A MOBILE TRANSACTION MODEL THAT CAPTURES BOTH THE DATA AND MOVEMENT BEHAVIOR

Margaret H. Dunham [*]
Dept of Computer Science and Eng
Southern Methodist University
Dallas, Texas 75275
mhd@seas.smu.edu

Abdelsalam Helal [†]
Dept of Computer Sciences
Purdue University
W. Lafayette, Indiana 47907
helal@cs.purdue.edu

Santosh Balakrishnan
Dept of Computer Science
University of Texas at Arlington
Arlington, Texas 76019
santosh@cse.uta.edu

Abstract

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 a 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 systems in a fixed network. 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.

---

[*] 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. Paper was written 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.

We envision that access to future information systems via mobile computers will be performed using transactions. However a transaction definition in this environment changes from that within a centralized database or even a distributed environment. 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 disconnection failures and maintain mutual consistency among replicated data. In this paper, we present a transaction model based on this hopping behavior. We build on existing models by adding the movement characteristics of mobile transactions.

We build on two previously proposed transaction models: global transactions in a multidatabase environment [?, ?, ?], and split transactions [?]. 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 and how it relates to previous models. 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 **??** shows the existing (and widely accepted) architectural model of a system that supports mobile computing [?, ?, ?, ?, ?, ?, ?, ?]. The model consists of stationary and mobile components. The only mobile component is the mobile unit. A *Mobile Unit* is a mobile computer which is capable of connecting to the fixed network via a wireless link. Stationed hosts are connected together via a fixed high-speed network (Mbps to Gbps). Components in this fixed network are classified as either fixed hosts or base stations. 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 this paper, we use the existing mobile computing model in Figure **??** 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 [?]. Our reference model consists of three layers: the source system, the data access agent, and the mobile transaction. The purpose of the reference model is to lay a foundation ground of the system architecture, the

-bb-error =

Figure 1: Architecture for Mobile Systems

system components that are important, and the assumptions made. Table **??** provides an overview of this reference model. We further examine each of these layers below.

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

Table 1: Reference Model Layers

Source System . 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 design. A distributed database system is a perfect example. Typically, the source system components reside at either base stations or fixed hosts in the fixed network. A source system (or part of it), however, can reside on a mobile unit. A specific mobile unit could, for example, contain a simple DBMS to reference or archive data at that unit. In this paper, we assume that the source system resides entirely in the fixed hosts or base stations.

Data Access Agent . 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 handle location information of the mobile user, even though it will need to use this information. Location information will be available as a separate base station service to the DAA. 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. To perform this task, the DAA must have source-system location information to determine which transaction should be executed where. 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. The target host, after it performs the transaction, returns the results to the DAA. 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. All DAAs involved in processing the transaction must be linked together. In this manner when a response is sent to a DAA, even if the mobile unit has moved out of the cell to another DAA, the response can be sent to the mobile user. To facilitate this communication, there are transaction status tables located at each DAA which are linked to transaction status tables at predecessor and successor DAAs involved in processing the same transaction.

Mobile Transaction . The mobile user accesses data and information by issuing transactions. We define the *Mobile Transaction* as the basic unit of computation on the mobile environment. Unlike a distributed transaction, a mobile transaction is not identified by a home and remote sites. 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 distributed transaction is executed concurrently on multiple processors and data sets. The mobile transaction, on the other hand, is executed sequentially through multiple base stations. The execution, depends on the movement of the mobile unit. 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.

-bb-error =

Figure 2: A Classification of Mobile Database Systems (adapted from Figure 4.4 in [**?**])

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 [?]. Figure **??** 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 **??**) 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 **??** 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 [?]. In fact, we will build our mobile transaction model using the concepts of Open Nested Transactions [?] and Split Transactions [?]. 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. Since we assume that the source system components provide the necessary software to access data efficiently in a consistent manner, we do not want to duplicate it. As much as possible we use what is provided by the source system. 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. We build the mobile transaction management on top of the existing source system MDBSs. Note that the MDBSs could in turn be built on top of other MDBSs and DBMSs. A multidatabase system assumes that there are two types of transactions: global and local. Global transactions are controlled by the *Global DataBase System (GDBS)*, while local transactions are controlled by a local *DataBase Management System (DBMS)*. A component of the GDBS is responsible for this global control: *Global Transaction Manager (GTM)*. Figure **??** 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 GDBS and DBMS software residing on the fixed network. Its relationship to the GDBS is similar to the

GDBS relationship to local DBMS systems. A GDBS assumes that the local DBMS systems perform the required transaction processing functions including recovery and concurrency. We assume that the GDMS performs similarly. That is, the DAA requires that each GDMS perform the requested transactions and that they are either committed or aborted. The fact that the GDBS may in turn request many local transactions from either component DBMS or GDBS system, is transparent to the DAA. 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 **??** there may be terminals in the fixed network issuing the same transactions from the same GDBS. The GDBS is aware that the two transactions come from two different locations in the fixed network (one from the terminal the other from a base station). However, the GDBS is not aware that one originated from a mobile unit. It treats the two transactions in the same manner. Thus 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. The DAA must know which GDBS is to execute each transaction, but it need not know any details about how the execution occurs. This implies that the DAA must have minimal addressing information to determine, based on transaction type (or content), which GDBS will execute it. In addition, the DAA must maintain a simplified log that includes information about transactions, status information about subtransactions executed at component GDBS systems, and movement information about the mobile transaction. The log will not, however, contain specifics about data behavior. It is assumed that the individual GDBS and DBMS systems maintain appropriate logs to perform this. Likewise, the DAA will perform no concurrency control functions. This is performed by the GDBS and DBMS systems. In fact, the DAA may not even be aware of what relations (objects) are being accessed. It knows only enough to forward subtransactions to the appropriate GDBS systems.

-bb-error =

Figure 3: Relationship of Mobile Transaction Management to Multidatabase Systems

One aspect to mobile transaction management which differs greatly from that in distributed transaction management is the movement of the transaction through the network. As the mobile unit hops from cell to cell, the associated base station changes. 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 begins execution, the DAA at the originating base station creates status information including creating a transaction ID and writing information in a local log. As the transaction is executed additional information is written to this log (including checkpoint data) and information about which subtransactions are (or have been executed) at which GDBS systems. 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. When the mobile unit hops to a new cell, the DAA at that cell immediately (after the handoff) becomes the new transaction manager. It has the ability, however, to trace backwards through prior controlling base stations to the originating base station for this transaction. Each DAA has log information for that portion of the transaction that is executed from its site.

-bb-error =

Figure 4: Mobile Transaction Example

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 [**?**]. 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 **??** 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 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. As they are executed, the data needed for the final report is collected. At any time the adjuster may decide to view the data at her mobile computer screen. Note that the processing is occurring as she is enroute to the location of the damaged automobile. By the time she arrives at the car location, all of the needed data is there. 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. It has been active over a period of several hours and has become an interactive transaction. Notice that in Figure **??**b) the transaction in 2005 is divided into several portions. We view each of these as actually being performed by subtransaction (either global transactions under control of a GDBS or local transactions under control of a DBMS).

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. Unlike conventional transactions, then, the commit of transactions is not performed at the end of the transaction. Instead it follows a sequence of subcommits performed at each hop of the transaction. Each hop point represents a handoff point. During the handoff a checkpoint of all information needed to track the transaction status takes place. In addition, the transaction is split and the work performed in the immediately preceding portion between hops is committed. We thus define a hop in terms of three physical occurrences: handoff of transaction between base stations, DAA checkpoints transaction information, and DAA splits transaction and commits final split. We assume that at the end of the last split transaction, the entire transaction is committed. If the mobile unit is shut down, the transaction is undone to the last commit point and can be restarted at a later time from the time of the last commit point. If, however, the entire transaction should be aborted, then a compensating transaction is executed to "undo" the effects of the previously committed subtransactions. We thus see the need to support different execution strategies for transactions. One relating to the ability to restart the interrupted transaction at a later time and the other completely undoing all work of the aborted transaction. It is conceivable that the same transaction type requested by two different individuals would execute in two different modes. For example, one adjuster prefers the ability to resume her interrupted transactions while another wishes to restart from the beginning.

Another example that demonstrates the importance of capturing movement behavior into the processing of mobile transactions is given below. Consider a transaction of ordering a take-away Burritto from a chain of Mexican Restaurants while traveling in a car. The transaction involves contacting a central server to initiate the take-away order. The current location of the mobile customer is passed by the DAA to the central service so that the latter can determine the determine the nearest franchise. The server then passes on the *order* transaction to the franchisee currently nearest to the mobile customer. The customer while ordering might pass the franchisee currently processing the order. The current franchisee, depending on direction of the mobile customer then determines the next franchisee on his path and then passes on the transaction (this forwarding could be coordinated by another DAA covering the cell area of the current franchisee). Once the order is complete the system can decide which franchisee on the customers path should deliver the burritto. That franchisee is asked to prepare the burritto. It then informs the customer the exact route to reach the franchisee where he can collect his order. Fast food chains are becoming more ubiquitously accessible through the Internet (like Pizza Hut at http://www.pizzahut.com/, and the Pizzanet at http://www2.ecst.csuchico.edu/p̃izza/).

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. This assumes the most general case of database systems (see figure **??**). Implications of this requirement include observing autonomy and resolving semantic heterogeneity.

2. Capture movement of mobile transaction as well as data access. Move transaction control as mobile unit moves. The need and importance of this requirement is demonstrated by the aforementioned examples.

3. Provide flexibility in terms of atomicity feature. This includes the ability for at least two techniques to handle interrupted transactions: one allowing complete *undo* of a transaction, and the other allowing restart at a later time.

4. Support long lived transactions. Capturing the movement, transferring transaction control, and providing alternative commitment strategies, combined support the notion of long lived transactions.

4 Kangaroo Transactions

In this section we introduce our mobile transaction model which we call Kangaroo Transactions. To examine transactions in a mobile computing environment, we must first understand transactions in fixed network architectures (including single processor, multiple processor, and distributed database systems). To do this we classify transactions based upon network type and transaction type.

4.1 Transaction Classifications

In a single processor, multi-processor, or even distributed environment the network type is fixed. When we add mobile units, the network becomes variable or mobile. Based upon figure **??** the mobile network includes the variable components (mobile units) and fixed components (base stations and fixed hosts) as well. We can also classify based on transaction type. A local transaction is one which executes under control of a single DBMS (Note that this could be in a multi-processor or single processor system.). A global transaction is viewed as executing under control of a single MDBS and consists of many subtransactions executed as local transactions. A mixed transaction contains several local and global transactions together. This leads to the following view for transactions:

$$\left.\begin{array}{c} Fixed \\ Mobile \end{array}\right\} \left\{\begin{array}{c} Local \\ Global \\ Mixed \end{array}\right.$$

We thus have six different types of transaction views as shown in Table **??**. Traditional transactions execute in a fixed network and as local transactions under control of a single DBMS. In a multidatabase system, global transactions still execute in a fixed network. They may consist of a collection of subtransactions (viewed as local transaction to some DBMS) or may themselves contain other global transactions under control of another multidatabase system. This gives us two different views for multidatabase transactions. We call the first view is *limited* and the second view is *general.* When we examine mobile transactions, the network, as shown in figure **??**, consists of both a fixed and mobile component. This leads to three different types of mobile transactions. Note that in a mobile environment, a local transaction could exist entirely at mobile unit. Although this may not be desirable due to the possibly high failure rate of mobile units, our system reference model and transaction view does allow it. This would actually be categorized as a traditional transaction which executes entirely under control of the mobile unit. Mobile-Simplistic are those transactions initiated at a mobile unit which are executed entirely at one base station. The source system components needed to execute the transaction reside at the base station with which the mobile unit is associated at the time the transaction is requested. The Mobile-Nonhopping transactions are those that will be executed at several points in the fixed network, however the mobile unit does not move out of the cell covered by the base station during the transaction execution. Both of these types of mobile transactions do not hop. They remain under the control of the DAA at one base station. The more general mobile transaction is the Mobile-Hopping. During the lifetime of the transaction, the mobile unit passes through several cells. Thus the transaction execution is controlled by several DAA components. This is the true hopping (or Kangaroo view) for mobile transactions.

Network Type	Transaction Type	System Type
Fixed	Local	Traditional
Fixed	Global	Multidatabase-Limited
Fixed	Mixed	Multidatabase-General
Mobile	Local	Mobile-Simplistic
Mobile	Global	Mobile-Nonhopping
Mobile	Mixed	Mobile-Hopping

Table 2: Transaction Views

We view traditional transactions as a sequence of operations executed under the control of one DBMS. (Although our general transaction model could support a more complex view for transactions, such as nested, we assume this simplistic flat view as our starting point.) Figure **??** 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. In this tree the order of the operation nodes under the root implies the sequence (or perceived) sequence in which the operations are executed. Even though there may not be a total ordering of the operations, there is a partial ordering and the order shown in the tree is a total ordering which satisfies the partial ordering. The first operation (op11) must be a begin transaction while the last (op13) must be either a commit or abort.

-bb-error =

Figure 5: Traditional Transaction View

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 **??**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 **??**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. In the remaining sections of this paper we assume the latter view of global transactions.

-bb-error =

Figure 6: Global Transaction Views

4.2 Introducing Kangaroo Transactions

Global transactions serve as the basis upon which we define our mobile transactions. Each DBMS or multidatabase system is located at a fixed host or base station. When involved in mobile transactions, however, we do not allow a GT or LT to be executed at a mobile unit. This restriction is for recoverability purposes only. Since we assume that any crucial data located on a mobile unit is backed up by a fixed host or base station, this assumption is not a restriction. Global transactions alone, however, do not capture the "hopping" nature of mobile transactions. This leads to our view of mobile transactions. Based on the hopping property, we call our model of mobile transactions *Kangaroo Transactions (KT)*[2]

-bb-error =

Figure 7: Kangaroo Transaction View

Figure **??** 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 discuss the contents of this ID in a subsequent paragraph.) The execution of a KT is actually associated with a chain of subtransactions. Each subtransaction represents the unit of execution at one base station. Each subtransaction is called a *Joey Transaction (JT)*. All Joeys belonging to the same Kangaroo are said to be in the same pouch. We define a *Pouch* to be the set of global and local transactions which are executed under a given Kangaroo Transaction. The origination base station initially creates a JT for its execution. (Note that a simple nonhopping mobile transaction would only have one JT.) A JT can be viewed as a type of GT. 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

---

[2] The fact that the first author was on sabbatical in Australia when this paper was written certainly contributed to this name.

new JT. In Figure **??**, 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 accomplished by undoing the currently executing JTs and compensating any previously completed JTs.

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.

Determining an ID structure for a mobile transaction is difficult. We envision that these mobile transactions will be quite volatile. They may be partially completed and then at a later time aborted (compensated) or completed. It is possible that the same user may perform this function from another mobile unit or perhaps as a different user from a mobile unit. It is conceivable that either of these functions could be performed from a terminal in the fixed network itself. Thus we would like to avoid an ID using either a user ID or mobile unit ID. Each mobile transaction will be associated uniquely with the base station where it was created. To complete or undo this transaction, certainly the identification of this base station must be known. We thus define a KTID as follows:

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

The Sequence Number is a unique number over the base station. Here + is a string catenation operation similar to the dot notation used in the Domain Naming System [**?**].

So we assume that the DAA at each base station simply assigns sequence numbers to KTs originated at that site. If a DAA at any other base station receives a request to complete or undo a mobile transaction, the ID of the base station where the transaction originated must be supplied. This DAA will then be able to communicate the request to that base station.

We envision two different processing modes for Kangaroo Transactions: *ACID Mode* and *Split Mode*. When the ACID mode is followed a KT always satisfies the ACID principle, while with split mode it may not. When a KT executes under the ACID principle, the failure of any JT causes the current JT and any preceding or following JTs to be undone. Active JTs will be undone based upon the technique provided by the source system. 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 and is how a split transaction is normally viewed. In this mode, when a JT fails, only currently executing JTs associated with the same KT will be undone. Previously committed JTs will not be compensated for.

Figure **??** 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. All actual work in a KT must be performed from within a JT. 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 **??**. Figure **??**b) represents a short lived or slow mobility type of transaction, where as Figure **??**c) shows a long lived or fast mobility transaction. This figure shows three different transactions which are equivalent to that in Figure **??**. 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.

-bb-error =

Figure 8: Relationship Between Kangaroo Transaction and Mobile Unit Movement

We conclude this section by describing in more detail the manner in which Joeys are created. The Joeys are created by a split operation. From [**?**] we know that a split transaction "divides an ongoing transaction into two serializable transactions." As part of the handoff procedure, a split operation is always performed. When looking at Figure **??**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. The local and global transactions under JT1 will all occur before those in JT2. Thus 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 **??**c) shows that JT4 has been created but no subtransactions yet exist. We envision that this could occur frequently. A long running global transaction could be executing while the mobile unit moves between many cells. This is shown in Figure **??**c). Here JT2, JT3, and JT4 have been created because the mobile unit has moved into three new cells while GT31 is being executed. This list of Joeys is needed to tie the mobile behavior of the transaction to the transaction structure. This is a major feature of the Kangaroo Transaction model. We combine the database and mobile behavior for the transaction together. The database behavior reflects the access to the data located in the source system and the mobile behavior shows the movement of the mobile unit through the cells. Both behaviors must be captured in any true mobile transaction model. The data behavior is captured by the Pouch concept.

The mobile behavior is captured in the Joey Transaction concept and is realized through the split operations. Even if the data behavior is known a priori, the mobile behavior is always dynamic. Two different transactions with the same data behavior can have different mobile behavior based on the movement (during transaction execution) of the mobile unit. From an economical point of view, the movement behavior of the mobile transaction may prove critical to the mobile user. This is because, the movement of the user away from the data source may involve higher communication charges (e.g., long distance vs local calls). Capturing movement behavior of the user can enable dynamic redistribution or caching of the data to a close by location from the user.

4.3 A Formal Definition for Kangaroo Transactions

In the following paragraphs, 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 [**?**] 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.

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.

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.

-bb-error =

Figure 9: Equivalent Kangaroo Transactions

4.4 Implementation Issues for Kangaroo Transactions

In this section we discuss implementation and management issues of Kangaroo Transactions. Unless otherwise indicated, all of the functions and data structure will reside in the DAA located at each

Base Station. Some of the implementation techniques differ based on the transaction mode (ACID or Split). Where this is the case, we indicate so.

4.4.1 Mobile Transaction Manager

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 **??**).

Record Type	Attribute	Description
KT	KTID	ID for KT
	Mode	Split, ACID, or Unknown
	FirstJTID	pointer to first JT status record for this KT
hline heightJT	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: KT Transaction Status Table Entries

Each base station maintains a local log 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. Table **??** shows the log records. When (after) a Kangaroo Transaction is started an 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 log while a CTKT (ConTinuing KT) record is written into the destination's 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.

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

Table 4: Log Records

4.4.2 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 create an entry in the transaction status table. The MTM also creates the first Joey transaction to execute locally in its communication cell. The content of the first Joey transaction is made to be the entire Kangaroo transaction. 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. Finally a BTJT record is written into the log.

3. The Kangaroo transaction is executed in the context of Joey transactions. For the first JT, translation is made to map the KT operations into specific source system global and local transactions. Based on the translation results, an entry is created in the transaction table for each native (local or global) transaction. A log record is written in the MTM log before shipping each native transaction. Notice that a KT may move and get handed over to another DAA before completion of execution of all native transactions of a JT. In this case, the KT will be executing in the context of one or more Joey transactions.

4. When network-level handoff occurs, the DAA is immediatly 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. It coordinates an atomic commit protocol on all global and local transactions of the current Joey transaction. Based on whether the Joey transaction is committed or aborted, the transaction status table is updated (committed or aborted) and

an ETJT record is logged. In case committed, information about the rest of the transaction is passed on to the next DAA the kangaroo transaction is hopping to. This information includes the KTID, transaction mode, and JTID of the committed JT. The handoff procedure must forward this information to the destination base station in a reliable way. This only requires a fault-tolerant transport protocol like the TCP/IP connection-oriented transport protocol.

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. If a Joey transaction is uncommitable (that is it must be aborted), the split operation proceeds differently. In this case, based on the transaction mode, the active Joey transactions are undone (or simply discared in an implementation that defers updates) in case split mode is used. If ACID mode is used, in addition to undoing active Joeys, each committed Joey transaction must be compensated for by a compensating transaction. Compensating transactions are canned transactioned that can be retrieved from the transaction status table.
6. When the Kangaroo transaction comes to its end successfully, *Maggie: here we have to describe how to "globally" commits the transaction. Which is nothing more than flushing out logs and discarding status entries. This must be described for both ACID and Split modes. I did not write this item because I have a problem that I would like to discuss with you first on the phone.*

4.4.3 Concurrency Control

As discussed earlier, our current view for Kangaroo transactions is that no subtransactions are executed in parallel, thus we need not worry about intra-Kangaroo transaction concurrency control. Future work will investigate how to relax this restriction. In our model, however, we are concerned with inter-Kangaroo concurrency. This concurrency arises in an implementation that allows for shared caches at the DAA. In this case, cached data can be shared by applying in-cache data access control, like the two phase locking concurrency control protocol. If cashe is not allowed to be sharable no concurrency control is needed since the underlying GDBS and DBMS systems perform their own concurrency control. We consider sharable cache because it optimizes many applications that require scalable access of large number of transactions to the same data set.

5 Relation to Other Research

In this section we briefly review previous research related to our view of mobile transactions The authors are aware of only two previously published transaction views for the mobile computing environment [?, ?]. Besides reviewing these articles, we also briefly examine related work which has influenced this paper.

5.1 Global Transactions

We build mobile transactions using the concept of Global Transactions in a *MultiDataBase System (MDBS)* [?, ?, ?]. However, we add an additional layer on top of the GDMS (see Figure **??**). In the mobile environment, local transactions will continue to be requested at the local DBMS level as will global transactions at the GDBS level. We now, however, also assume that mobile transactions will be requested of the DAA which is logically at a level above the GDBS. Many GDMS views assume the existence of one GDBS system on top of several DBMSs. Our view, however, is that there may be many GDBS systems and that they reside at multiple levels. Thus a global transaction is built of subtransactions which are either local transactions or other global transactions. 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. While a DAA at a base station must coordinate a mobile transaction, this coordination process migrates from base station to base station as the mobile transaction hops across the network. To capture this moving nature of the mobile transaction, there should be some mechanism to dynamically split mobile transactions at the "hop" points.

5.2 Sagas

Viewing long lived transactions as *Sagas* was first proposed by Garcia-Molina and Salem [?]. A Saga is made up of independent steps each of which has a compensating transaction which logically undoes its effect on the database. Even though we may view each portion of a mobile transaction executed at one base station as a step in the overall transaction, we do not ensure that each of them has an associated compensating transaction. Based on the mode of execution (item 3 from the list of required features for mobile transactions), none of the subtransactions or only a subset (not all) of them may have compensating transactions.

5.3 Split Transactions

In our model, we use the concept of *Split Transactions* [?] to incorporate this feature. Although originally devised to handle open-ended transactions in the CAD environment, split transactions lend themselves to the mobile environment very well. Mobile Transactions may be long running and interactive in nature. A split-transaction "divides an ongoing transaction into two serializable transactions." [?] It is assumed that each of the subtransactions then executes independently. The earlier subtransaction thus created is committed and the second subtransaction continues execution. Resources used by the first subtransaction are then released. There are several important features of split transactions which make them desirable in a mobile computing environment. The first is that the serialization order of the subtransactions created is that in which they are split. When a hop occurs with a mobile transaction, the activities which were directed by the DAA at the first base station will occur before (in serializable order) those at the second base station. Thus the split operation captures this serialization order. A second major desirable property of the split transaction is that the split occurs dynamically and on the request of the split operation. The hopping nature of a mobile transaction is associated with the handoff procedure when a mobile unit moves between cells. This movement can not be predicted prior to the transaction execution. If the same (equivalent set of basic database operations) transaction is executed a second time, this mobile behavior will probably be completely different. Thus to capture the mobile behavior, the transaction structure should be changeable dynamically (as the hop occurs) and on demand (when the handoff procedure is executed). We thus feel that the split transaction concept can be used to capture the "hopping" nature of a mobile transaction.

5.4 Workflow Model

There are some similarities between our view of mobile transactions and workflow activity models [?, ?, ?]. A *workflow* is a set of related tasks which are executed to satisfy a business requirement. The individual tasks in a workflow can be thought of as traditional short running transactions obeying the ACID principle. The overall execution restrictions for these transactions are reflected as control dependencies between the tasks. An execution precedence graph can be used to capture the control flow dependencies. It has been argued that transaction and workflow models should be integrated into one architecture [?]. 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. In some applications these two levels of dependencies should be captured in one model. 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 limited to the sequential dependencies created by the hopping nature of the mobile unit. Thus, when we capture the movement behavior of mobile transactions, we are reflecting the control flow. Thus our definition of mobile transactions merges control and data behavior into one transaction model. The reflected control dependencies are very simple, however, when compared to that for workflow management.

5.5 Other Mobile Transaction Models

Chrysanthis has proposed a transaction view for mobile computing [?]. Like our approach, they view 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. In our view, the Mobile Transaction Manager at the base station does not provide all the functions of the GTM. The MTM performs no concurrency control, for example. Thus our DAA is not a full fledged GDBS. We build on top of a GDBS but do not assume that the MTM is a GDBS. 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. We assume this behavior as well. They divide the subtransactions of mobile transactions into classes based on their behavior. 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). Even though we do not reflect these transaction types in our model, we use the relevant mechanism of delegation in our split-initiated commit of the accomplished part of the split transaction (See section **??**). We feel that there are two major differences between their approach and that which we propose in this paper. First of all, their model does not capture the movement behavior of the transaction. They seem to only be concerned with the data behavior. Secondly, their management of the mobile transaction appears to remain fixed at the originating site.

A recent paper by Yeo and Zaslavsky examined how multidatabase transactions could be submitted from mobile workstations [**?**]. A major premise of this article is that mobile units may voluntarily disconnect from the network prior to having any associated transactions completed. Our model of transactions includes this view, but also allows that involuntary connections may occur with similar results. That is, we do not always assume that a mobile unit disconnect should force the abnormal termination of a mobile transaction. What will happen when a disconnect occurs depends on the mode within which the mobile transaction is executed. 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" [**?**].

?? Add Santosh's table here as well as his description of clustering approach??

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 aspects 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.

7 Acknowledgements

The authors would like to thank Alex Delis, Mario Nascimento, Santosh Balakrishna, and Srihari Ramachandra for comments on earlier versions of this article. We thank Alex Delis for providing the much improved version of Figure **??** which is seen in the paper. We particularly appreciate input from Maria Orlowska who pointed out the similarity to work flow problems.

Function	Description
CreateKT	Generate KTID Write BTKT record in log Write KT status record CreateJT
CreateJT	Generate JTID Write BTJT record in log Write JT status record
CreateST	Write BTST record in log Write BT status record
Split(Originator)	Write ETJT record in log JTID.status = Commit
Split(Destination)	Write CTKT record in log Write KT status record CreateJT
CommitKT	Write ETKT in log For the first JT entry for this KT in the status table Send a CommitKT message to the MTM associated with the prior JTID Delete all KT records in status table Flush log records
CommitST	IWrite ETST in log If KTID.mode = ACID then STID.status = Commit else delete ST status records If all ST status record for this JT are deleted then CommitJT
CommitJT	Write ETJT in log If KTID.mode = Split then delete all JT records in status table

Table 5: MTM Functions