes are made concerning the damage. The adjuster then drives back to her office and executes the adjustment transaction. All of the information gathered throughout the day is input via this transaction and the appropriate report is sent to her supervisor. Much of the work was performed off line while the needed information was being gathered. Now, how do we envision that this work could be performed in the year 2005 using mobile transactions? First of all, much of the gathering of the needed data will be performed online. After the adjuster has examined the claims to be filed for that day, she determines which to evaluate first. Then she immediately begins the adjustment transaction. Logically, this transaction performs the same functions as that of the 1995 model, but the gathering of data is done by the transaction itself. Subtransactions will be generated to obtain all of the needed data automatically. By the time she arrives at the car location, all of the needed data is available. While examining the car she may retrieve any of the data desired. When she examines the car she enters all the needed information into the mobile unit which automatically updates the data in the source DBMS. Any needed reports are filed at that time. When she leaves the scene, she has finished processing the claim. She can then begin on the next adjustment transaction as she drives to the location of the next damaged automobile. The use of the mobile unit has had several impacts on the processing of the claim. First of all the claims processing has been speeded up and overall efficiency of the the adjuster has been increased. Absolutely no paperwork has been involved. Secondly, the length of time that the transaction is active has increased.

There is one additional complicating factor that may occur. Let's suppose that the first adjustment transaction has been completed at 11:00 am. The second transaction is started at that time. On the way to examine the second automobile, however, the adjuster decides to stop for lunch. To conserve power, she turns off her mobile unit. After lunch she turns the unit back on and resumes the active transaction at the point in time that she left. Thus if a voluntary shut down of the mobile unit occurs, the active transaction should not be terminated. Note that she may wish to do the same thing if, for some unforseen problem, the computer was involuntarily shut down. For example, she may lose power in the mobile unit battery. She would like to resume the active transaction when the unit is brought back on line. This complicates the management of the mobile transaction as active transactions which have been interrupted may need to be resumed. The time between interruption and resumption of the transaction could be quite long. To make matters worse, there may be cases where the user would like to see these interrupted transactions aborted.

To facilitate this ability to resume transactions, we see the need to commit portions of the mobile transaction early. These early commits can release valuable resources (locks) rather than holding them for long periods of time.

We summarize this section by listing the basic requirements which we think any mobile transaction model should satisfy.

1. Build on existing multidatabase systems and do not duplicate support provided by a source system.
2. Capture movement of mobile transaction as well as data access. Move transaction control as mobile unit moves.

3. Provide flexibility in terms of atomicity feature.
4. Support long lived transactions.

4. Kangaroo Transactions

In this section we introduce our mobile transaction model which we call Kangaroo Transactions. Our model is built on traditional transactions which are a sequence of operations executed under the control of one DBMS. Figure 5 shows the basic structure. Here three operations (op11, op12, and op13) are performed as part of the transaction. LT is used to identify the traditional transaction as it is executed as a *Local Transaction* to some DBMS. The operations performed are the normal read, write, begin transaction, abort transaction, and commit transaction. The first operation (op11) must be a begin transaction while the last (op13) must be either a commit or abort.

Our view of global transactions in multidatabase systems is somewhat broader than that which is often assumed. We actually consider two types of global transactions. The limited view of global transactions is shown in figure 6(a). Notice that the Global Transaction root (GT) is composed of subtransactions which can be viewed as Local Transaction (LT) to some existing DBMS. The local transactions are often called subtransaction (or Global SubTransaction, GST) of the GT. Each of these can in turn be viewed as consisting of a sequence of operations. Figure 6(b) takes the more general view of global transactions. In this case the subtransactions may themselves be global transactions to another multidatabase system. So we would have (for this figure) operations underneath LT1 and LT4. Underneath GT2 and GT3 would be other LTs and GTs. This view of global transactions gives a recursive definition based on the limiting bottom view of local transactions.

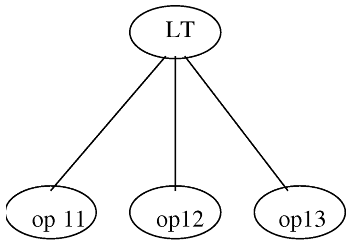
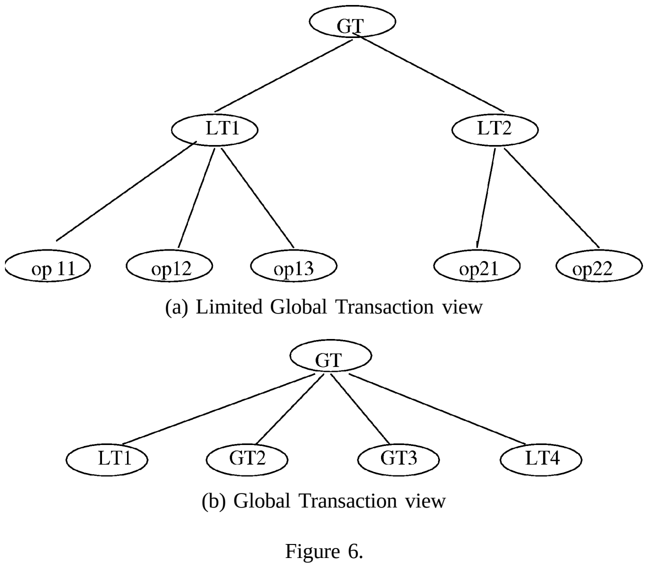


Figure 5.


GT
LT1
LT2
op 11
op12
op13
op21
op22
(a) Limited Global Transaction view
GT
LT1
GT2
GT3
LT4
(b) Global Transaction view
Figure 6.

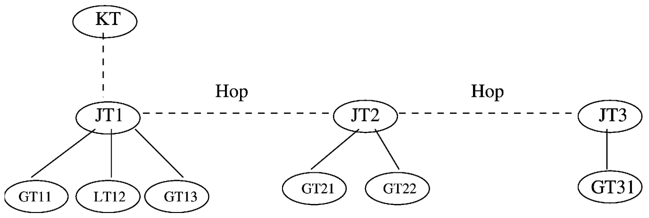


Figure 7.

4.1. Introducing Kangaroo Transactions

Global transactions serve as the basis upon which we define our mobile transactions. Global transactions alone, however, do not capture the “hopping” nature of mobile transactions. Based on the hopping property, we call our model of mobile transactions *Kangaroo Transactions* (KT).[1]

Figure 7 shows the basic structure of a Kangaroo Transaction. When a transaction request is made by a mobile unit the DAA at the associated base station creates a mobile transaction to realize this request. A *Kangaroo Transaction ID* (KTID) is created to identify the transaction. We define a KTID as follows:

$$\text{KTID} = \text{Base Station ID} + \text{Sequence Number},$$

where the base station ID is unique, the sequence number is unique at a base station, and + is a string catenation operation. Each subtransaction represents the unit of execution at one base station and is called a *Joey Transaction* (JT). We define a *Pouch* to be the sequence of global and local transactions which are executed under a given Kangaroo Transaction. The origination base station initially creates a JT for its execution. The only difference between a JT and a GT is that the JT is part of a KT and that it must be coordinated by a DAA at some base station site. When the mobile unit hops from one cell to another, the control of the KT changes to a new DAA at another base station. The DAA at the new base station site creates a new JT (as part of the handoff process). It is assumed that JTs are simply assigned identification numbers in sequence. Thus a *Joey Transaction ID* (JTID) consists of the KTID + Sequence Number. This creation of a new JT is accomplished by a split operation. The old JT is thus committed independently of the new JT. In figure 7, JT1 is committed independently from JT2 and JT3. At any time, however, the failure of a JT may cause the entire KT to be undone. This is only

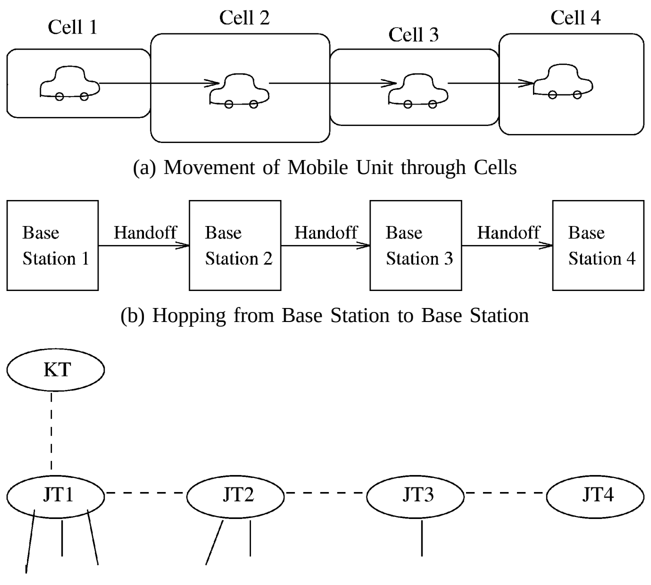
[1] The fact that the first author was on sabbatical in Australia when this idea was developed certainly contributed to this name.

accomplished by compensating any previously completed JTs as the autonomy of the local DBMSs must be assured.

To manage the KT execution and recovery, a doubly linked list is maintained between the base station sites involved in executing a Kangaroo Transaction. Control information about a JT is identified by its JTID. To complete a partially completed KT, this linked list is traversed in a forward manner starting at the originating base station site. Thus to restart an interrupted transaction, the user must be able to provide the starting site (base station) for the transaction. To undo a KT the linked list is traversed in a backward manner starting at the current JT base station site.

There are two different processing modes for Kangaroo Transactions: *Compensating Mode* and *Split Mode*. When a KT executes under the Compensating mode, the failure of any JT causes the current JT and any preceding or following JTs to be undone. Previously committed JTs will have to be compensated for. Operating in this mode requires that the user (or source system) provide information needed to create compensating transactions. This includes information that the JT is compensatable in the first place. Deciding whether a JT is compensatable or not is a difficult problem. Not only does the JT itself need to be compensatable, but the source system should also be able to guarantee the successful commitment of the compensating transaction. The split mode is the default mode. In this mode, when a JT fails no new global or local transactions are requested as part of the KT. However, the decision as to commit or abort currently executing ones is, of course, left up to the component DBMSs. Previously committed JTs will not be compensated for. Neither the Compensating nor Split modes guarantees serializability of the kangaroo transactions. Although Compensating mode ensures atomicity, isolation may be violated (thus violating the ACID principle) because locks are obtained and released at the local transaction level. With the Compensating mode, however, Joey subtransactions are serializable.

Figure 8 shows the relationship between movement of a mobile unit between cells and the corresponding Kangaroo Transaction. Here we assume that when the transaction is started, the mobile unit is in Cell 1 which is associated with Base Station 1. At this time, the DAA at this Base Station created a new Kangaroo Transaction and immediately created a Joey Transaction. When the mobile unit moves to Cell 2, a handoff is performed. As part of this handoff, the KT is split into two transactions. The first part of this transaction is the subtransactions under JT1, the remainder (at this time) will be part of JT2. Similarly, when the mobile unit moves into Cell 3 and Cell 4, handoffs occur and new Joey Transactions are created via a split operation. Note that this process is dynamic. A new Joey is created only when a hop between cells occurs: no hop – no Joey. The same transaction requested at two different times could have different structures. We illustrate this fact in figure 9. Figure 9(b) represents a short lived or slow mobility type of transaction, where as figure 9(c) shows a long lived or

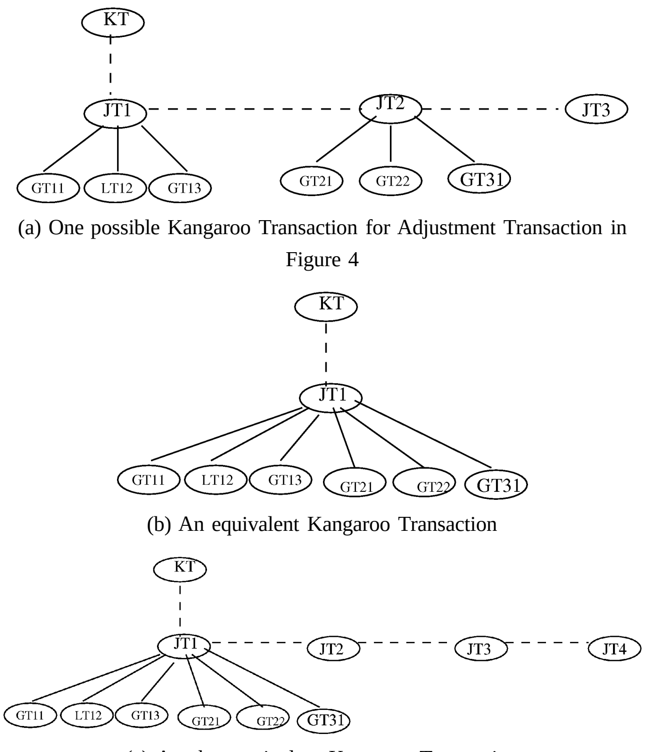


(a) Movement of Mobile Unit through Cells

(b) Hopping from Base Station to Base Station

(c) Kangaroo Transaction

Figure 8.



(a) One possible Kangaroo Transaction for Adjustment Transaction in Figure 4

(b) An equivalent Kangaroo Transaction

(c) Another equivalent Kangaroo Transaction

Figure 9.

fast mobility transaction. This figure shows three different transactions which are equivalent to that in figure 7. Even though the structure of each is different in terms of the number of Joeys and the format of the Joey, the underlying set of global and local transactions of each is the same. We thus say that two Kangaroo Transactions are *Equivalent* if they have the same pouch.

We conclude this section by describing in more detail the manner in which Joeys are created. The Joeys are created by a split operation. As part of the handoff procedure, a split operation is always performed. When looking at figure 8(b) a split is first performed during the hop from Base Station 1 to Base Station 2. The KT is then split into two subtransactions: JT1 and JT2. We assume that the subtransactions of the mobile transaction (that is the global and local transactions within it) are executed in sequence.[2] The user does not request the next subtransaction until the previous one is completed. Thus the local and global transactions under JT1 will all occur before those in JT2. This guarantees that JT1 precedes JT2 in serializable order. In fact, we assume that no subtransactions under JT2 will be created until the currently executing one in JT1 is committed. Notice that when JT2 is created there will probably be a local or global transaction under JT1 which is currently being executed. The JT2, however, must be created when the handoff occurs. The situation shown in figure 8(c) shows that JT4 has been created but no subtransactions yet exist.

4.2. A formal definition for Kangaroo Transactions

In the following, we more formally define Kangaroo Transactions. They are defined recursively with the basic building block being a local transaction defined for a DBMS. Our definitions follow that found in [2] as a starting point and building block.

Definition 1. A *Local Transaction LT* is a sequence of read (r_i) and write (w_i) operations ending in either a commit(c_i) or abort(a_i) operation.

Definition 2. A *Global Transaction GT* is any sequence of global transactions G_j and local transactions L_j.

Definition 3. A *Joey Transaction JT* is a sequence of zero or more global transactions G_k and local transactions L_k followed by either a commit c_k, abort a_k, or split s_k operation.

Note that this definition for JT ensures that the GTs and LTs within it represent a sequence. As stated earlier, we assume that all operations of one Joey are executed prior to those of the next.

Definition 4. A *Kangaroo Transaction KT* is a sequence of one or more Joey Transactions J_l. The last Joey Transaction must end in a commit c_l or abort a_l. All Joey Transactions other than the last one must end in a split s_l.

The Kangaroo Transaction captures the movement behavior of the mobile transaction by forcing all joeys but the last to end in a split.

[2] Our discussion assumes that subtransactions are requested by the user at the MU, but they may actually be requested by an agent at the base station.

Definition 5. The sequence of local and global transactions which belong to a Kangaroo Transaction is called its *Pouch*.

Definition 6. Two Kangaroo Transactions are said to be *Equivalent Kangaroo Transactions* if they have exactly the same pouch.

4.3. Mobile Transaction Manager data structures

The functions of the MTM are those related to managing a mobile transaction. The primary data structure at each site which is used to do this is the transaction status table (see table 2).

Each base station maintains a local log (see table 3) into which the MTM writes records needed for recovery purposes. Unlike DBMS and GDBS systems, this log contains no records dealing with recovering (undo or redo) data base updates. The log however is stored in stable storage in the base station. Most of these records are related to the transaction status entries. During handoff processing the log buffer must be flushed to ensure recoverability if a failure occurs during the handoff process. When (after) a Kangaroo Transaction is started a BTKT (Begin Trans. KT) record is written to the log. During handoff processing (before), an HOKT (HandOff KT) record is written into the originator's (Base Station requesting handoff) log while a CTKT (ConTinuing KT) record is written into the destination's (Base Station being handed off to) log (after). Joeys are documented in the log with a Begin (BTJT) record and a Commit (ETJT) record. BTJT records are linked together in reverse order of creation while ETJT records are linked together in forward order of creation. Subtransactions within a joey have begin (BTST) and end (ETST) records. BTST records contain the actual local/global transaction requested and the compensating transaction if the KT is compensatable.

4.4. Kangaroo Transaction processing

The flow of control of processing Kangaroo Transactions by the MTM can be described as follows:

1. When a mobile unit issues a Kangaroo Transaction, the corresponding DAA passes the transaction to its MTM to generate a unique identifier (KTID) and creates an entry in the transaction status table. The MTM also creates the first Joey Transaction to execute locally in its communication cell. At the end of this setup, a BTKT record is written into the MTM transaction log.
2. The creation of a Joey Transaction (be it the first or otherwise) is also done by the MTM and involves generating a unique JTID and creating an entry in the transaction status table. The count of number of active Joeys in the KT status table entry is incremented by 1. A BTJT record is then written into the log. Finally a JT entry is written into the transaction status table.

Table 2
KT transaction status table entries.

Record type	Attribute	Description
KT	KTID	ID for KT
	Mode	Split or Compensating
	Joey Count	Count of number of active Joeys in this KT
	Status	Active, Commiting, Aborting
	FirstJTID	Pointer to first JT status record for this KT
JT	JTID	ID for JT
	NextJTID	Pointer to next JT status record for this KT
	PriorJTID	Pointer to previous JT status record for this KT
	Status	Active, Commit, or Abort
	STList	List of local and global transactions ST
	Compensatable	Yes/No
ST	STID	ID for ST
	Status	Active, Commit, or Abort
	Request	GT or LT requested
	Compensatable	Yes/No
	CompTR	Compensating transaction

Table 3
Log records.

Record type	Contents
BTKT	KTID, Mode
CTKT	KTID, Mode
BTJT	JTID, PriorJTID
BTST	STID, Request, Compensating
ETJT	JTID, NextJTID
ETST	STID
ETKT	KTID
HOKT	KTID

3. The Kangaroo Transaction is executed in the context of Joey Transactions. For each JT, translation is made to map the KT operations into specific source system global and local transactions. Based on the translation results, an ST entry is created in the transaction table for each native (local or global) transaction. A log record, BTST, is written in the MTM log before shipping each native transaction to the appropriate source system DBMS or GDBS.

4. When network-level handoff occurs, the DAA is immediately notified so that it responds by initiating a *transaction-level handoff protocol*. When this takes place, the DAA starts executing a split operation at the origination site. As part of this split the DAA writes an HOKT record in the log to indicate that a handoff has taken place. In addition, the log buffer is flushed to the stable log on disk. These actions represent a *checkpoint* operation. Notice that subtransactions may still be executing at the source system sites.

5. The destination base station then initiates the other part of the split operation. A CTKT record is logged after a new Joey Transaction is created with the tentative content being the rest of the entire Kangaroo Transaction. In addition, a KT entry is placed in the destination's status table.

6. To ensure the atomicity of the source system DBMS and GDBS systems, they will determine when to commit or abort the subtransactions. When the DAA is notified (by the DBMS or GDMS) that a subtransaction has been committed, the DAA writes an ETST record in the log. In addition, if the KT processing mode is split, then the ST entry in the transaction status table is removed. If the KT mode is Compensating, this entry remains in case the KT is later aborted and the compensating transaction must be executed. The STList in the status table is updated to reflect the fact that this subtransaction has committed. If there are no active subtransactions for this Joey, then a ETJT record is written to the log and the status of the JT status table entry is changed to commited. Finally, the count for number of active Joeys is decremented by 1. Note that the KT status table entry for the last active Base Station contains the current values for Status and Joey count. If the Joey count is 0 and the status of the KT is Commiting, then the KT is commited.

7. When the mobile user indicates that the transaction has ended, the status entry for the KT is changed to Commiting. At this time, if the active Joey count is 0 the KT is commited.

8. To commit a KT the ETKT entry is written to the log and all status table entries for all involved Base Stations are freed.

4.5. Logging and recovery with Kangaroo Transaction processing

To illustrate recovery possibilities, in table 4 we show a sequence of actions and log records created for a simple transaction. The MU requests a transaction through base station 1 (BS1). As a result a KT (BS1.1), a JT (BS1.1.1), and ST (BS1.1.1.1) are created. In the table, the $\wedge$ indicates an empty value. The MU then moves into the cell

Table 4
Log records.

Action	Log records at BS1	Log records at BS2	Comments
Transaction requested at MU	(BTKT, BS1.1, Split) (BTJT, BS1.1.1, $\wedge$) (BTST, BS1.1.1.1, Req, $\wedge$)		Create KT, JT, ST
HandOff and Split	(HOKT, BS1.1)	(CTKT, BS1.1, Split) (BTJT, BS1.1.2, BS1.1.1$\uparrow$)	Create JT
DBMS commits subtrans	(ETST, BS1.1.1.1) (ETJT, BS1.1.2$\uparrow$)		Commit JT, ST
Another subtrans requested		(BTST, BS1.1.2.1, Req, $\wedge$)	Create ST

associated with base station 2 (BS2). As a result an HOKT is written into the log at BS1 and a CTKT is written at BS2. Immediately a new JT (BS1.1.2) is created and a corresponding log record is written. Even though the split has occured the previous JT and ST can not be committed until this message is received from the DBMS. At that time the ET records are written into the log and the next ST is created.

Suppose an MU failure occurs (or it is lost or stolen) at this point in time. Since the KT is of split mode, the currently executing JT and ST will be allowed to complete, but no new STs will be created. No real recovery occurs. This allows the transaction to be restarted at a later point if desired. If, however, the KT is of compensating mode, then a compensating transaction would have been created for each of the subtransactions (BS1.1.1.1 and BS1.1.2.1). Then in case of MU failure, the work of these subtransactions (and as a result the KT) will be "undone".

5. Relation to other research

In this section we briefly review previous research related to our view of mobile transactions. Besides reviewing these articles, we also briefly examine related work which has influenced this paper.

5.1. Related research

Kangaroo Transactions are built using the concept of Global Transactions in a *MultiDataBase System* (MDBS) [2, 3,23]. However, we add an additional layer on top of the GDMS (see figure 3). A mobile transaction is in turn composed of several subtransactions which are themselves global or local transactions. A mobile transaction, in our view, is not a global transaction (from the MDBS perspective). This is true for two reasons: the DAA performs limited transaction management functions and the DAA must be able to dynamically change the transaction structure based on the movement of the mobile unit. A major difference between mobile transactions and global transactions is the fact that mobile transactions do not have one computer site which serves as the transaction management site.

Although originally devised to handle open-ended transactions in the CAD environment, *Split Transactions* [17] lend themselves to the mobile environment very well. A split-transaction "divides an ongoing transaction into two serializable transactions" [17]. With a KT, the serialization order of the subtransactions is that in which they are split. A second major desirable property of the split transaction is that the split occurs dynamically and on the request of the split operation. This is central to the idea of a Kangaroo Transaction. We thus feel that the split transaction concept can be used to capture the "hopping" nature of a mobile transaction.

There are some similarities between our view of mobile transactions and workflow activity models [6,19,20]. A *workflow* is a set of related tasks which are executed to satisfy a business requirement. It has been argued that transaction and workflow models should be integrated into one architecture [1]. In this article, it was pointed out that multidatabase transactions indicate how data operations can be interleaved and that workflow models capture the task dependencies. With our view of mobile transactions, we argue that the data dependencies are captured by multidatabase transaction models. The control flow (in our view of mobile transactions) is captured by the movement behavior of mobile transactions. Thus our definition of mobile transactions merges control and data behavior into one transaction model.

There has recently been much research concerning migration of state information in Mobile Computing environments [4,18]. The first work [4] deals with how best to perform a request made from a MU that will require processing on multiple nodes in the fixed network. The *itinerant agent* concept assumes that a special intelligent process is "launched" at the MU and then roams through the fixed network visiting servers which will help to complete the request made by the MU. As the agent moves through the network the state of completing the request is kept. With a KT, the user may request new subtransactions based on results of earlier ones. A similar function is performed dynamically by the itinerant agent itself. Although the itinerant agent concept could be used to implement a Kangaroo Transaction, the reverse is not true. However, with an itinerant agent, there would be no need for the logging and status table information at the base stations as this state information would move with the agent. The second work [18] discusses the desire to have a *user agent* inside the fixed network functioning on behalf of the user at the mobile unit. This server acts as an intermediary between the client (MU) and server. The DAA that we propose in our model serves a similar purpose, although it is not associated with one MU. The status table information and logging information which the DAA maintains is, however, unique to the MU which requested the mobile transaction. This paper also looks at how this agent migrates on the network. Although we do not migrate the DAA, the transaction status information (including the log) does migrate. Unlike both of the agent ideas, howevear, we do not migrate the entire state information. As a matter of fact, the entire state information of the kangaroo transaction is distributed at the various nodes on the fixed network which helped to execute it.

5.2. Other mobile transaction models

Like our approach, Chrysanthis [5] views mobile transactions as being built using concepts developed for multidatabase transactions. However, unlike our view, they assume that a mobile transaction is a special type of multidatabase transaction. To manage mobile transactions, they assume that a GDBS exists at each base station to control the management of the mobile transaction. They do assume that the subtransactions of the mobile transaction will commit or abort independently and that if a subtransaction aborts, all others which have yet to be committed will also abort. Their atomic and non-compensatable transactions serve the same purpose as our two different modes of execution for a mobile transaction. However, they also have two additional types of subtransactions (reporting and co-transactions).

The clustering model [9] assumes a fully distributed system, and the transaction model is designed to maintain consistency of the database. The database is divided into clusters. In this model a mobile transaction is decomposed into a set of weak and strict transactions. The decomposition is done based on the consistency requirement. The read and write operations are also classified as weak and strict. The weak operations are allowed to access only data elements belonging to the same cluster, where as strict operations are allowed database wide access. For every data item, two copies can be maintained – one of them strict and the other weak.

The semantics-based mobile transaction processing scheme [22] views mobile transaction processing as a concurrency and cache coherency problem. The model assumes a mobile transaction to be a long lived one characterised by long network delays and unpredictable disconnections. This approach utilizes the object organization to split large and complex objects into smaller manageable fragments. A stationary database server dishes out the fragments of a object on a request from a mobile unit. On completion of the transaction the mobile hosts return the fragments to the server. These fragments are put together again by the merge operation at the server. If the fragments can be recombined in any order then the objects are termed *reorderable* objects. Aggregate items, sets, and data structures like stacks and queues are examples of fragmentable objects.

A recent paper by Yeo and Zaslavsky examined how multidatabase transactions could be submitted from mobile workstations [25]. A major premise of this article is that mobile units may voluntarily disconnect from the network prior to having any associated transactions completed. Yeo's view, like ours and like Chrysanthis', is that mobile transactions should be built on top of multidatabase global transactions. These authors also indicate that any mobile transaction model should support the concept of “long-duration transactions and sagas” [25].

Tables 5 and 6 summarize the differences between the different models. In addition, none of the other models captures the movement property of a mobile transaction as does the Kangaroo Transaction.

The issue of consistency and concurrency control has been addressed by the models in a widely varying manner. In the Reporting and Co-Transaction model compensating transactions are used to maintain the consistency of data. Kangaroo Transactions and the MDSTPM assume that the underlying database maintains consistency via concurrency control. In the Clustered Data Model the entire data model is designed around maintaining data consistency in a distributed environment.

The various models make certain assumptions about software or hardware infrastructures needed to support the model. With the Reporting and Co-Transaction model, transaction managers will have to be modified to handle reporting and co-transactions. Since concepts like delegation and co-transactions are involved, it may be difficult to implement this model as a wrapper around an existing database. The Kangaroo transaction model assumes a Data Access Agent (DAA) exists at each base station. Thus a base station will have to be enhanced to provide this facility. The Clustering model requires the ability to define and execute strict and weak transactions, and additional attributes reflecting their consistency requirements. The Semantics based Mobile Transaction Model requires additional capability at both the server and the mobile unit end to split, operate and merge objects. Moreover the model also assumes that the database operates on objects so structured that fragmentation and merging is possible. The MDSTPM model is implemented by defining an MDSTPM layer over the existing DBMSs. This layer acts as an interface between the mobile hosts and the underlying Multidatabase system. The functionality of this layer is similar to that of a multidatabase system with more requirements than that needed by the the DAA for Kangaroo transactions.

Table 5
Mobile transaction models I.

Models	Consistency Concurrency	Database system model	Additional infrastructure
Reporting and Co-Transactions	Compensating transaction	Multidatabase	Trans Manager modified
Kangaroo model	Relies on underlying database	Heterogeneous multidatabase	Data access agent
Clustering model	Bounded inter-cluster consistency	Fully distributed database	Strict and Weak transactions
Semantics based model	Based on object semantics	Distributed multidatabase	Fragmentation
MDSTPM	Relies on underlying database	Heterogeneous multidatabase systems	MDSTPM layer

Table 6
Mobile transaction models II.

Models	Net management Comm. cost	User profile	Extensions required for commercial DB	Scalability
Reporting and Co-Transaction	Handoff info required. Reporting Co-Transaction info exchanged	Can be used for relocating transactions	Transaction Manager will have to be extended to handle new transaction types	Will require high bandwidth
Kangaroo model	Handoff info required. Assumes that each base station can handle transactions	Can be used to relocate transactions		
Clustering model	Handoff info required	Used to define clusters and for transaction migration	Transaction Manager should be enhanced to handle weak, strict transactions clusters definitions	Large number of clusters or large databases could lead to cluster mgmt
Semantics based model	Handoff info required	Not required	Objects should be fragmentable or reorderable. Object managers will be required	
MDSTPM	No handoff info required. Involves only submission and querying of results	Can be used for priority queuing	Requires the transaction Manager layer above the database system	Transaction queuing could create a bottle neck

Communication costs form a significant factor in mobile transaction execution. All models, except the MDSTPM, require handoff data include information about the mobile transaction. The MDSTPM assumes that the mobile transaction management completely resides at the base station where it was originally requested. In addition, the Reporting and Co-Transaction moded requires communication between a mobile node and the fixed server takes place through co-transaction pairs or between a reporting transaction and co-transaction pair.

6. Summary and future work

In this paper we have introduced a model for mobile transactions, Kangaroo Transaction, which captures both the data and hopping nature of mobile transactions. Our model is based on both the global transaction and the split transaction models. The novel aspect of the Kangaroo transaction model is its powerful generality. It can accommodate computing in a heterogeneous, multidatabase environment. It encompasses both short lived and long lived transactions. In addition, it naturally mimics the interaction of the mobile user with the data service provided by the mobile system, which means that programming application under the Kangaroo transaction model will be easier. Future work will examine checkpointing, logging, and caching, as requirements for performance guarantees of the Kangaroo transactions. In addition, we will examine other processing modes besides Compensating and Split.

Acknowledgements

The authors would like to thank Alex Delis, Mario Nascimento, and Srihari Ramachandra for comments on earlier versions of this article. We thank Alex Delis for providing the much improved version of figure 2 which is seen in the paper. We particularly appreciate input from Maria Orlowska who pointed out the similarity to work flow problems. In addition, we wish to thank reviewers of an earlier version of this paper. Their comments helped to greatly improve the final version.

References

[1] Y. Breitbart, A. Deacon, H.-J. Schek, A. Sheth and B. Weikum, Merging application-centric and data-centric approaches to support transaction-oriented multisystem workflows, ACM SIGMOD Record 22(3) (September 1993) 23–29.

[2] Y. Breitbart, H. Garcia-Molina and A. Silberschatz, Overview of multidatabase transaction management, VLDB Journal 2 (1992) 181–239.

[3] Y. Breitbart, A. Silberschatz and G.R. Thompson, Transaction management issues in a failure-prone multidatabase system environment, VLDB Journal 1 (1992) 1–39.

[4] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris and G. Tsudik, Itinerant agents for mobile computing, IEEE Personal Communications (October 1995) 34–49.

[5] P.K. Chrysanthis, Transaction processing in mobile computing environment, in: *Proceedings IEEE Workshop on Advances in Parallel and Distributed Systems* (October 1993) pp. 77–82.

[6] U. Dayal, M. Hsu and R. Ladin, A transactional model for long-running activities, in: *Proceedings of the International Conference on Very Large Data Bases* (1991).

[7] M.H. Dunham and A. Helal, Mobile computing and databases: Anything new? ACM SIGMOD Record 24(4) (December 1995) 5–9.

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

[9] P. Evaggelia and B. Bharat, Maintaining consistency of data in mobile distributed environments, in: *Proceedings of 15th International Conference on Distributed Computing Systems* (1995).

[10] A. Helal and M. Eich, Supporting mobile transaction processing in database systems, Technical Report TR-CSE-95-003, University of Texas at Arlington (April 1995).

[11] T. Imielinski and B.R. Badrinath, Mobile wireless computing, Communications of the ACM 37(10) (October 1994) 19–28.

[12] T. Imielinski, S. Vishwanathan and B.R. Badrinath, Data on the air: Organization and access, Technical Report DCS–TR, Department of Computer Science, Rutgers University, New Brunswick, NJ 08903 (1993).

[13] J. Ioannidis and G.Q. Maguire Jr, The design and implementation of a mobile internetworking architecture, in: *Proceedings of the 1993 Winter USENIX Conference* (January 1993) pp. 491–502.

[14] D. Johnson, Ubiquitous mobile host internetworking, in: *Proceedings of the Fourth Workshop on Workstation Operating Systems*, IEEE (October 1993).

[15] B. Marsh, F. Douglis and R. Caceres, Systems issues in mobile computing, Technical Report MITL–TR–50–93, Matsushita Information Technology Lab, 2 Research Way, Third Floor, Princeton, NJ (1993).

[16] M.T. Özsu and P. Valduriez, *Principles of Distributed Database Systems* (Prentice-Hall, 1991).

[17] C. Pu, G. Kaiser and N. Hutchinson, Split-transactions for open-ended activities, in: *Proceedings of the 14th VLDB Conference* (1988).

[18] R. Ramjee, T.F. La Porta and M. Veeraragahvan, The use of network-based migrating user agents for personal communication services, IEEE Personal Communications (December 1995) 62–68.

[19] A. Sheth and M. Rusinkiewicz, On transactional workflows, IEEE Data Engineering Bulletin 16(2) (1993).

[20] H. Wächter and A. Reuter, *Database Transaction Models for Advanced Applications* (Morgan Kaufmann, 1992) Chapter 7 – The ConTract model, pp. 219–263.

[21] H. Wada, T. Yozawa, T. Ohnishi and Y. Tanaka, Mobile computing environment based on internet packet forwarding, in: *Proceedings of the 1993 Winter USENIX Conference* (January 1993) pp. 503–517.

[22] G.D. Walborn and P.K. Chrysanthis, Supporting semantics-based transaction processing in mobile database applications, in: *Proceedings of the 14th IEEE Symposium on Reliable Distributed Systems* (September 1995).

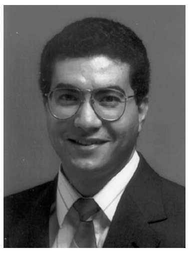
[23] G. Weikum and H.-J. Schek, *Database Transaction Models for Advanced Applications* (Morgan Kaufmann, 1992) Chapter 13 – Concepts and applications of multilevel transactions and open nested transactions, pp. 515–553.

[24] A. Whitcroft, T. Wilkinson and N. Williams, Nomads: The future of personal computing services, Technical Report TCU/SARC/1993/10, Systems Architecture Research Centre, City University, London, UK (1993).

[25] L.H. Yeo and A. Zaslavsky, Submission of transactions from mobile workstations in a cooperative multidatabase processing environment, in: *Proceedings of the 14th International Conference on Distributed Computing Systems* (1994) pp. 372–379.

Margaret (Maggie) H. Dunham received the B.A. and M.S. degrees in mathematics from Miami University, Oxford, Ohio, and the Ph.D. degree in computer science from Southern Methodist University in 1970, 1972, and 1984, respectively. From August 1984 to the present, she has been first an Assistant Professor and now Associate Professor in the Department of Computer Science and Engineering at Southern Methodist University in Dallas. In addition to her academic experience, Professor Dunham has nine years industry experience. Professor Dunham's research interests encompass main memory databases, distributed heterogeneous databases, temporal databases, and mobile computing. Dr. Dunham served as editor of the ACM SIGMOD Record from 1986 to 1988. She has served on the program and organizing committees for sev-

eral ACM and IEEE conferences. She served as guest editor for a special section of IEEE Transactions on Knowledge and Data Engineering devoted to Main Memory Databases as well as a special issue of the ACM SIGMOD Record devoted to Mobile Computing in Databases. She has published over seventy technical papers in such research areas as database concurrency control and recovery, database machines, and main memory databases.
E-mail: mhd@seas.smu.edu



Abdelsalam (Sumi) Helal received the B.Sc. and M.Sc. degrees in computer science and automatic control from Alexandria University, Alexandria, Egypt, and the M.S. and Ph.D. degrees in computer sciences from Purdue University, West Lafayette, Indiana. Before joining MCC to work on the Collaboration Management Infrastructure (CMI) project, he was an Assistant Professor at the University of Texas at Arlington, and later, a Visiting Professor of Computer Sciences at Purdue University. His research interests include large-scale systems, fault-tolerance, OLTP, mobile data management, heterogeneous processing, standards and interoperability, and performance modeling. Dr. Helal is a member of ACM and IEEE and the IEEE Computer Society, serving on the Executive Committee of the IEEE Computer Society Technical Committee on Operating Systems and Application Environments (TCOS). He is co-author of the recently published books *Replication Techniques in Distributed Systems* and *Video Database Systems: Research Issues, Application, and Products.*
E-mail: helal@mcc.com

Santosh Balakrishnan received his Bachelor's degree from the Regional College of Engineering, Nagpur, India, and Master's degree in computer science from the University of Texas at Arlington in 1992 and 1996, respectively. His current research interests include mobile agent environments, mobile systems architectures and mobile databases. He is currently with Trillium Digital Systems Inc., Los Angeles.
E-mail: santosh@trillium.com