A STUDY OF KANGAROO TRANSACTION MODEL FOR MOBILE TRANSACTION MANAGEMENT

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF INFORMATICS

OF

THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

ZÜLFÜ ÖRENÇ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF INFORMATION SYSTEMS

JUNE 2004

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Zülfü ÖRENÇ
Author

Approval of the Graduate School of Informatics

Prof.Dr. Neşe YALABIK

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science

Assoc.Prof.Dr. Onur DEMİRÖRS

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc.Prof.Dr. Ahmet COŞAR

Supervisor

Examining Committee Members

Assoc.Prof.Dr. Nazife BAYKAL _____

Assoc.Prof.Dr. Ahmet COŞAR _____

Assoc.Prof.Dr. İ.Hakkı TOROSLU _____

Assist.Prof.Dr. Erkan MUMCUOĞLU _____

Dr. Altan KOÇYİĞİT _____

ABSTRACT

A STUDY OF KANGAROO TRANSACTION MODEL FOR MOBILE TRANSACTION MANAGEMENT

Örenç, Zülfü

M.S. Department of Information Systems

Supervisor: Assoc.Prof.Dr. Ahmet COŞAR

June 2004, 75 pages

Wireless network technology has advanced to the point that it is possible to use Internet connectivity to perform job tasks while moving in a city. We simulate and experimentally evaluate Dunham's Kangaroo Transaction (KT) model, and a modified version of it. Our results show that the modified-KT model does not have much communication overhead (although more than the original KT model) and it is more resilient to failures of base stations.

Keywords: Kangaroo Transaction, Joey Transaction, Mobile Unit, Mobile Support Station.

ÖZ

HAREKETLİ İŞLEM YÖNETİMİ UYGULAMASI

Örenç, Zülfü

Yüksek Lisans, Bilişim Sistemleri Bölümü

Tez Yöneticisi: Doç. Dr. Ahmet COŞAR

Haziran 2004, 81 sayfa

Donanım, kablosuz iletişim teknolojisi ve taşınabilir işlem cihazlarındaki hızlı gelişmeler,fiziksel yerleri ve hareketli olup olmadığı önemli olmadan, taşınabilir işlem cihazları taşıyan kullanıcılara bilgiye ve hizmetlere ulaşma imkanı veren, **hareketli işlem** diye isimlendirilen yeni bir model oluşturdu. Bu çalışmada hareketli işlem ortamında,geleneksel merkezi ve dağıtık istemci sunucu modellerinden farklı olan hareketli işlem yönetimi incelenmiştir. Hareketli işlem modellerinden biri olan,hem veri hem de hareketi yakalayan kanguru işlem modeli, seyyar oto tamir servisine uygulanmıştır.

Anahtar Kelimeler : Kanguru İşlemi, Joey İşlemi, Bölge İşlemi, Hareketli Birim, Hareketi Destek İstasyonu

ACKNOWLEDGEMENTS

I express sincere appreciation to Assoc.Prof.Dr.Ahmet COŞAR for his guidance, assistance and insight throughout the research. To my wife, Hale, I offer sincere thanks for her unshakable faith in me and her willingness to endure with me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

ACID **:** Atomicity, Consistency, Isolation, and Durability.

DAA **:** Data Access Agent

DB **:** Database

DBMS **:** Database Management System

FH **:** Fixed Host

GDBS **:** Global Database System

GT **:** Global Transaction

GST **:** Global Sub Transaction

JT **:** Joey Transaction

KM **:** Kangaroo Model

KT **:** Kangaroo Transaction

LLT **:** Long-lived transactions

LT **:** Local Transaction

MDSTPM **:** Multidatabase Transaction Processing Manager

MH **:** Mobile Host

MKM **:** Modified Kangaroo Model

MKT **:** Modified Kangaroo Transaction

MSS **:** Mobile Support Station

MTM : Mobile Transaction Manager

MU : Mobile Unit

RMI : Remote Method Invocation

SQL : Structured Query Language

ST : Site Transaction

TM : Transaction Management

CHAPTER 1

INTRODUCTION

Advances in portable computing devices, computer technology and wireless communication networks have lead to the emergence of mobile computing systems that have provided users the opportunity to access information and services regardless of their physical location or movement behavior [1].

Mobile computing enables enterprises to increase employee productivity, improve customer service and reduce costs. Many organizations deploy mobile solutions, by extending their existing applications or building new ones [5].

Wireless technology offers enterprises two fundamental opportunities – to run a company more efficiently using the wireless Internet, and to exploit new business opportunities that arise by providing customers, partners, suppliers, and employees with expanded access to the Internet from a variety of wireless devices. Early adopters of wireless technology are transforming their business and seeing competitive advantages in their respective industries in the form of cost reduction and productivity.

Example applications for mobile computing systems may be found in both business and personal communities. Shipping providers use mobile computing systems to track packages as each carrier navigates a route. Individuals carrying a web enabled cellular device may obtain stock quotes, submit trade orders, and receive order confirmation all while walking in a large city. The “Wishard Memorial Hospital Ambulatory Service of Indianapolis, Indiana” implemented a mobile computing system.

The system enabled emergency medical technicians to access data from a remote server using a portable-computing device while responding to a call for service. Medical technician access patient history records and writes updates to the data. At the same time, the treatment center staff accesses the data and prepare for the patient's arrival. The system enables real-time data access and sharing of potentially life saving data [2].

All of the example applications mentioned exhibit characteristics commonly found in the mobile environment. These characteristics include high mobility, frequent disconnection, limited battery power, user pauses resulting in long duration transactions, and data access [15].

1.1 Problem Statement

In Kangaroo Transaction model each time a mobile host connects to a different base station, communication will take place with the previous base stations in order to transfer transaction information to the new base station. Then, a new JT (Joey transaction) will be created in the new base station.

Each Kangaroo Transactions use either the **"compensating mode**" or the" **split mode**". In split mode if a JT fails, no extra operation is needed on previously committed JTs. n compensating mode, however all of the previously executed (and committed) JTs must be compensated for by using compensating transactions. This requires a linked list of all these JTs to be maintained by keeping the previous **"base station id"** at each JT site. However, when/if a base station fails/ (or is disconnected) it will not be possible to find out the preceding JTs.

In order to eliminate this problem, we propose to store JT information on the database server. Thus, it will be possible to identify all involved JTs and compensate for them. However, storing extra JT information and accessing the database server will increase the transaction execution time. In the rest of this thesis, we refer to this method as **Modified Kangaroo Transaction (MKT)** model.

We use simulation of Kangaroo Transactions using the Modified Kangaroo Transaction model to experimentally determine the overhead of MKT.

1.2 Approach Taken

The purpose of this thesis is to simulate a transaction model that copes with the hopping behavior of the mobile user and compare two different models (Kangaroo model and Modified Kangaroo model). The study is performed in three phases: development of the simulation model and testing environment, and generation of experimental mobile transaction scripts and running of the experiments.

In the first phase, Kangaroo transaction model is simulated via a computer program developed using the java programming language. The simulation model has been applied to a mobile auto repair service.

In the second phase, we have created 4 different mobile transaction scripts, to simulate mobility of the transactions.

In the last phase, as a result of this simulation, the two models (Kangaroo model and Modified Kangaroo model) are compared and their advantages and disadvantages are investigated.

1.3 Thesis outline

This study is divided into six chapters, namely introduction, background, simulator for Kangaroo transaction model, design and implementation of Kangaroo transaction model, experimental results and conclusions.

After an introduction is given in chapter 1, the background of the research is described in chapter 2. In the background chapter, the basic characteristics of mobile environment, mobile computing model, transactions and mobile transaction models are explained.

In Chapter 3, the simulator developed in this work for Kangaroo model is presented. In this chapter, conceptual overview and conceptual model for mobile transaction environment are also given .

In Chapter 4, design and implementation of the simulation tool is described. After general overview of the simulation tool, its database model is presented.

In chapter 5, experimental set up is discussed and results are reported.

Finally, Chapter 6 summarizes conclusions and discusses some possible related future work.

CHAPTER 2

BACKGROUND

In this chapter, firstly the basic characteristic of mobile environment, a mobile computing model, transaction operations, Kangaroo transaction model and other mobile transactions models are presented.

2.1 The basic characteristics of a mobile environment

Three essential properties pose difficulties in the design of applications for the mobile computing environment: wireless communication, mobility, and portability [6]:

· **Wireless Communication:** Mobile computers use wireless network access for communication. Lower bandwidths, higher error rates, and more frequent spurious disconnections often characterize wireless communication.

Wireless communication has some problems in the areas of:

1. Disconnection: Wireless networks are inherently more prone to disconnection. Since computer applications that rely heavily on the network may cease to function during network failures, proper management of disconnection is of vital importance in mobile computing

2. Limited Bandwidth: Wireless networks deliver lower bandwidth than wired networks. Cutting-edge products for portable wireless communication achieve only 1 megabit per second for infrared communication, 2 Mbps for radio communication, and 9 – 14 kbps for cellular telephony. On the other hand, Ethernet provides 10 Mbps, fast

Ethernet and FDDI, 100 Mbps, and ATM (Asynchronous Transfer Mode) 155 Mbps [6]. Available bandwidth is often divided among users sharing a cell. Thus, high utilization is of vital importance.

· **Mobility:** The ability to change location while keeping network connection is very important for mobile computing. As mobile computers move, they encounter heterogeneous networks with different features. A mobile computer may need to switch interfaces and protocols. Security considerations exist because a wireless connection can be easily compromised. Appropriate security measures must be taken to prevent unauthorized disclosure of information. The amount of data stored locally should be minimal; backup copies must be propagated to stationary servers as soon as possible as is done in replicated systems.

· **Portability:** Mobile computers generally small, light, durable, operational under wide environmental conditions, and require minimal power usage for long battery life. Concessions have to be made in each of the areas to enhance functionality [6]. Some of the design pressures that result from portability constraints include:

1. Low Power: Batteries are the largest single source of weight in portable computers. Reducing battery weight is important, however too small a battery can undermine the value of portability. Applications should be designed to require less communication and computation. Preference should be given to listening rather than transmitting since receptions consumes a fraction of the power it takes to transmit.

2. Limited Storage capacity: Physical size and power requirements effectively limit storage space on portable computers. Disk drives, which are an asset in stationary computers, are a liability in mobile computers because they consume more power than memory chips. This restricts the amount of data that can be stored on mobile devices. Solutions include compressing files systems, accessing remote storage over the network, shared code libraries, and compressing virtual memory [7].

2.2 A Mobile Computing Model

A widely accepted model of mobile computing is shown in Figure 1.

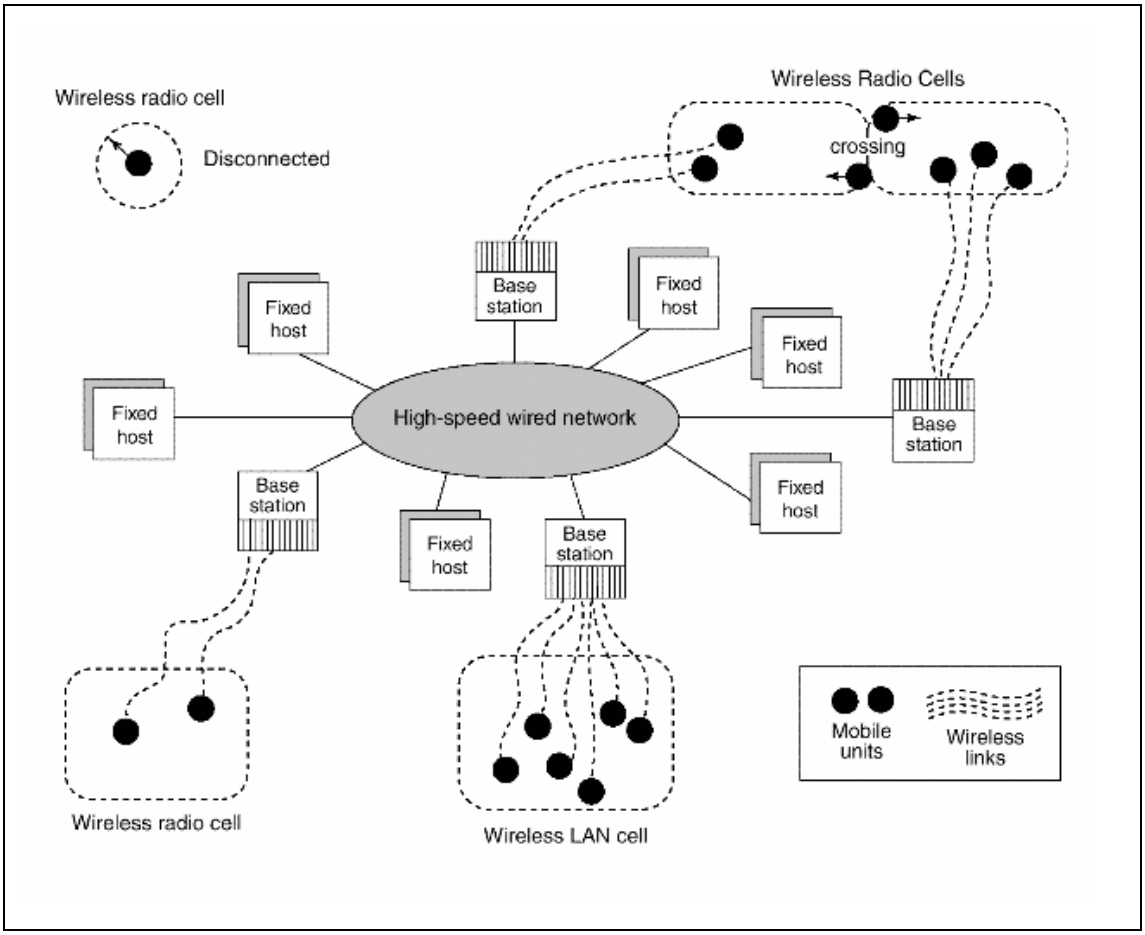


Figure 1: Mobile-Computing Model (from [1])

The model consists of stationary and mobile components. A Mobile Host (MH) is a portable computing device capable of connecting to the fixed network via a wireless link to a base station. A Fixed Host (FH) is a computer connected to a fixed network and not capable of connecting to wireless devices. A Database Management System resides on the FH to provide data storage and management facilities. A Mobile Support Station (MSS), also called "base station", is in the middle layer and capable of connecting to

both wired and wireless devices. Each MSS communicates with MHs located in its coverage area called a cell. A cell could be either a cellular connection, satellite connection, or a wireless local area network. A MH can communicate with a MSS if it is located within the cell governed by the MSS. MHs can move within a cell or between cells, effectively disconnecting from one MSS and connecting to another. At any point in time a MH can be connected to only one MSS. MHs are portable computers that vary in size, processing power, memory, etc. The size of a cell is dependent upon the cellular technology available. A MH may move within a cell or from cell to cell while retaining networking connectivity [3].

The MSS is an interface between the MH and the database residing on the FH. It also serves as an application server for MH to download software and access messages. Each MH is assigned a unique identifier and "home" MSS .The home MSS stores information such as user profiles, login files, and access rights.

The MH may act as both a data client and data server. As a data client, it submits requests for data and stores the results locally. The MH transmits location information to the MSS when connected. If the MH supports data access in disconnected mode, then it acts as a data server and must support basic transaction operations such as local read, write, commit, and abort [14].

2.3 Transactions

A common element of all mobile computing systems is the use of a transaction. A transaction is defined as a collection of database operations that form a single, logical unit of work. A traditional relational database management system is responsible to maintain four key properties for all transactions. The properties are Atomicity, Consistency, Isolation, and Durability. The meaning of each is provided below.

- **Atomicity:** All operations occur or none at all.
- **Consistency:** Relations and constraints among data elements must preserve consistency of the data.

- **Isolation:** Each transaction must appear to execute as if no other transaction is executing at the same time.
- **Durability:** The effect of the transaction must be permanent and persist, even if there are system failures [13].

Collectively the four properties referred to as the ACID properties by members of the database community. The acronym is derived from the first letter of each property.

The ANSI standard structured query language (SQL) contains statements to both define and control the logic of a transaction. The start of a transaction is an implicit BEGIN. In normal operation, a transaction ends when a COMMIT or ROLLBACK is issued. The COMMIT statement ends the current transaction by making all pending data changes permanent and automatically begins a new one. The ROLLBACK statement ends the current transaction by discarding all pending data changes. If a transaction terminates due to system crash, the state of the database depends upon the DBMS implementation. The DBMS may issue an implicit ROLLBACK to discard all pending changes and restart the database [8].

2.3.1 Transaction operations

- **Begin transaction**. This marks the beginning of transaction execution.
- **Read or Write**. These specify read or write operations on the database items that are executed as part of a transaction.
- **Commit transaction:** This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.
- **Rollback (or Abort):** This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

- **End transaction**. This specifies that read and write transaction operations have ended and marks the end of transaction execution. However, at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it violates serializability [13].

2.3.2 Mobile Transactions Characteristics

The access to the future information systems through mobile computers will be performed with the help of mobile transactions.

However, a transaction in this environment is different from the transactions in the centralized or distributed databases in the following ways.

- The mobile transactions might have to split their computations into sets of operations, some of which execute on mobile host while others on stationary host. A mobile transaction share their states and partial results with other transactions due to disconnection and mobility.
- The mobile transactions require computations and communications to be supported by stationary hosts.
- As the mobile hosts move from one cell to another, the states of transaction, states of accessed data objects, and the location information also move.
- The mobile transactions are long-lived transactions due to the mobility of both the data and users, and due to the frequent disconnections.
- The mobile transactions should support and handle concurrency, recovery, disconnection and mutual consistency of the replicated data objects [4].

2.3.3 System Log

To be able to recover from failures that affect transactions, the system maintains a log to keep track of all transaction operations that affect the values of database items. This information may be needed to permit recovery from failures. The log is kept on disk, so any type of failure except for disk or catastrophic failure does not affect it. In addition, the log is periodically backed up to archival storage (tape) to guard against such catastrophic failures.

2.4 Overview of Mobile Transaction Models

- **Kangaroo Transaction Model**

The architecture of the Kangaroo Transaction Model consists of three tiers. An important addition to the common mobile computing architecture is the inclusion of a Data Access Agent (DAA) in the middle tier. It is assumed that each MSS is capable of hosting a DAA. A major function performed by the services in the middle tier is Mobile Transaction Management (MTM). MTM involves tracking the execution status of all transactions, logging recovery information, forwarding mobile host (MH) transaction requests to the underlying DBMS, and participating in network handoff as the MH hops from station to another [1].

- **Reporting and Co-Transactions**

Chrysanthis[12] has proposed a transaction view for mobile computing .Like Kangaroo model, they view MTs as being built using concepts developed for multidatabase transactions. To manage mobile transactions, they assume that a GDBS exists at each base station to control the execution 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 are yet to be committed will also abort. However, they also have two additional types of subtransactions (reporting and co-transactions).

- **Clustering model**

With the clustering model [16, 17], the database is divided into clusters. A cluster defines a set of mutually consistent data. Inconsistencies are allowed to exist between clusters. In a mobile computing environment, data at an MU is in a different cluster than data in the fixed network. Transactions that execute at the MU do not ensure consistency between these two clusters. For each transaction executed at an MU, a proxy, which has only the associated update operations, will be executed later in the fixed network.

- **Semantics-based model**

The semantics-based mobile transaction-processing scheme [18] views mobile transaction processing as a concurrency and cache coherency problem. The model assumes a mobile transaction to be a long lived one characterized 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.

- **Multidatabase Transaction Processing Manager (MDSTPM)**

Yeo and Zaslavsky examined how multidatabase transactions could be submitted from mobile workstations [11]. 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 Kangaroo 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" [11].

A summary of the features and infrastructure associated with a selection of proposed models is shown in Table 1.

Of all previously proposed mobile transaction models, the Kangaroo model is the only one that captures the movement nature of the MU. So Kangaroo model is selected for study.

Table 1: Mobile Transaction Models

Models	Database system model	Additional infrastructure	Execution in
Reporting and Co-Transactions	Multidatabase	Transtion Manager modified	MU or Fixed Network
Kangaroo model	Heterogeneous multidatabase	Data access agent	Fixed Network
Clustering model	Fully distributed database	Strict and Weak transactions	MU or Fixed Network
Semantics based model	Distributed multidatabase	Fragmentation based model object	Restricted Server/MU
MDSTPM	Heterogeneous multidatabase systems	MDSTPM layer	MU or Fixed Network

2.5 Sagas

Transactions in the mobile computing environment exhibit characteristics that require special control mechanisms. Mobile environment transactions are long-lived in nature, subject to frequent disconnection, and may start and end at different cells. Transaction processing in the mobile computing environment is different from fixed host transaction or distributed transaction processing [1]. During the late 1980's, Hector

Garcia-Molina and Kenneth Salem authored a paper titled Sagas. The paper describes the results of their study on long-lived transactions (LLT) and suggested transaction processing control mechanism.

A long-lived transaction is a transaction that takes a substantial amount of time to execute. Execution may take several hours or days to complete. Since long-lived transactions are transactions, the database management system must apply the ACID properties. Most database management systems will lock resources requested by a transaction until it completes. Locking resources for extended periods will cause a conflict with other transactions. This results in a deadlock and a high abort rate [10].

The authors introduced the concept of a saga and a control mechanism to resolve the problems introduced by LLT. A saga is a collection of sub-transactions that may be intermixed with other transactions. The sub-transactions in a saga are related and collectively must satisfy the atomicity property. Each sub-transaction must be executed. Any partial execution of the saga is unfavorable and must be compensated for. The concept of a saga relaxes the atomic property so resources may be released as each sub-transaction is executed. The saga as a whole must satisfy the atomic property[10].

For illustration purposes, consider a saga consisting of transactions *T1*, *T2*, *T3*, and *T4*.Associated with each transaction is a compensating transaction *C1*, *C2*,*C3*, and *C4***.**The purpose of each compensating transaction is to undo any actions performed by the transaction. Executing an action and its compensating transaction returns the database to the "same" state as before. However, the meaning of "same" database state does not guarantee the database will be restored to a state identical to that which existed before. Each compensating transaction may not return the database to the identical state that existed prior to the execution of the transaction. It is possible that some other transaction was executed between the time the action was executed and its compensating transaction. No effort is made to notify the other transactions at the time a compensating transaction is executed. The net effect on the database state is as if neither the actions nor the compensating transaction was executed. The execution of a saga consisting of transactions *T1*…*Tn* and compensating transactions C1…*Cn-1* is guaranteed to result in

two possible outcomes. The first outcome is optimistic in the sense all transactions execute in the order *T1…Tn*. The second outcome addresses the partial execution scenario in which the order of execution is as follows: T1, T2,….Tj , Cj,…..C2,C1 for some $0 \leq j \leq n$ [10]. It may appear from the description that a saga is a nested transaction. However, a saga is a special type of nested transaction. Sagas limit the number of levels to two - the top level and simple sub-transactions. This limit enables the DBMS to utilize traditional concurrency control mechanisms such as locking on the sub-transactions and compensating transactions for the overall transaction [10].

CHAPTER 3

SIMULATOR FOR KANGAROO TRANSACTION MODEL

In this chapter, conceptual overview and conceptual model for mobile transaction environment are discussed.

3.1 Conceptual Overview

3.1.1 Mobile Auto Repair Service

Mobile auto repair service is given here as an example application. Figure 2 shows the advantage of mobile transactions in comparison to classical transactions. In this example when a car breakdown is reported, auto repair service must physically repair the automobile or have it towed to a service center. In a classical system after gathering needed information, repair service personnel go to location of the automobile. After repair, repair service will return to the service center for the next job to be assigned. However, in mobile environment, repair service can receive the needed information while it is moving towards the breakdown location, so it convenes time by obtaining information while it is moving. Also, it can be assigned to the next job remotely, without requiring it to go back to the service center. Of course, it will receive required information (such as location, payment method, insurance, etc) about the next job using again mobile transactions.

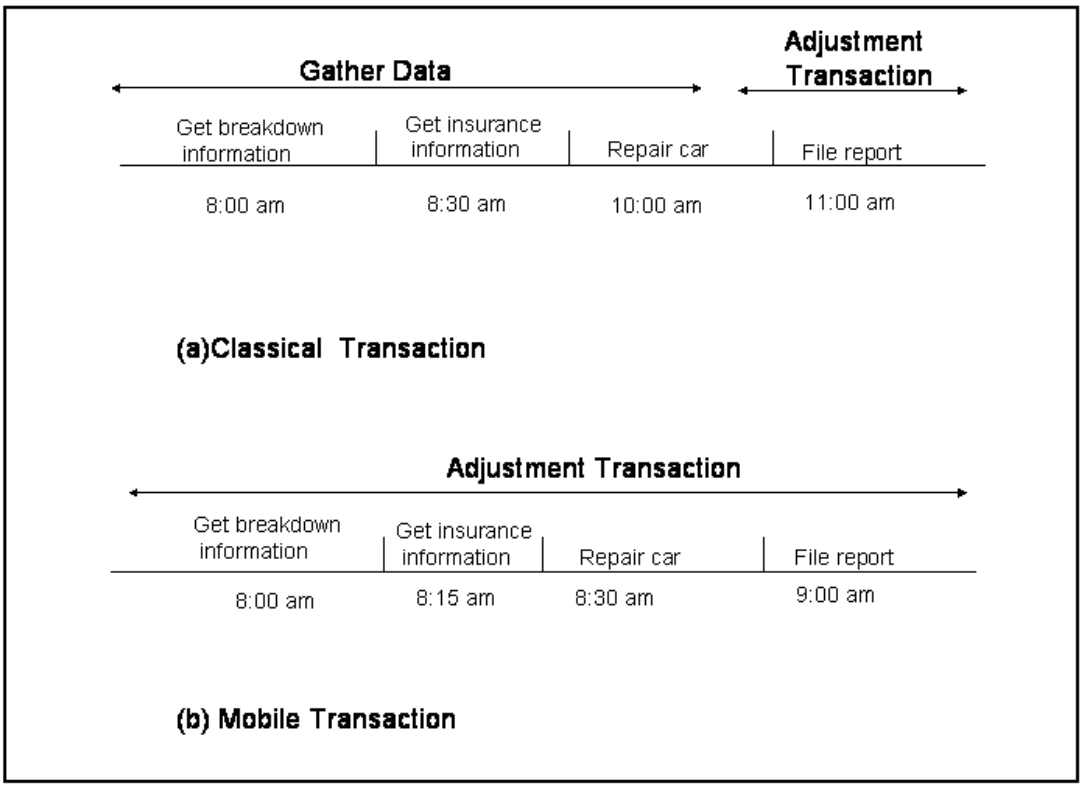


Figure 2: Mobile Auto Repair Service

3.1.2 Kangaroo Transaction

A mobile transaction model has been defined addressing the movement behavior of transactions [1]. Mobile transactions are named as Kangaroo Transactions, which incorporate the property that the transactions in a mobile environment hop from one base station to another as the mobile unit moves. The model captures this movement behavior and the data behavior reflecting the access to data located in databases throughout the static network.

The reference model assumed in [1] has a Data Access Agent (DAA), which is used for accessing data in the database (of fixed host, base station or mobile unit) and each base station hosts a DAA. When it receives a transaction request from a mobile user, the DAA forwards it to the specific base stations or fixed hosts that contain the

required data. DAA acts as a Mobile Transaction Manager and data access coordinator for the site. It is built on top of an existing Global Database System (GDBS). A GDBS assumes that the local DBMS systems perform the required transaction processing functions including recovery and concurrency. A DDA's view of the GDBS is similar to that seen by a user at a fixed terminal and GDBS is not aware of the mobile nature of some nodes in the network. DDA is also not aware of the implementation details of each requested transaction.

When a mobile transaction moves to a new cell, the control of the transaction may move or may retain at the originating site. If it remains at the originating site, messages would have to be sent from the originating site to the current base station any time the mobile unit requests information. If the transaction management function moves with the mobile unit, the overhead of these messages can be avoided. For the logging side of this movement, each DAA will have the log information for its corresponding portion of the executed transaction.

The model is based on traditional transaction concept which is a sequence of operations including, read, write, begin transaction, end transaction, commit and abort transaction operations. The basic structure is mainly a Local transaction (LT) to a particular DBMS.

On the other hand, Global Transactions (GT) can consist of either sub-transactions viewed as LTs to some DBMS (Global Sub-transaction -GST) or sub-transactions viewed as sequence of operations which can be global themselves (GTs). This kind of nested viewing gives a recursive definition based on the limiting bottom view of local transactions. A hopping property is added to model the mobility of the transactions and Figure 3 shows this basic Kangaroo Transaction (KT) structure and Figure 4 shows the relationship between movement of a mobile unit between cells and the corresponding Kangaroo Transaction.

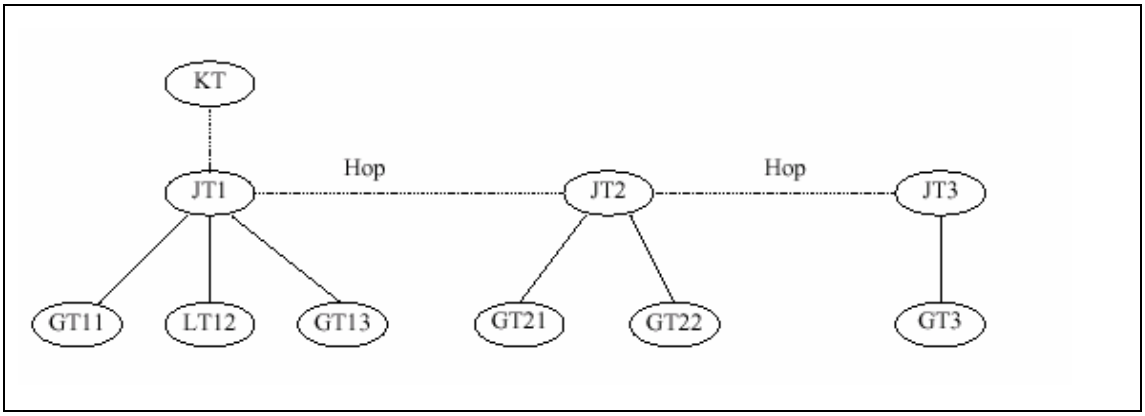
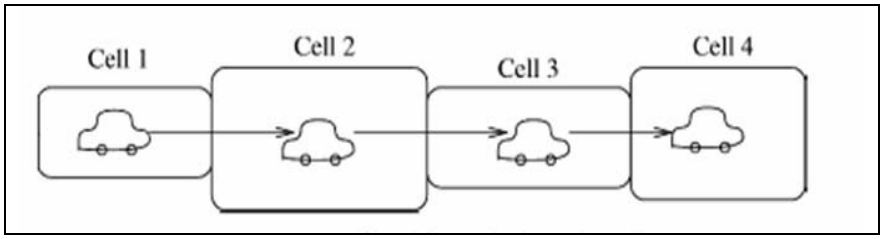
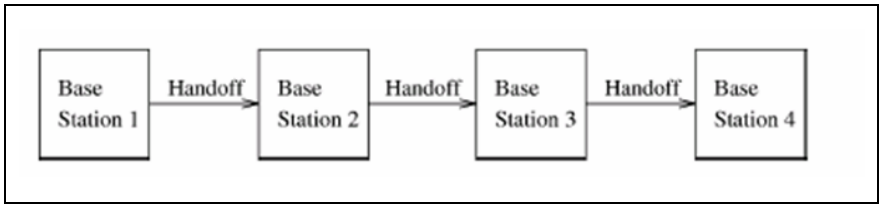


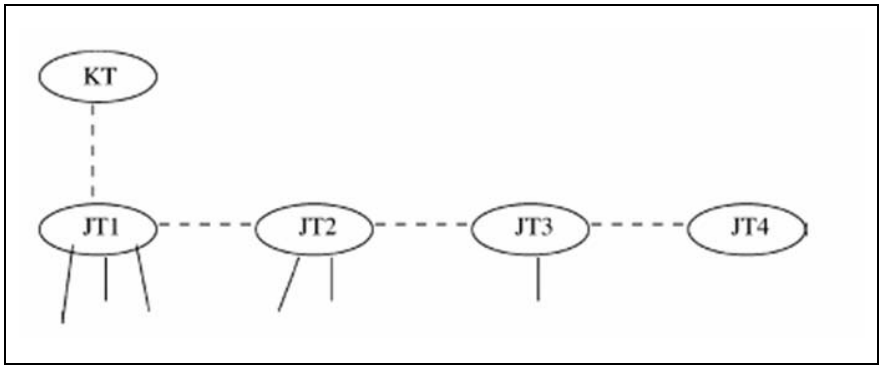
Figure 3: Basic Structure of Kangaroo Transaction (from [1])



(a) Movement of Mobile Unit through Cells



(b) Hopping from Base Station to Base Station



(c) Kangaroo Transaction

Figure 4: Kangaroo Execution (from [1])

Each sub-transaction represents the unit of execution at one base station and is called a Joey Transaction (JT). The sequence of global and local transactions which are executed under a given KT is defined as a Pouch. The origin of base station initially creates a JT for its execution. A GT and a JT are different from each other only JT is a part of KT and it must be coordinated by a DAA at some base station site. A KT has a unique identification number consisting of the base station number and unique sequence number within the base station. When the mobile unit moves 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 the result of the handoff process. JTs have also identifications numbers in sequence where a JT ID has both the KT ID and the sequence number.

The mobility of the transaction model is captured by the use of split transactions. The old JT is thus committed independently of the new JT. In Figure 4, JT1 is committed independently from JT2 and JT3. If a failure of any JT occurs, that may result the entire KT to be undone by compensating any previously completed JTs since the autonomy of the local DBMSs must be assured. Therefore, a Kangaroo Transaction could be in a Split Mode or in a Compensating Mode. A split transaction divides an ongoing transaction into serializable sub-transactions. Earlier created sub-transaction may be committed and the second one can continue to its execution. However, the

decision as to abort or commit currently executing ones is left up to the component DBMSs. Previously JTs may not be compensated so that neither Splitting Mode nor Compensating Mode guarantees serializability of kangaroo transactions. Although Compensating Mode assures atomicity, isolation may be violated because locks are obtained and released at the local transaction level. With the Compensating Mode, Joey sub-transactions are serializable. The Mobile transaction Manager (MTM) keeps a Transaction Status Table on the base station DAA to maintain the status of those transactions. It also keeps a local log into which the MTM writes the records needed for recovery purposes, but the log does not contain any records related to recovering database operations. Most records in the log are related to KT transaction status and some compensating information. Kangaroo Transaction model captures both the data and moving behavior of mobile transactions and it is defined as a general model where it can provide mobile transaction processing in a heterogeneous, multidatabase environment. The model can deal with both short-lived and long-lived transactions. The mobile agents concept [9] for multi-node processing of a KT can be used when the user requests new sub-transactions based on the results of earlier ones. This idea is discussed in [1] as pointing out that there will be no need to keep status table and log files in the base stations DAA. In this case, agent infrastructure must provide the movement of the state information with the moving agent.

3.1.3 Kangaroo Execution

The flow of a Kangaroo Transaction is outlined in Tables 2 and 3. The processing mode under consideration is Compensating mode and involves two MSSs. The process begins when the MU requests a transaction from MSS1 and ends when the MU indicates the end of the transaction. The status values assigned are Active (1), Committing (2), or Aborting (3). The Log records shown in table 3 either Begin (B) or End (E) followed by the transaction type. Handoff from one MSS to another is recorded in the log as Hand Off KT and Continue KT record types. Compensating transaction data is stored in the status table for all site transaction records. Each transaction identifier is a string formed by concatenating the MSS name and a sequence number. The KTID assigned in the example is MSS1.1. An example JTID is MSS1.1.1.

Table 2: Kangaroo Transaction Example Status Table Records

Sequence	Action	Status Table MSS1			Status Table MSS2		
		Type	ID	Status	Type	ID	Status
1	MU requests transaction at MSS1.	KT	MSS1.1	1			
2	MSS1 creates a JT to execute locally.	JT	MSS1.1.1	1			
3	A site transaction (ST) is executed.	ST	MSS1.1.1.1	1			
4	MSS1 COMMIT site transaction.	ST	MSS1.1.1.1	2			
5	A second ST executes locally as part of the JT.	ST	MSS1.1.1.2	1			
6	MSS1 COMMIT site transaction	ST	MSS1.1.1.2	2			
7	Handoff between MSS1 and MSS2 occurs.	JT	MSS1.1.1	2	JT	MSS1.1.2	1
8	A site transaction is executed.				ST	MSS1.1.2.1	1
9	Site transaction committed.				ST	MSS1.1.2.1	2
10	MU indicates intent to end transaction				JT	MSS1.1.2	2
11	MSS1 updates KT status to COMMIT.	KT	MSS1.1	2			

KT : Kangaroo Transaction **JT** : Joey Transaction

ST : Site Transaction **Status** : 1(Active), 2(Commit)

Table 3: Kangaroo Transaction Example Log Records

Sequence	Log Record MSS1		Log Record MSS2	
	Type	Contents	Type	Contents
1	BKT	KTID (MSS1.1)		
2	BJT	JTID (MSS1.1.1)		
3	BST	STID (MSS1.1.1.1)		
4	EST	STID (MSS1.1.1.1)		
5	BST	STID (MSS1.1.1.2)		
6	EST	STID (MSS1.1.1.2)		
7	EJT	JTID (MSS1.1.1)		
7	HOKT	KTID (MSS1.1)	CKT	KTID (MSS1.1)
7			BJT	JTID (MSS1.1.2)
8			BST	STID (MSS1.1.2.1)
9			EST	STID (MSS1.1.2.1)
10			EJT	JTID (MSS1.1.2)
11	EKT	KTID (MSS1.1)		

BKT : Begin kangaroo transaction
BJT : Begin joey transaction
BST : Begin site transaction
HOKT : Hand-off kangaroo transaction
EKT : End kangaroo transaction
EJT : End joey transaction
EST : End site transaction
CKT : Continue kangaroo transaction

3.2 Conceptual Model for Mobile Transaction Environment

3.2.1 Entities and Objects

Mobile Unit (MU)

- Mobile computer capable of connecting to a fixed host via wireless link.

- Client side user interface to submit queries and display results.

Fixed Host (FH)

- Commercial DBMS resides on **FH** along with other services offered to clients.
- Communicates with clients using wired network.

Mobile Support Station (MSS)

- Interface between **MU** and **FH**. It can establish wireless link(s) to **MU**s and has permanent wired connectivity to **FH**.
- Host Data Access Agent (**DAA**) responsible for transaction management.
- Location management services provide previous MSS information when a new MSS is connected.

Database (DB)

- A **database** is a collection of related data of mobile auto repair service.

3.2.2 System Capability Requirements

To develop transaction script language to simulate mobility and realize the mobile transactions. High-level requirements shall be:

- When a transaction is initially created, a "**kangaroo transaction**" will be created, along with a local "**joey transaction**".
- System shall allow mobile users to login and connect from within any cell.
- System shall execute a transaction script, after authenticating the mobile user.
- System shall send the result of each script line to mobile user.

- System shall allow the mobile user unrestricted movement among Mobile Support Station (MSS).
- After any handoff, system shall allow the new MSS to create a new local **"joey transaction"** and continue execution of the remaining part of the **"kangaroo transaction"**.
- When a mobile user is disconnected, system shall start a timer and holdup execution of the **"kangaroo transaction"**.
- System shall allow a mobile user to reconnect and resume transaction processing at any MSS before a timeout occurs (currently set to five minutes).
- System shall abort all **"joey transactions"** and the **"kangaroo transaction"** execution, if mobile user fails to reconnect within the timeout interval.
- System shall execute each "**joey transaction**" independently.
- After any handoff, the local "**joey transaction**" shall commit independently.
- When the last "**joey transaction**" of a "**kangaroo transaction**" is committed, the "**kangaroo transaction**" is removed from the system.

The planned system is represented in appendices (J-K) as data flow diagrams in Level-0 and level-1 levels.

Use case diagram in Figure 5 is used to capture the functional requirements of the system.

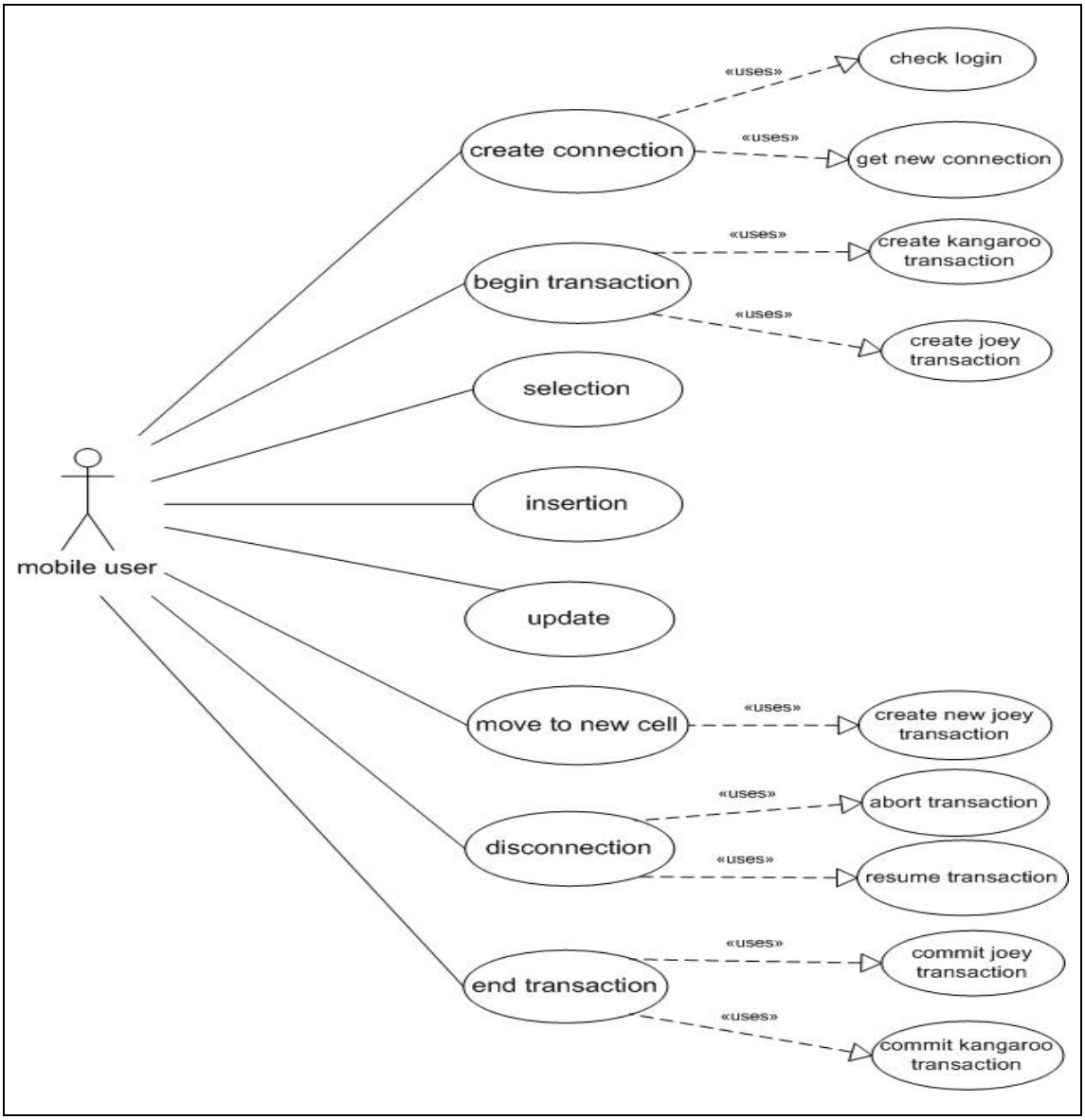


Figure 5: Use Case Diagram of System

Use Case: Create Connection

Actor actions:	System response:
The mobile user will connect to MSS.	The system creates connection after authentication.

Use Case: Begin Transaction

The mobile user starts transaction execution.	The system creates kangaroo transaction and joey transaction.

Use Case: Selection, Insertion and Update

The mobile user realizes transaction operation (read/write).	The system returns the result of each transaction operation.

Use Case: Move New Cell

The mobile user comes into new cell.	The system continues the remaining transaction execution.

Use Case: Disconnection

The mobile user disconnect to MSS because of failure.	If mobile user reconnects in limited time, the system resumes transaction, otherwise abort transaction execution.

Use Case: End Transaction

The mobile user finishes kangaroo transaction execution.	The system commit joey transaction and kangaroo transaction execution respectively

3.2.3 Transactions Terminology

A mobile transaction is processed as a sequence of sub-transactions and execution moves with the MU. Transaction operations executed at a MSS are called Joey Transactions. A sequence of Joey Transactions defines a Kangaroo Transaction.

3.2.4 Processing Modes

There are two different processing modes for Kangaroo Transactions. Compensating Mode and Split Mode. The system (simulator) supports split mode. 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.

3.2.5 Assumptions and Limitations

- According to Kangaroo model (KM), if a handoff occurs, new MSS communicates with the previous MSS to get transaction status information. However, in Modified Kangaroo model (MKM), the new MSS will get transaction status information directly from the central database.
- We simulate a wireless network on an Ethernet network by closing an existing TCP connection and creating a new TCP connection to the new MSS, an extra delay is introduced to account for the “handoff” delay.
- Only four MSSs and one mobile user are used to demonstrate the execution of mobile transaction.
- It is assumed that mobile user knows the SQL syntax.

CHAPTER 4

DESIGN AND IMPLEMENTATION OF KANGAROO TRANSACTION MODEL

In this chapter, simulation tool is described. Client layer, middle layer, server layer, system testing and experimental result are explained.

4.1 General Overview of the Simulation Tool

4.1.1 Logical Architecture

The logical architecture for the simulation is shown in Figure 6.

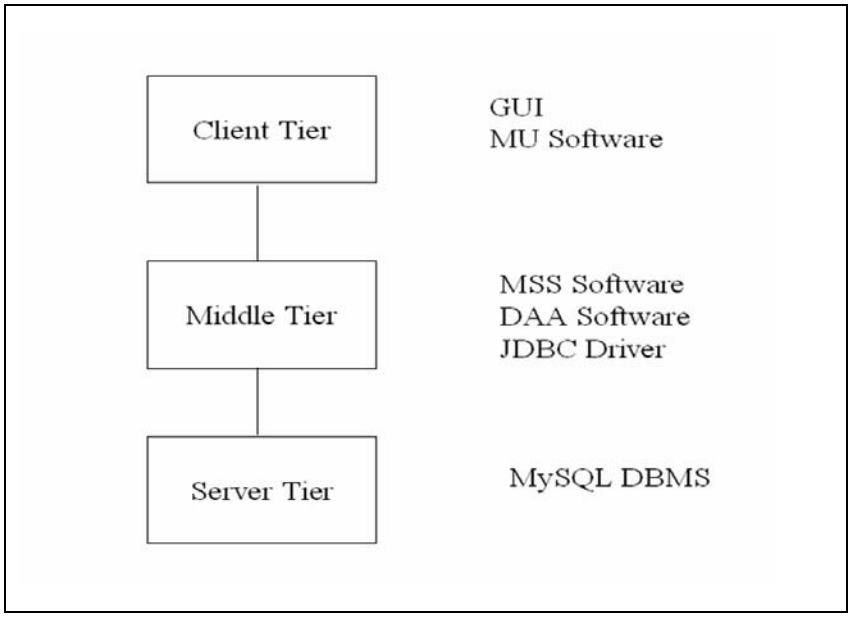


Figure 6: Logical Model

The architecture consists of three tiers representing the client layer, the middle layer, and the server layer.

The client layer is shown in the model as a MU connected to the system using a portable computer. The user interacts with the system through a graphical user interface written for the simulation tool. The interface enables the user to connect to the system, access data, and simulate movement in the network.

The middle layer represents the MSS. Each MSS consists of software installed on a office computer. The MSS accepts client connections from a MU on a specified port using java remote method invocation (RMI). Also located in the middle layer is software to implement the DAA. The DAA utilizes the fixed network to open a JDBC connection to the DBMS. The MSS acts as a data client by invoking DAA methods to query and manipulate the data. The DAA acts as a data server and returns result sets to the client.

The server layer consists of a windows machine located off office with the DBMS installed. Connectivity to the DBMS is enabling through a specified port. The DAA software contains parameters to identify the port, user account, password, and database name to open a connection.

4.1.2 Communication between MU and MSS

Communication between a MU and MSS is achieved by java remote method invocation. The public methods accessible by a MU and implemented by the MSS class are defined in an interface file MSSBaseServer. Associated with each script keyword is a method in the interface. During script execution, the MU creates a MSS object, looks up the name in the registry service, and invokes the method associated with the current line. The MSSBaseServerImpl class contains methods to read each line in the script file, write results to the transactions result window.

4.1.3 Mobile User Migration Scheme

A MU may migrate from one MSS to another at any frequency. Mobility is simulated in a script file using the MOVE keyword. When a user moves to a new MSS,

the site is responsible for remaining kangaroo transaction execution. The user must only provide the new MSS with its login id and password to establish a new connection. The new MSS responds to the connection request by performing a user validation security check. Upon successful validation, the new MSS create new joey transaction and resumes transaction execution.

4.1.4 Disconnection Operation Scheme

A MU may disconnect from the network at anytime. In real world scenarios, the disconnection may be voluntary or involuntary due to communication failure. Disconnection is considered a temporary interruption of transaction processing. The interruption may last a few seconds, minutes, or hours depending upon the cause. It is unusual and unlikely for a communication failure to result in an interruption of service lasting days. Transaction processing resumes when the mobile user reconnects to the network. To simulate disconnection and thus suspension of any transaction the script keyword list includes DISCONNECT. During script execution, the keyword DISCONNECT results in the immediate change of the kangaroo and joey transaction status to DISCONNECTED. No site transactions may be added to the joey transaction until the user reconnects. The system allows the MU five minutes to reconnect and resume the transaction. If the user fails to reconnect within the time limit, the transaction is aborted.

4.1.5 Script Execution

Each script execution begins with the user pressing the **'Execute'** button and invoking the appropriate method in the MSSBaseServer interface. A description of each script is provided below and the Appendix contains sequence diagrams for each script keyword.

.

4.1.5.1 CONNECT

The connect method is used to establish a connection with an MSS. The method accepts two parameters for the login id and password. The return value is a boolean representing the success or failure of the request. Any MSS may receive a connection request. The MSS calls local methods validate the mobile user. The DAA method setUserConnect queries the login table for a record matching the login id and password parameters. The getCell_id method returns the cell_id value. If the parameters are valid connection is success. The method isUserAuthorized check. User hashtable, if connection already exists ,responses 'connection exists'. Activity diagram of 'CONNECTION' script is shown in Figure 7.

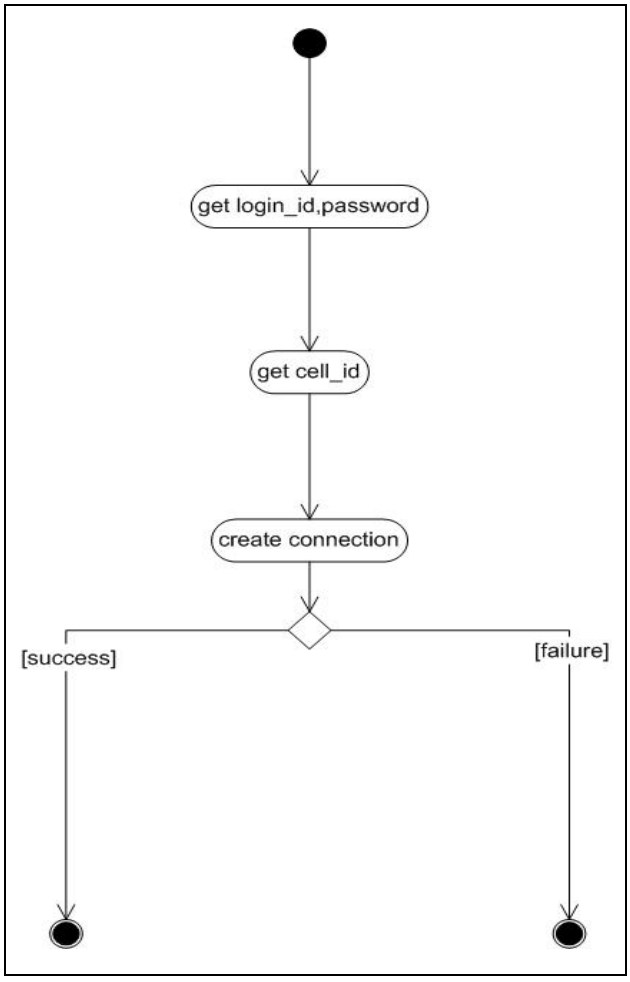


Figure 7: Activity diagram of 'CONNECTION' script

4.1.5.2 BEGIN

Begin indicates the start of a kangaroo transaction. Any MSS may receive a request to begin a transaction. The begin method create kangaroo_id , joey_id , get current time and insert related entries user_trans and joey_trans table.Returns a boolean representing success or failure of the request. Activity diagram of 'BEGIN' script is shown in Figure 8.

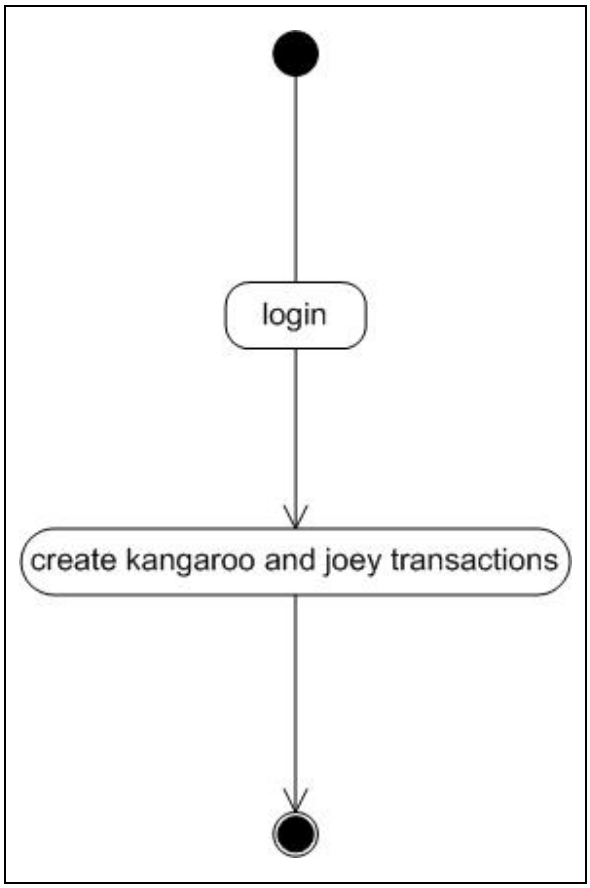


Figure 8: Activity diagram of 'BEGIN' script

4.1.5.3 MOVE

The move method is used to establish a connection with other MSS. The method accepts two parameters for the login id and password. The return value of type boolean represents the success or failure of the request. Any MSS may receive a move request. The logic is similar to that of the connect method. The method calls local methods to validate the mobile user and, set connection .The same methods are called to access the DBMS. Activity diagram of 'MOVE' script is shown in Figure 9.

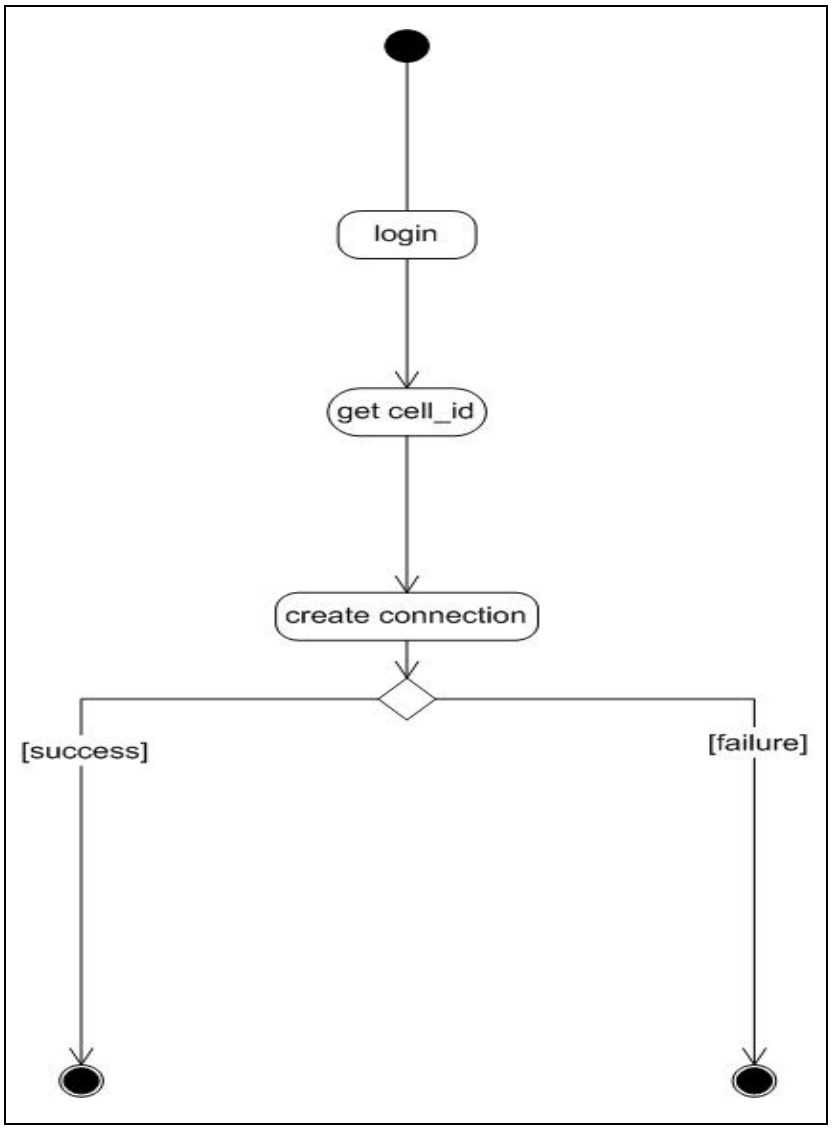


Figure 9: Activity diagram of 'MOVE' script

4.1.5.4 DISCONNECT

The disconnect method is used to simulate interruption in transaction processing. The method accepts a single parameter representing the login id. The return value is of type boolean representing the success or failure of the request. Any MSS may receive a disconnection request. The method calls a method disconnectionControl to set Timer and update user_trans table status field DISCONNECTED(2).If mobile user reconnect limited time , transaction continue. Otherwise, transaction aborted. Activity diagram of 'DISCONNECT' script is shown in Figure 10.

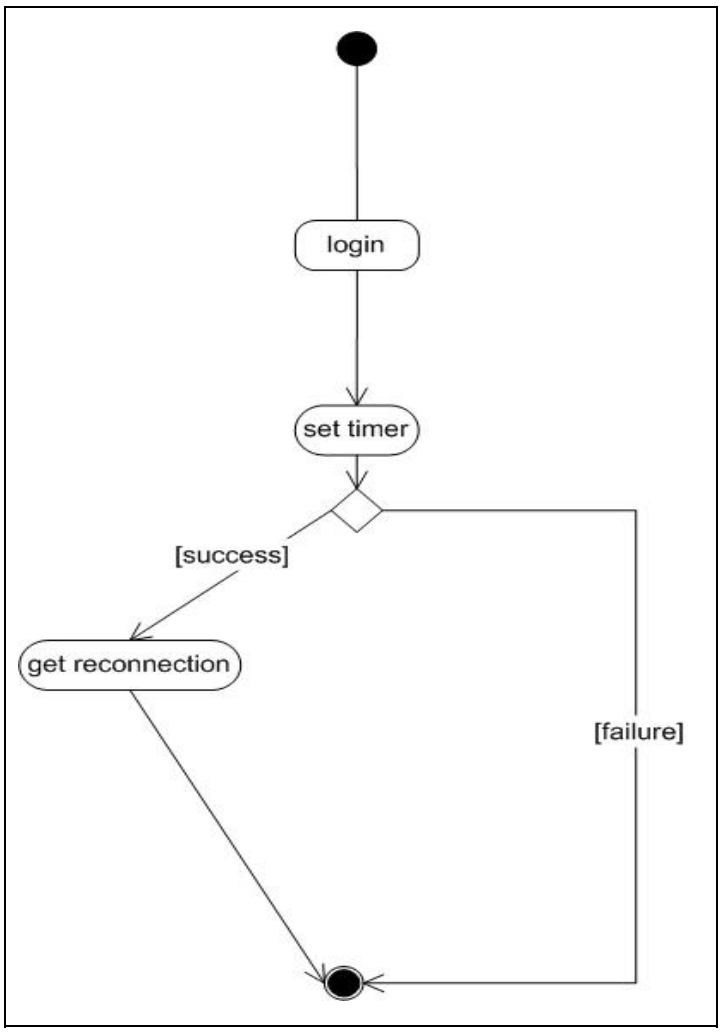


Figure 10: Activity diagram of 'DISCONNECTION' script

4.1.5.5 SELECT

The select method is used to query the database. The method accepts parameter values and returns a value of type String. The logic begins with lookup the user_trans and joey_trans table. If kangaroo transaction is active and joey transactions is not committed, an instance of the SiteTrans class is constructed with status ACTIVE, after generating a site transaction identifier and its object is added to the site hashtable through a call to the put method. On the other hand, if joey transaction is committed then a new joey transaction is created and other operations are performed as mentioned in the previous process. After execution, the query the method execute_query receives a result string. The site transaction status is updated to COMMIT in the local site hashtable. The result string is returned to the calling program. Activity diagram of 'SELECT' script is shown in Figure 11.

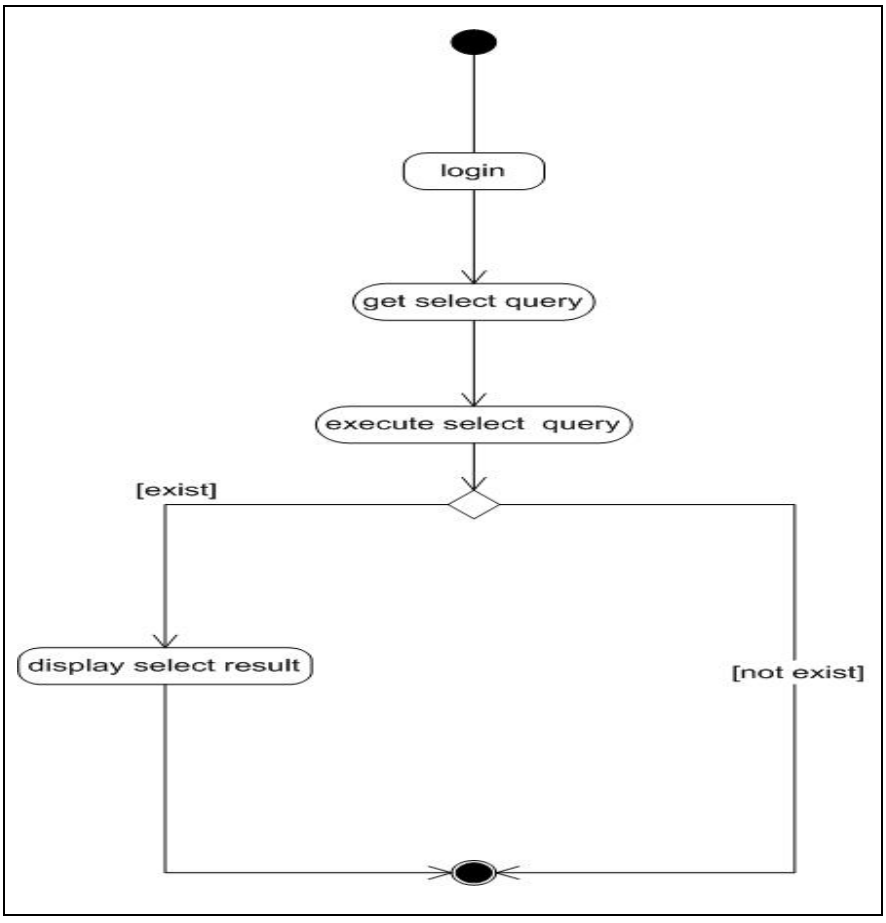


Figure 11: Activity diagram of 'SELECT' script

4.1.5.6 INSERT

The insert method is used to add records to tables in the DBMS. The method accepts insert statement request and returns a value of type boolean representing the success or failure of the request.

4.1.5.7 UPDATE

The update method is used to change records to tables in the DBMS. The method accepts update statement request and returns a value of type boolean representing the success or failure of the request.

Activity diagram of 'INSERT' and 'UPDATE' script is shown in Figure 12.

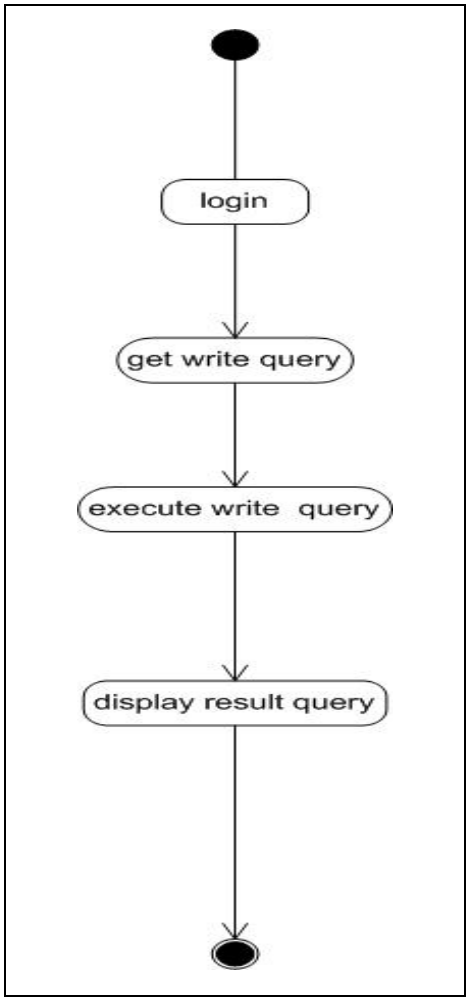


Figure 12: Activity diagram of 'INSERT and UPDATE' script

4.1.5.8 SAVE

The ‘SAVE’ script state ending of kangaroo transaction. The method accepts the login id as a parameter and returns a boolean value. Any MSS may receive a save request.The local MSS commits it’s joey transaction, kangaroo transaction and disconnect database. Activity diagram of ‘SAVE’ script is shown in Figure 13.

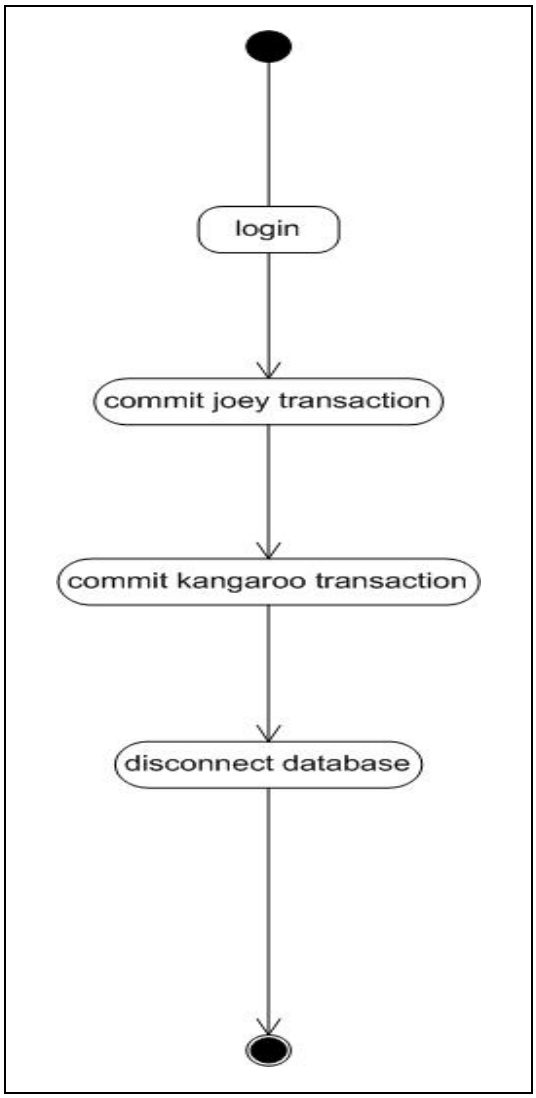


Figure 13: Activity diagram of ‘SAVE’ script

4.1.6 Client Layer

The client layer consists of software developed to enable data access, simulate user mobility, and simulate common characteristics of mobile computing

4.1.6.1 Graphical User Interface

User is mobile. The features and windows are described below.

4.1.6.1.1 Login Window

A login window is included in the user interface to capture a login id and password pair. A snapshot of the window is shown in Figure 14 below. The password value is entered in a java password field and encrypted using an alphabet substitution scheme. Both the login id and encrypted password are parameters in the CONNECT and MOVE script commands. The encryption scheme utilized is primitive yet provides a level of security adequate for this project.

Figure 14: Login Window

4.1.6.1.2 Network Mobility Window

Mobility is simulated through network commands CONNECT and MOVE. The base stations window shown in Figure 15. The name of each office computer running the MSS programs is shown. Each workstation defines the boundaries of a simulated

network cell. The script command, login name and password values, and cell selected are written to the script text area in the transaction keywords, edit and display window (Figure 16) when the user clicks on the okay button. Clicking on the cancel button does not write to the script text area.

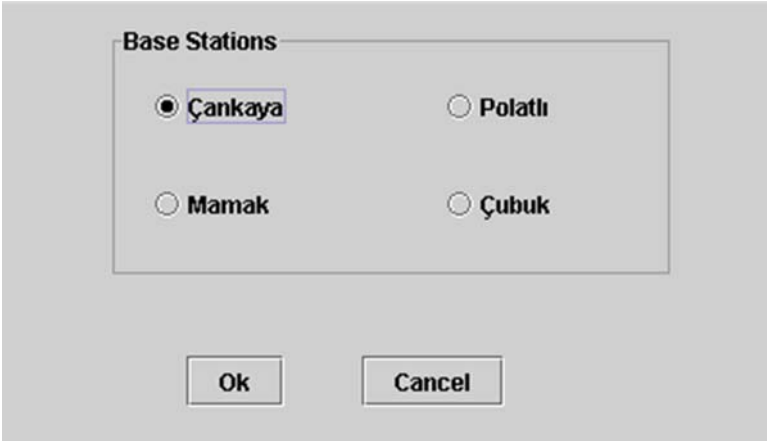


Figure 15: Base Stations Window

4.1.6.1.3 Transaction Keywords, Edit and Display Result Window

Transaction Keywords panel located in the top, left of the main window contains the list of transaction keywords. A script line is created by the action of selecting keywords from the list. Several characteristics of the mobile computing environment are simulated through specialized scripting commands. The scripting commands simulate user mobility between cells, disconnection, and data access. Changes to the underlying data model will not result in changes to the scripting commands. The user need only know the syntax of the scripting commands to write valid scripts. Traditional transaction processing actions such as begin, abort, commit, and rollback are available to users in the scripting commands. The reserved words, syntax, and meaning of each scripting command is provided in Table 4.

When the user clicks on the Execute button in the script text panel, the system call method execute_guery and writes result of each script line the transactions result window. The result value of most lines is a SUCCESS or FAIL status message.

Table 4: Script Language Commands

CONNECT	Open connection to MSS with specified cell_name for user with login id userName and password password .
BEGIN	SQL BEGIN transaction statement. Begin the kangaroo transactions.
MOVE	Simulate user mobility by opening a connection to MSS specified by cell_name for user userName and authenticated by password .
DISCONNECT	Send request to close connection.
SELECT	SQL SELECT statement with attribute and predicate values.
INSERT	SQL INSERT statement to insert record in table breakdown_log.
UPDATE	SQL UPDATE statement to update record(s) in table breakdown_log and breakdown_profile.
SAVE	SQL COMMIT statement. Terminate the Kangaroo transaction execution.

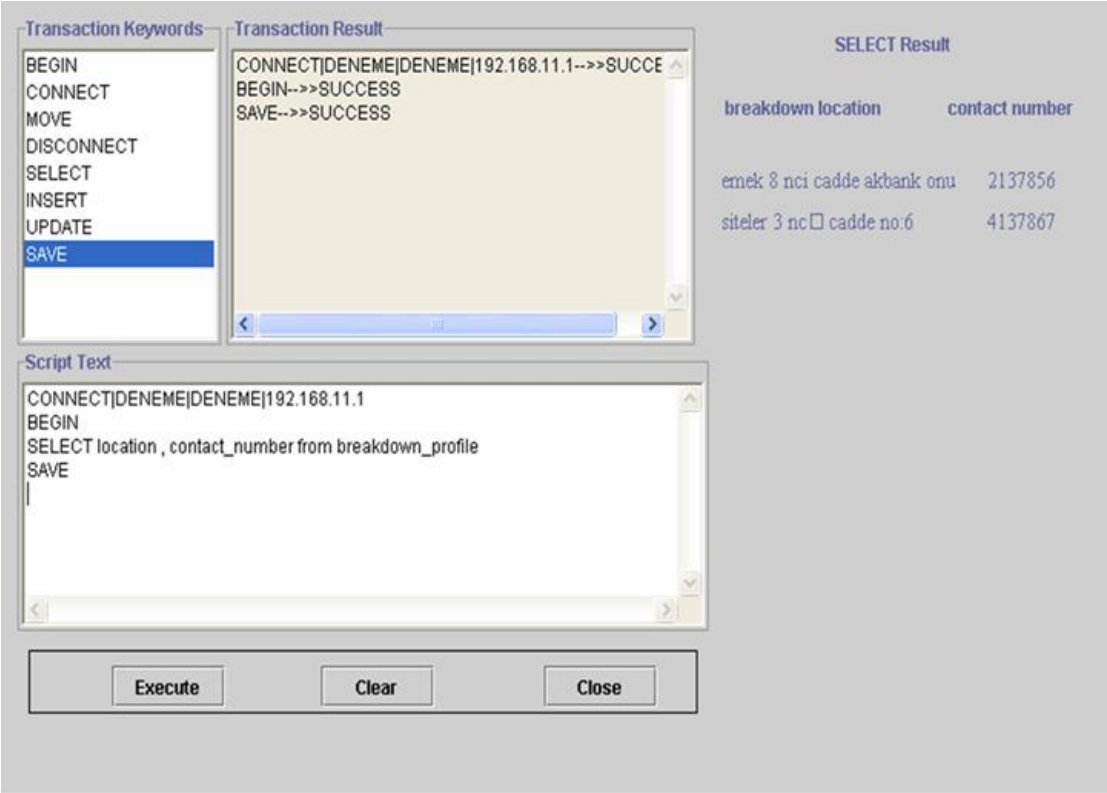


Figure 16: Transaction Keywords, Edit and Display Result Window

4.1.7 Middle Layer

The middle layer consists of software developed for the MSS and DAA functions.

4.1.7.1 Mobile Support Station (MSS)

The MSS implementation consists of class and interface files to perform transaction management operations. These operations included tracking transaction status, invoking methods to read and write data to the DBMS, and coordinating resumption and recovery operations when failure occurs. To fail safe operations and provide audit trail the MSS maintains logs for all active and completed mobile transactions. The MSS acts as a data client and forwards requests from MH for data to the DAA. All result sets are returned to the MH. Communication between the MH and

MSS is in the form of remote method invocation and scripting commands. Network handoff in response to MH mobility is accomplished through remote method invocation from one MSS to another.

4.1.7.2 Data Access Agent (DAA)

The DAA is responsible for opening a connection to the DBMS, executing SQL commands, closing the connection, and returning the result set. An instance of the DAA class is created during construction of a MSS object. Connection parameters such as user login account and password information are passed to the DAA constructor. The DAA class utilizes JDBC to connect to a data source and Java SQL package methods to execute statements.

4.1.7.3 Initialization

With the registry service running, an MSS is started by typing on the command line the following: java MSSBaseServerImpl <cellname> .The parameter cellname assign to the MSS a cell name.The cell name used for the project is the Sun workstation name. The port specified in the project is 1099.

An MSSBaseServer object is created when the main program calls the constructor with the parameter value entered on the command line. The value is assigned to variable to store the local cellname. The constructor also creates an instance of the DAA class for data access and three instances of the HashTable class to store transaction data. The DAA constructor is called with values for the database user and password as parameters. The three HashTable objects represent the kangaroo, joey, and site transaction data storage needs. All HashTable objects are declared as private data members.

4.1.7.4 Transaction Management

Transaction management is the primary function of the MSS. As script execution proceeds, each MSS involved must store and share sufficient data to perform resumption, commit, and recovery operations. Transaction data values representing the User, Kangaroo, Joey, and Site entities must be stored both in memory and log file format.

The User entity contains values specific to each MU. The Kangaroo entity contains values for the global transaction execution. The Joey entity contains values for the actions executed at the local cell. Each MSS contains one data structures to store each entity. The user hash data structure contain if there is exist connection and kangaroo hash data only contain value for transaction executed by subscribers assigned to the MSS. Thus a MSS will only store user and global data values for its subscribers. The values reside in the data structures in memory until the transaction terminates. Class files to represent each entity were created for the project. The class files User, Hashtable_kangaroo contain methods and variables to construct an instance. Properties of each entity are shown in Table 5 below. The User class contains properties to described data values associated with a subscriber. The key values are the subscriber Id, identifier of any global transaction associated with the subscriber, and the transaction status. The compensating transaction data is utilized when the site receives a request to abort the transaction and undo all changes.

Table 5: Transaction Data Values

Entity	**Property**	**Description**
user	userName	login id
	password	Password
hashtable_kangaroo	record _id	
	id	Transaction ID
	type	Kangaroo, joey, site
	status	Transaction status
	pr_joey_id	Previous joey transaction id
	next_joey_id	Next joey transaction id

4.1.7.4.1 Status values

Associated with each kangaroo, joeys and site transaction are status values. The MSS utilizes the status values when coordinating resumption, save, and recovery operations. Table 6 lists status values and meanings.

Table 6: Transaction Status Values

Status Code	Status Display	Meaning
ACTIVE	1	User connected, transaction is executing
DISCONNECTED	2	User disconnected, reconnection expected
COMMIT	3	Transaction committed successfully
ABORT	4	Transaction aborted

4.1.7.4.2 Data structures

Data storage needs required the choice of a structure to efficiently perform lookup, insertion, and deletion operations. The data structure chosen for the project is a hash table implemented as an array of linked lists. The structure is implemented in the class file HashTable and instantiated in the MSSBaseServerImpl constructor. For user the hash function applied utilizes the login id and for Hashtable_kangaroo function utilizes the record id as the key. The linked list is navigated in both a forward and backward fashion depending upon the type of lookup request using a list iterator. Insertion operations always add to the end of the list using methods provided in the Java util package.

4.1.7.4.3 Transactions ID generation

The MSSBaserServerImpl class defines a property for assignment of a unique identifier for kangaroo, joey and site transactions. Identifier assignment involves a call

by MSS to the DAA to query the DBMS for the last value assigned and followed up with an update call after assignment. The database provides permanent storage and guarantees uniqueness. The uniqueness of the identifier is not compromised in the event of a MSS failure and restart. The table structure was chosen for storage. A database sequence is an alternative structure to consider if supported by the DBMS.

4.1.7.4.4 System Initiated Abort

The system will initiate an abort of a kangaroo transaction in the event a MU fails to reconnect and resume execution within the allowed time. The clock starts when the MU disconnects. The MSS logic updates the status in the kangaroo and joey hash structure to DISCONNECTED. The insertUserTrans method of the DAA class is invoked to insert a record in the user_trans table. The disconnectionControl class contains a subclass RemindTask which extends the Java TimerTask interface. disconnectionControl schedules the RemindTask run event to check the DBMS for expired user_trans records every 10 seconds. The RemindTask class calls the DAA selectUserTrans method to query the DB and return the kangaroo id and cell name of each expired transaction. If time is up the cell executes the steps necessary to abort the kangaroo transaction.

4.1.8 Server Layer

4.1.8.1 DBMS Installation

The DBMS utilized for the project is MySQL. MySQL is an open source DBMS that supports transactions with commit, rollback, and savepoint statements. The version downloaded for the project is version 4.0.14 and JDBC driver version 3.0.8. The database instance created for the project is called 'project'. All JDBC connection requests utilized port 3306. Administrator scripts to create the tables with constraints and insert initial data in the tables were written using SQL commands. The scripts may be run from within the MySQL command line interface. A special user account was created with password protection and granted privileges on the project database.

4.1.8.2 Database Model

The database model developed for the sample mobile application is based on a mobile auto repair service application. The model includes tables and relations between the entities.

Login table consists of entries (username, password) for authenticating the user. User_trans and joey_trans tables store the transactions' status data. Network cells are defined in the cell table and are identified with a cell name and cell_ id. Each subscriber is assigned to a cell and this cell is referred to as the home cell. A cell may be assigned zero or more subscribers.

The service is organized as a mobile auto repair service. Each mobile service has a unique id (service_id) and service area (service_area) entity, which states the responsibility area of the service.

Service records data into breakdown_profile table and notifies relevant auto repair service to fix when it gets a report for a car breakdown. In addition to that, the service may check the status on whether the mobile service successfully repaired the automobile or had it towed to a service center for further labor (status: 1= mobile service not reached yet, 2=malfunction fixed, 3=vehicle towed to the service center).

After repair, mobile service inserts a record into breakdown_log table. This includes description of repair done, bill data, date and time data. In addition to these, mobile service updates the status field in the breakdown_profile table as discussed in the previous paragraph.

The ER diagram for the database model and sql script to create the physical database is given in the Appendix of this thesis.

CHAPTER 5

EXPERIMENTAL SET UP AND RESULTS

In this chapter, we discuss how experiments were designed to be performed, and what results were obtained from these experiments. In our experiments, we tried to measure the effect of MKM on transaction performances.

5.1 Experimental environment in a LAN

The physical architecture for simulation tool is shown in Figure 17.

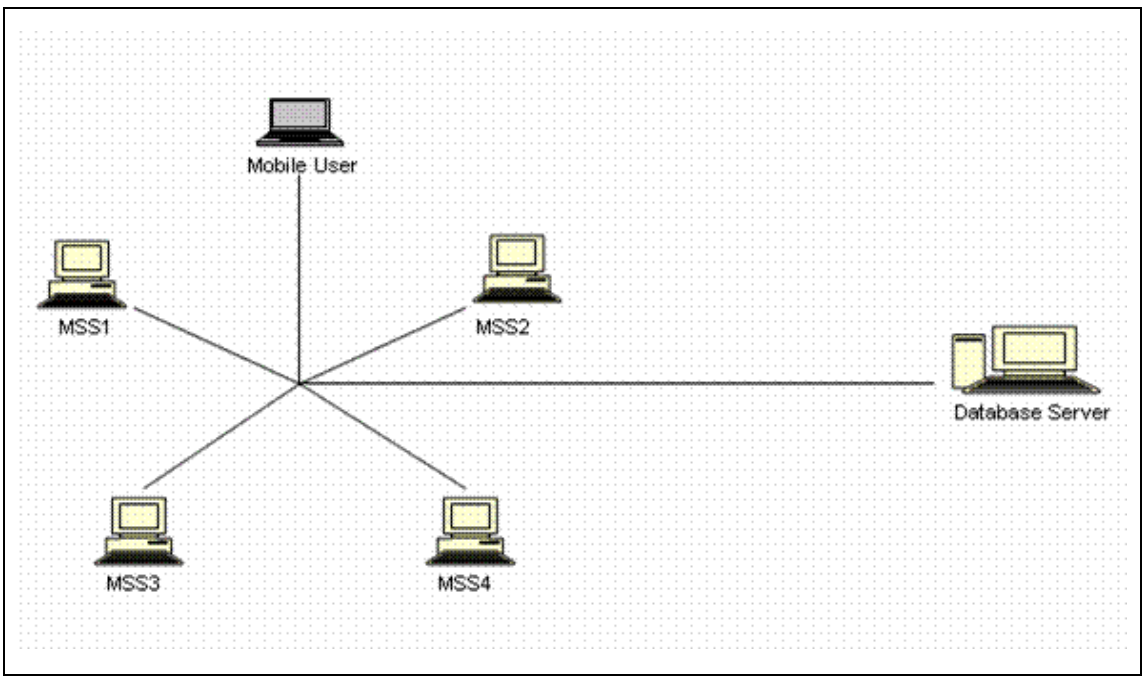


Figure 17: Physical Architecture

The office network partition includes four computers from the office labeled MSS1 to MSS4 in Figure 17 and one mobile user connected to the office network using portable computing devices (laptop).

Each computer represents a MSS in the mobile computing model. The coverage area or cell for each MSS is defined by the machine name and a specific port registered with the java naming service upon startup. The simulation tool simulates characteristics of the mobile environment such as movement between cells and disconnection. Movement between cells is simulated in the MU software using the registered port to open a new connection and send messages to the new MSS. Disconnection is also simulated in the software through method calls on the registered port. The port utilized for the communicating with MSSs is 1099.

The database server is a Desktop PC running Windows XP and the DBMS. The DBMS used in the tests is the open source MySQL version 4.0.14. The MySQL process listens for connection requests on port 3306.

The simulation tool supports multiple users and a configurable number of MSS. To facilitate debug and system testing, four computers were configured as MSS sites and one mobile user was present.

5.2 System testing

System testing involved executing tests to determine operation under different conditions. Table 7 summarizes the tests conducted and provides a reference to the script utilized. Hardcopy of each script is included in the Appendix-A.

Table 7: Test Summary

Test No.	Test description	System capability under Test	Script name
1	Connect to MSS, begin transaction operations, include mobility, and commit changes.	• Connectivity and mobility • Select/Insert/Update • Commit	Basic.txt
2	Execute transaction across multiple cells and resumption at remote cell.	• Disconnection • Resumption	Resume.txt
3	Attempt connection with invalid user id and password values.	• User validation • Scrip execution halts	Validate.txt
4	Begin a transaction and disconnect. Do not reconnect to resume. System will detect timeout and abort transaction.	• Resumption timeout • System generated abort Kangaroo transaction	Disconnection Control.txt
5	Transaction script for user with no mobility.	• Single MSS execution	ZeroMobile.txt
6	Transaction script for highly mobile user.	• Multiple MSS execution sites.	HighMobile.txt

5.3 Simulation parameters

In the experiments, we designed scripts with increasing number of Joey transactions, from 1 up to 4. In order to measure execution times accurately and minimize O/S interference, each script was run 10 times consecutively and total time was divided by 10 to obtain execution time of that script.

Table 8: Simulation Parameters

Parameters	Description
1 movement	Mobile user remains in the same MSS during the **"Kangaroo transaction"**, no handoff occurs.
2,3 and 4 movements	Mobile user changes connect to 2, 3 and 4 MSSs. That is to say 1, 2 and 3 handoffs occur.
Transaction execution time	Time difference between begin transaction and end transaction.
Site transaction	Sequence of read and write operations, all performed on the local DBMS.

For each script with a certain number of Joey transactions are also increase the number of site transactions from 4 to 6, 8, and 10. Thus, we can also observe the effect of having longer running transactions.

For simulating wireless handoff operations, we assumed it takes just the same time as required on a LAN. We also assumed no packet error would occur, while these would have to be considered in a more realistic simulation to better reflect the real world operation.

5.4 Experimental Results

The mobile transaction execution time was measured for both approaches and compared their execution times. The transaction start and transaction end times are stored in the central database. When a MSS executes '**BEGIN**' command in the transaction script, I insert the transaction starting time information into the "**user_trans"** table (using "**kangaroo transaction"** id) and after execution a '**SAVE**' command from

the transaction script, the transaction ending time information is updated in the **"user_trans"** table.

Two models (MKM and KM) are compared using transaction execution time under various MSS handoff scenarios. Comparison graphics for two models are illustrated in Figure 18 through Figure 20. The vertical axis shows transactions' execution time (in milliseconds) for **"kangaroo transaction"** script execution time. The horizontal axis represents the number of site transaction.

Times of transactions with varying number of site transaction vs. number of movements (in milliseconds) are shown in Table 9.

Table 9: Transaction Execution Time

movements	4 site trans.		6 site trans.		8 site trans.		10 site trans.	
	KM	MKM	KM	MKM	KM	MKM	KM	MKM
1	40	100	90	410	190	430	220	540
2	100	190	180	620	320	890	380	990
3	220	380	270	920	600	1390	670	1580
4	270	450	400	1040	730	1630	910	1820

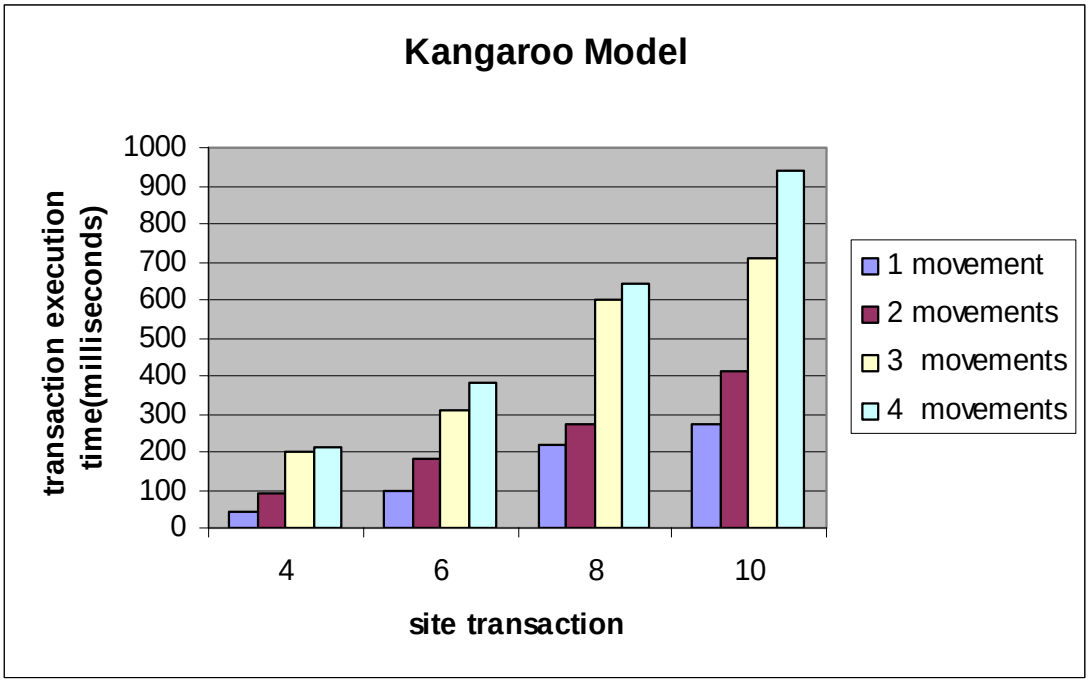


Figure 18: Kangaroo Model

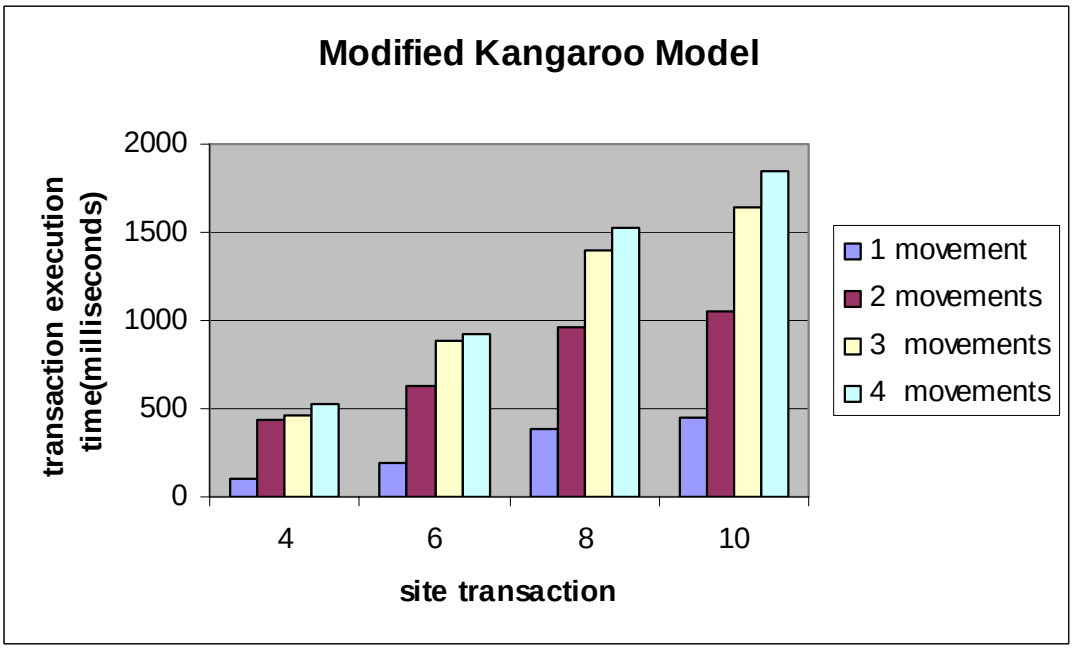


Figure 19: Modified Kangaroo Model

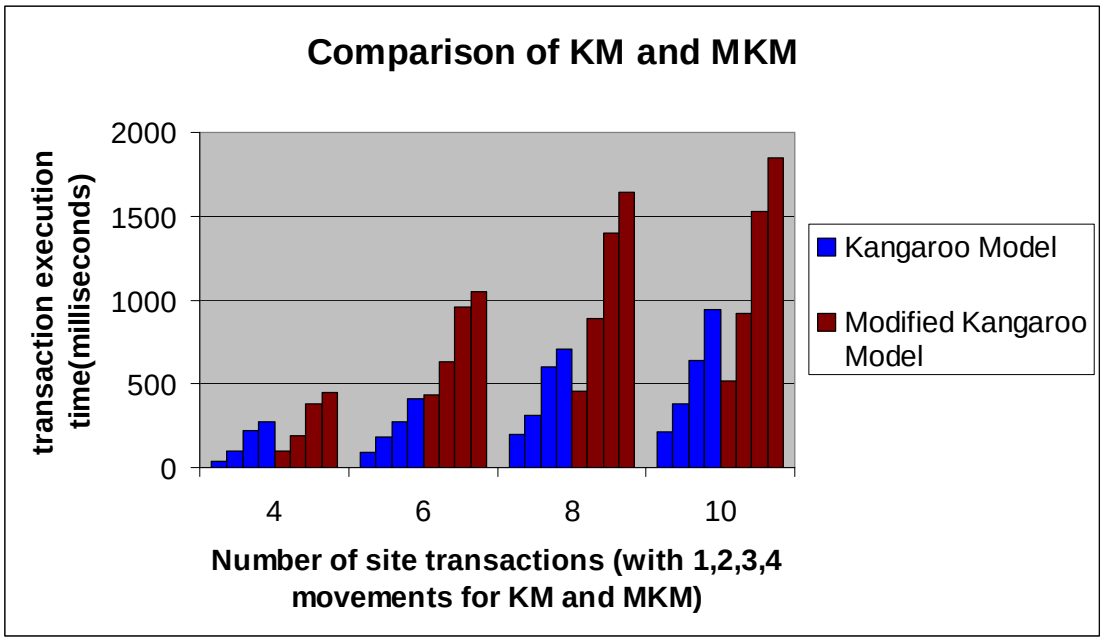


Figure 20: Comparison of KM and MKM

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

A simulation environment was developed, in which base stations are modeled as processes running on a computer(s) and serving connection requests. Once MU-MSS connection is established, the commands sent by the mobile host are performed.

In order to be able to perform compensating transactions when/if intermediate base stations fail, making it possible to determine all previous JT base stations, we proposed to store this information in a central database.

We measured, using simulated workload, the amount of overhead required by the Modified Kangaroo transaction model was compared with the original Kangaroo transaction model.

Our results showed that the overhead of MKT is not prohibitive. Also, as the transactions become longer the overhead of MKT becomes even a smaller percentage of the transaction execution time.

We were not able to simulate complete transaction, as that would require a complete system implementation. We have not able to obtain real-life communication parameters either. By using more accurate parameter values, more detailed analyses of Kangaroo transaction and other transaction models could be performed.

The following observations have been made:

- When movements increase, the transaction execution time also increases for both models.
- According to the Modified Kangaroo model (MKM), (MSS check transaction status information directly from the central database), transaction execution times are always longer than the Kangaroo model.
- Modified Kangaroo model is more resilient to base station failures than Kangaroo model. In Kangaroo model in case of any failure of a MSS, the **"kangaroo transaction"** execution cannot continue. However, Modified Kangaroo model, since the new MSS will get transaction information directly from a central database, it does not need any information from preceding MSS therefore the mobile user will be able to continue transaction execution in case the previous MSS fails.

6.1 Future work

Data management in the mobile computing environment is an area rich with research opportunities. This project focused on the issue of transaction processing and examined one of the many proposed models (Kangaroo transaction), also providing a small modification to this model.

This work considers the disconnected operation. All transaction execution occurred on the FH. The model enabled disconnected users to resume transaction processing but did not facilitate offline processing. Future work could propose a modified version of the Kangaroo model that supports transaction execution on the MH (while disconnected) and realizes compensating transactions.

REFERENCES

[1] Dunham, M.H., Helal, A., and Balakrishnan, S. "A Mobile Transaction Model that Captures Both the Data and Movement Behavior", ACM-Baltzer Journal on Mobile Networks and Applications , vol.2, no.2, October 1997, pp.149-161.

[2] Bukhres, O. and S. Morton. "Utilizing Mobile Computing in the Wishard Memorial Hospital Ambulatory Service." Proceedings of the 1997 ACM symposium on applied computing. 1997: 287-294.

[3] Budiarto, S. and M. Tsukamoto "Data management issues in mobile and peer-to-peer environments." Data & Knowledge Engineering. 41 (2002): 183-204.

[4] Madria, S.K, and Bhargava, B**."**A Transaction Model for Mobile Computing",

An International Journal, Kluwer Publishers, 1999.

[5] Oracle Corporation. Oracle Application Server Wireless: Complete Mobile Platform An Oracle Technical White Paper August 2003.

[6] Forman, G. H. and Zahorjan, J."The Challenges of Mobile Computing", IEEE Computer Volume: 27(4), pp. 38-47, April 1994.

[7] R. Alonso and H. F. Korth. Database System Issues in Nomadic Computing. In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pp. 388 – 392, 1993.

[8] Garcia-Molina, H. et al. Database Systems the Complete Book. New Jersey: Prentice Hall Inc, 2002.

[9] D.Chess, C.Harrison, A.Kershenbaum, “Mobile Agents: Are they a good idea?”,IBM Research Report, T.J.Watson Research Center, NY, 1995.

[10] Garcia-Molina, H. and K. Salem. "Sagas." Proceedings of the ACM SIGMOD 1987. 16 (1987):249-259.

[11] Yeo, L. H. and A. Zaslavsky. "Submission of Transactions from Mobile Workstations in a Cooperative Multidatabase Processing Environment." Proceedings of the 14 th International Conference on Distributed Computing Systems. 1994: 372-379.

[12] Chrysanthis, P. K., “Transaction Processing in Mobile Computing Environment”, In IEEE Workshop on Advances in Parallel and Distributed Systems, pages 77-82, October 1993.

[13] Elmasri, R. and Navathe, S.[2000],”Fundamental of Database Systems”, Third Edition,Addison-Wesley,2000.

[14] Dunham, M.H.,Helal, A., “Mobile Computing and Databases: Anything New?”, ACM SIGMOD Record vol.24, no.4, December 1995, pp.5-9.

[15] Dunham, M.H.,Kumar, V., “Impact of Mobility on Transaction Management”, Proceedings of the International Workshop on Data Engineering for Wirelss and Mobile Access, MobiDE ’99, Seattle, WA, USA, August 1999, pp.14-21.

[16] Pitoura, E. and Bhargava, P., “Revising transaction Concepts for Mobile Computing”. Proceedings of the Workshop on Mobile Computing Systems and Applications, 1994, pp. 164-167.

[17] Pitoura, E. and Bhargava, P.. “Maintaining Consistency of Data in Mobile Distributed Environment.” 15th Int. Conference on Distributed Computing Systems, Vancouver, Canada, May, 1996.

[18] Walborn, D. G., and Chrysanthis, P. K. “Supporting Semantics Based Transaction Processing in mobile Database Applications”. Proc. 14th IEEE Symp. on Reliable Distributed Systems, September 1995.

APPENDIX

Appendix A: System Testing Scripts

Test No. 1: basic.txt

CONNECT|test|abc123|192.168.11.1

BEGIN

SELECT location , contact_number from breakdown_profile

INSERT INTO breakdown_log valueS(‘15’,’rnt1’,’breakdown is solved’,’12/03/2004’, 50000000,’23’)

UPDATE breakdown_profile SET service_status=’2’ where id=’12’

MOVE

CONNECT|test|abc123|192.168.11.128

SELECT location , contact_number from breakdown_profile where service_status=’1’

SAVE

Test No. 2: resume.txt

CONNECT|test|abc123|192.168.11.1

BEGIN

SELECT location , contact_number from breakdown_profile

INSERT INTO breakdown_log values(‘15’,’rnt1’,’breakdown is solved’,’12/03/2004’, 50000000,’23’)

DISCONNECT

CONNECT|test|abc123|192.168.11.128

SELECT location , contact_number from breakdown_profile

UPDATE breakdown_profile SET service_status=’2’ where id=’12’

SELECT location , contact_number from breakdown_profile

SAVE

Test No. 3: validate.txt

CONNECT|test|abc123|192.168.11.128

BEGIN

SELECT location from breakdown_profile where service_status=’1’

SAVE

Test No. 4: DisconnectionControl.txt

CONNECT|test|abc123|192.168.11.1

BEGIN

SELECT location from breakdown_profile where service_status=’1’

INSERT INTO breakdown_log valueS(‘15’,’rnt1’,’breakdown is solved’,’12/03/2004’, 50000000,’23’)

DISCONNECT

Test No. 5: zeromobile.txt

CONNECT|test|abc123|192.168.11.128

BEGIN

SELECT location from breakdown_profile

INSERT INTO breakdown_log valueS(‘15’,’rnt1’,’breakdown is solved’,’12/03/2004’, 50000000,’23’)

SELECT contact_number from breakdown_profile

SAVE

Test No. 6: highmobile.txt

CONNECT|test|abc123|192.168.11.1

BEGIN

SELECT location ,contact_number from breakdown_profile

INSERT INTO breakdown_log valueS(‘15’,’rnt1’,’breakdown is solved’,’12/03/2004’, 50000000,’23’)

MOVE

CONNECT|test|abc123|192.168.11.128

SELECT location ,contact_number from breakdown_profile

MOVE

CONNECT|test|abc123|192.168.11.129

UPDATE breakdown_profile SET service_status=’2’ where id=’12’

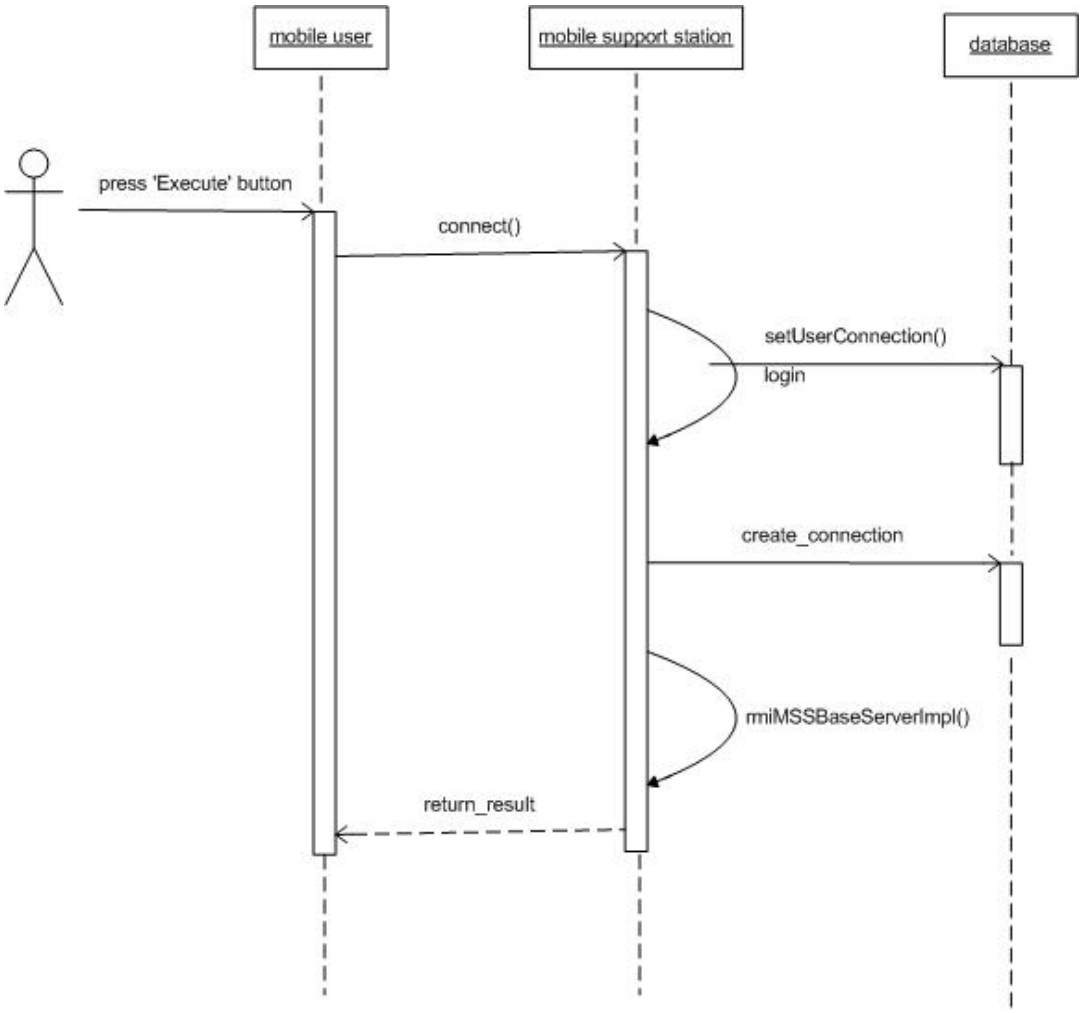
MOVE

CONNECT|test|abc123|192.168.11.130

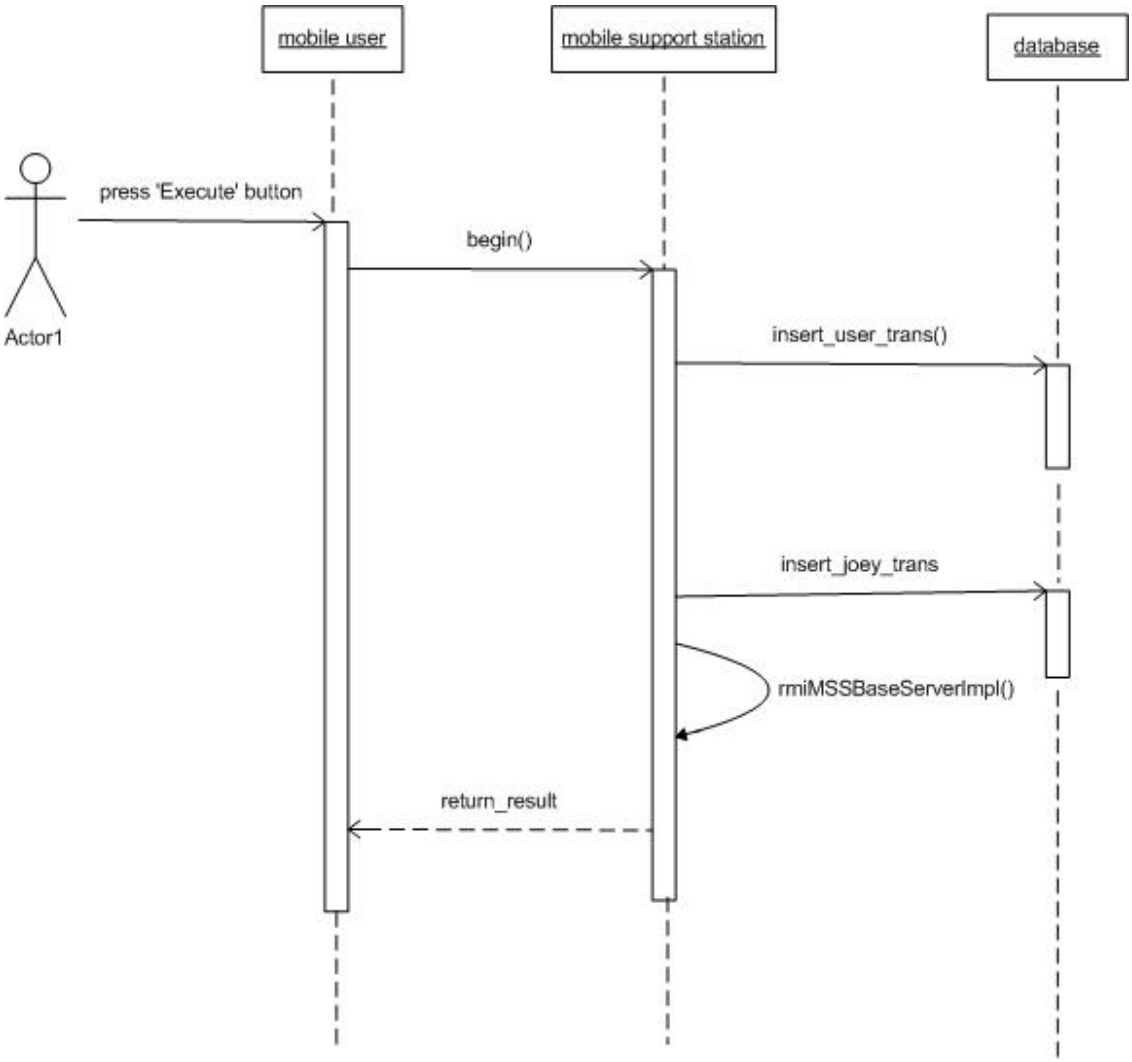
SELECT location ,contact_number from breakdown_profile

SAVE

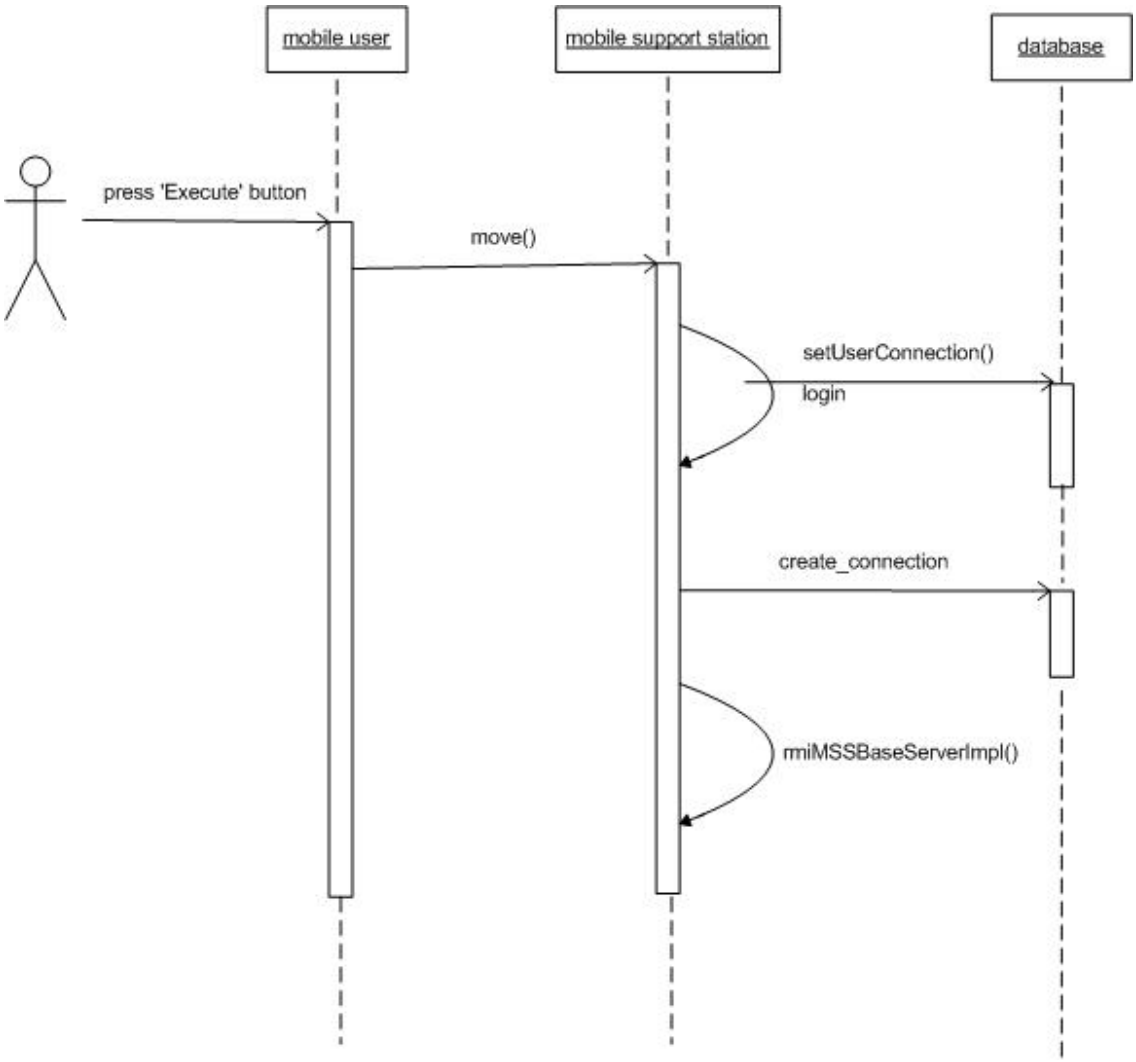
Appendix B: Sequence diagram for ‘CONNECT’ keyword



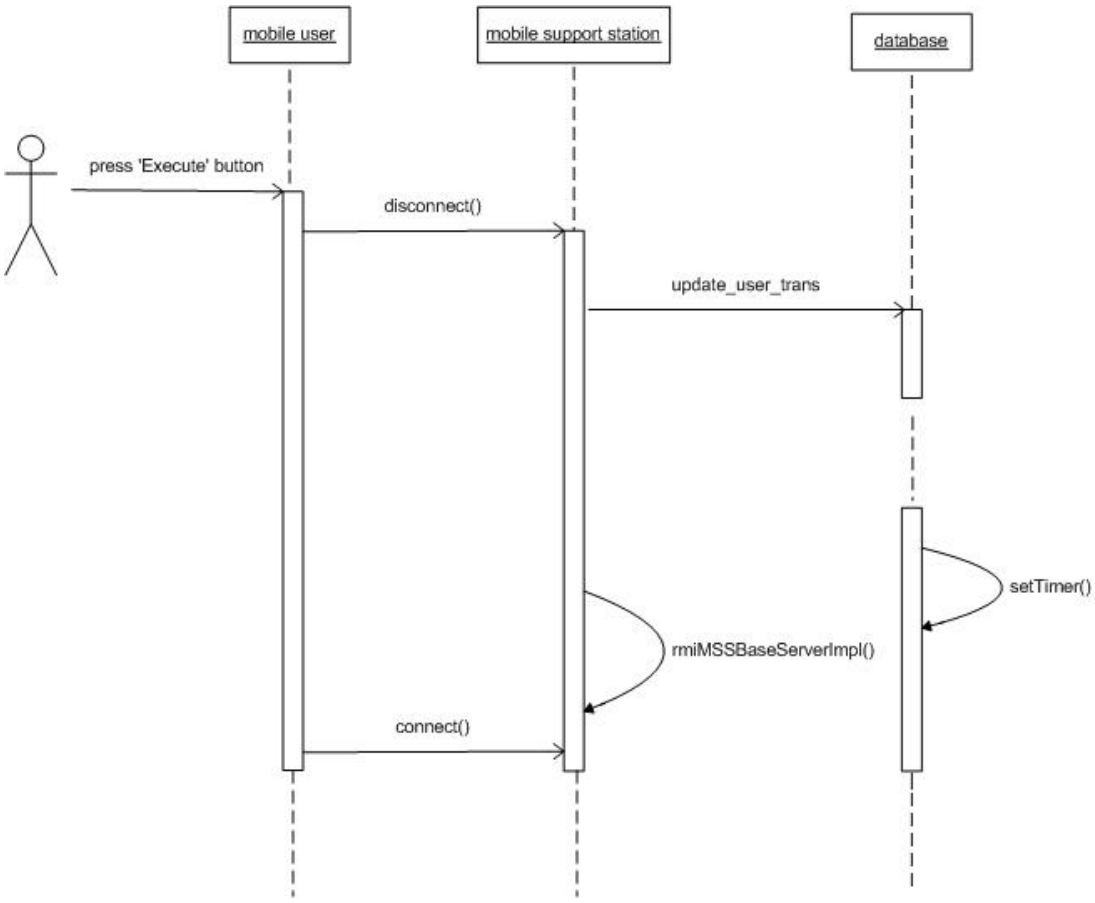
Appendix C: Sequence diagram for 'BEGIN' keyword



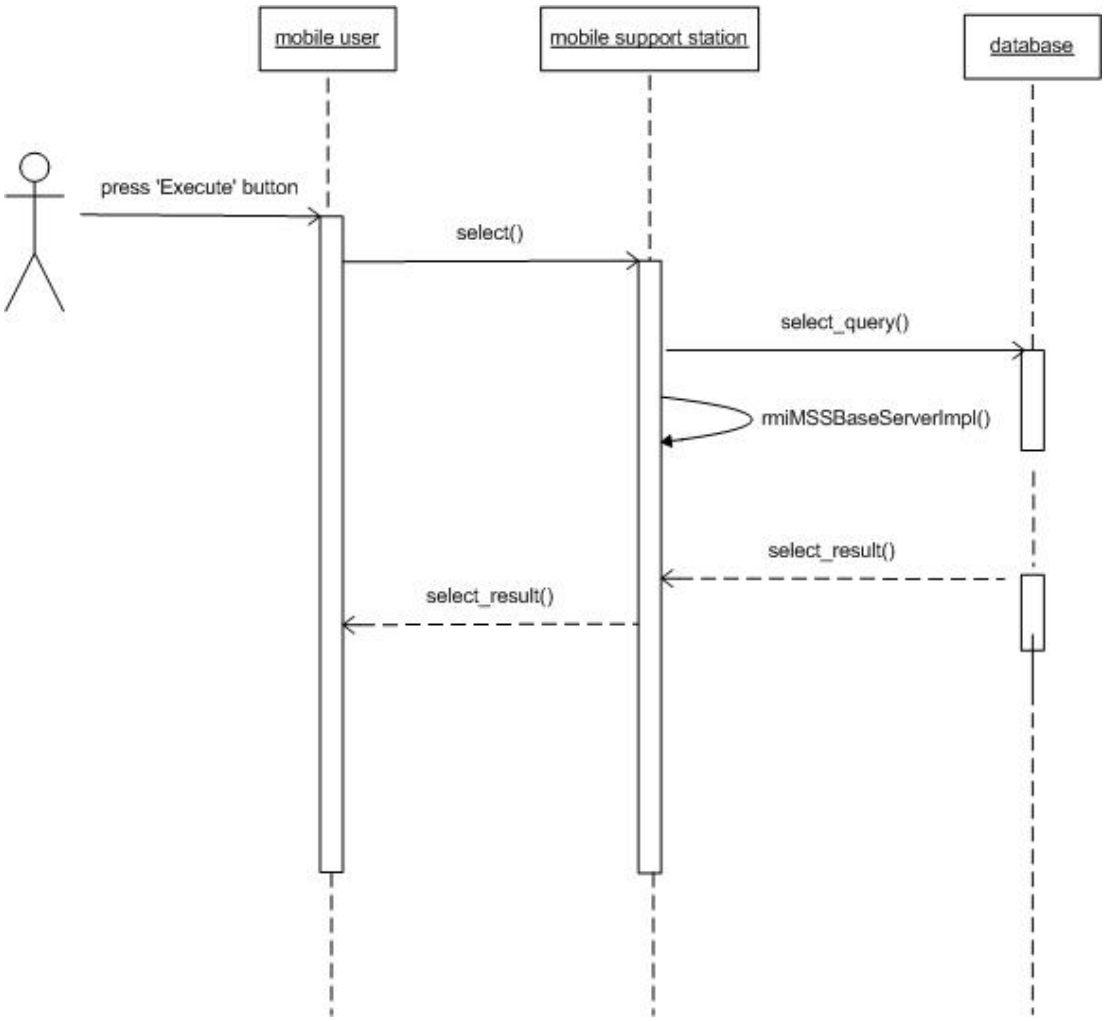
Appendix D: Sequence diagram for ‘MOVE’ keyword



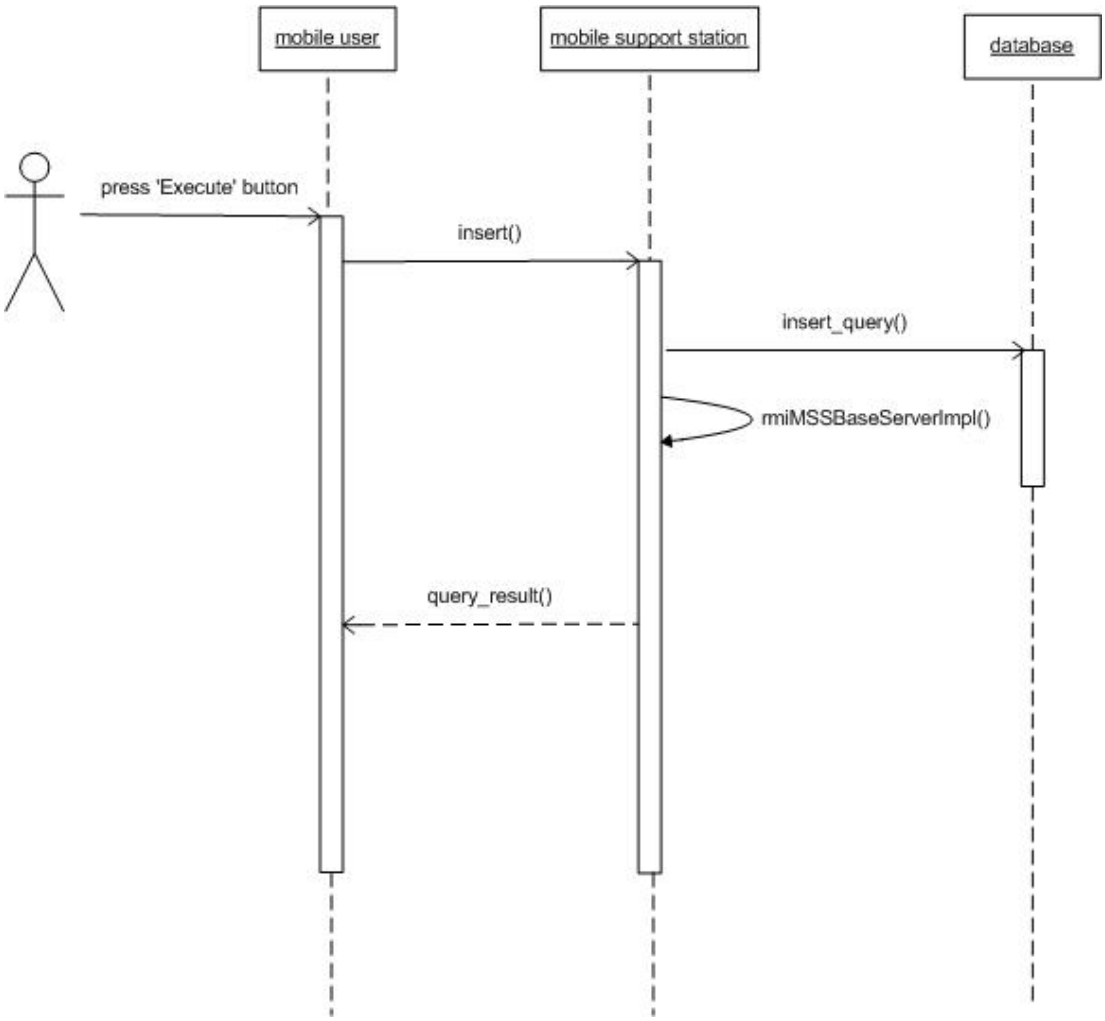
Appendix E: Sequence diagram for ‘DISCONNECT’ keyword



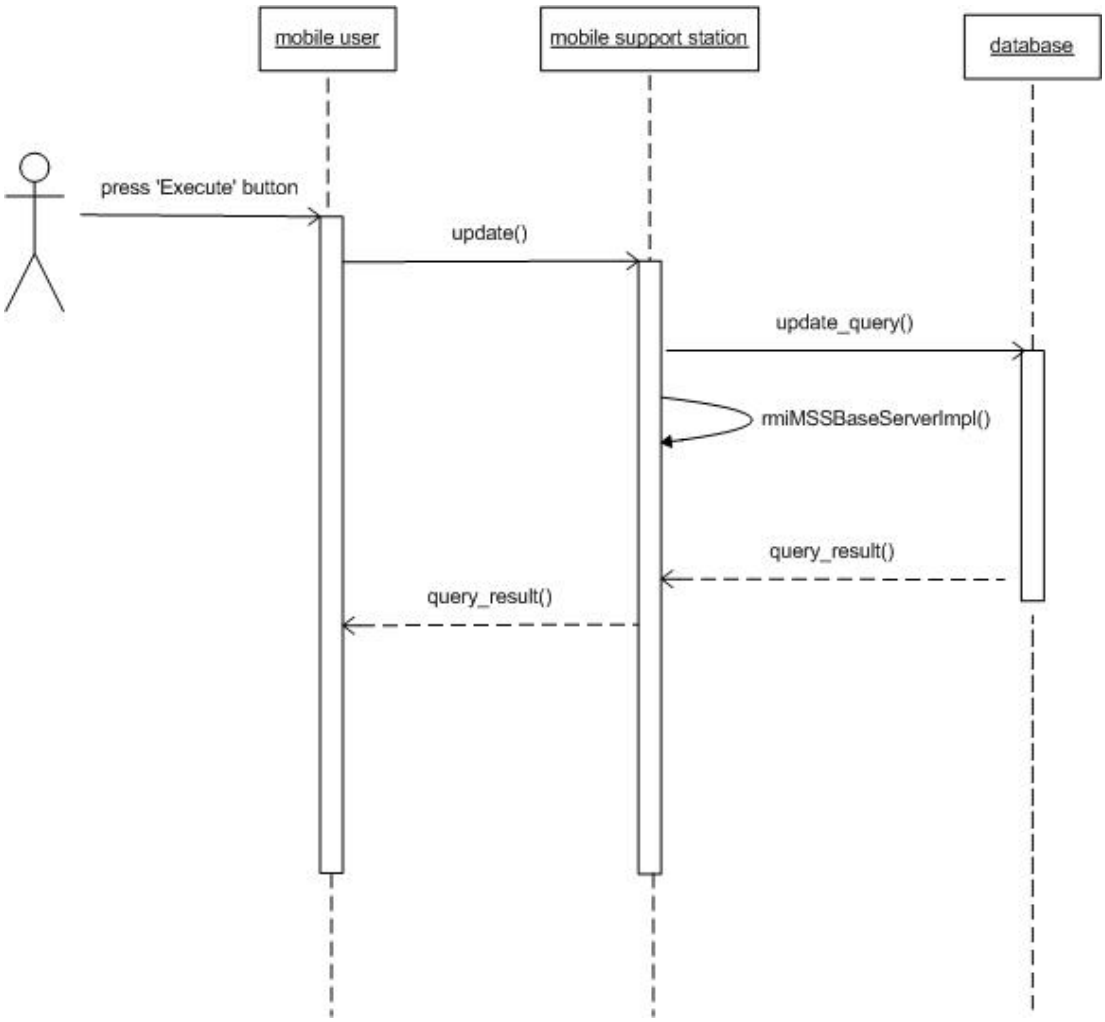
Appendix F: Sequence diagram for ‘SELECT’ keyword



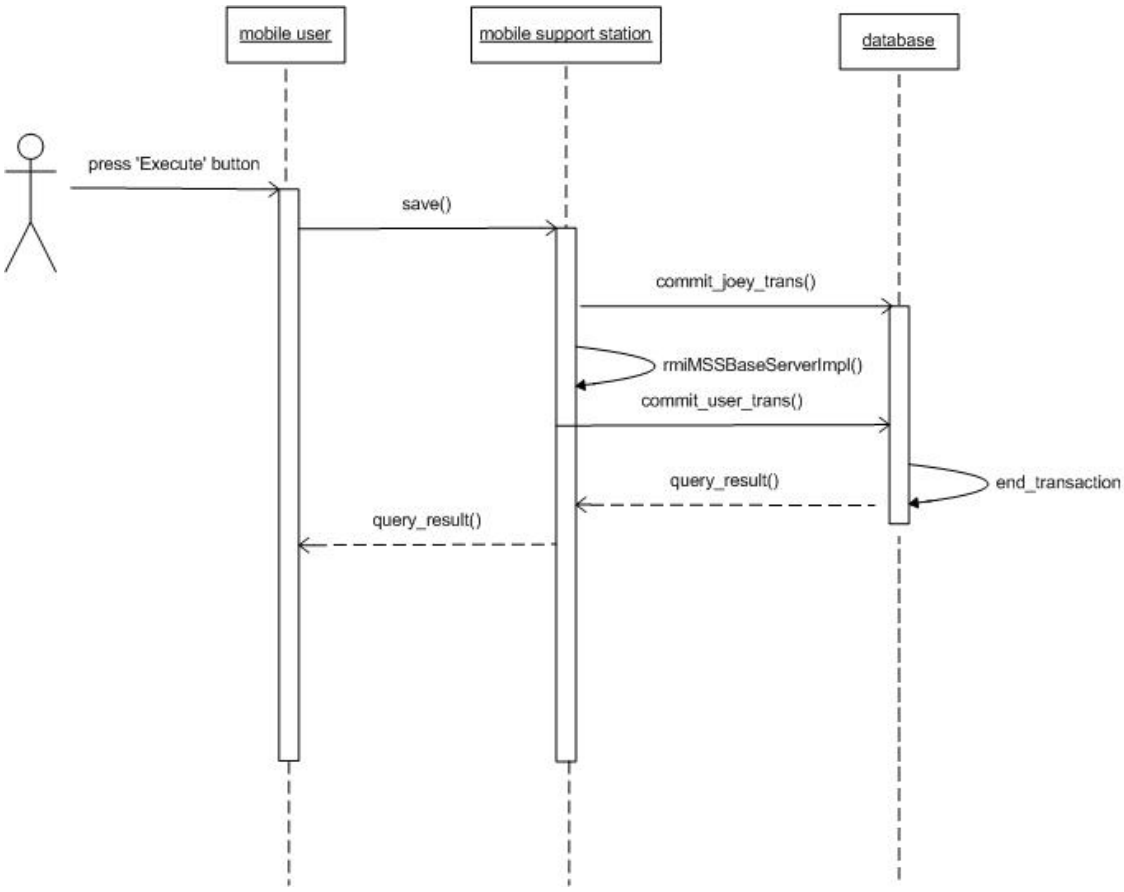
Appendix G: Sequence diagram for ‘INSERT’ keyword



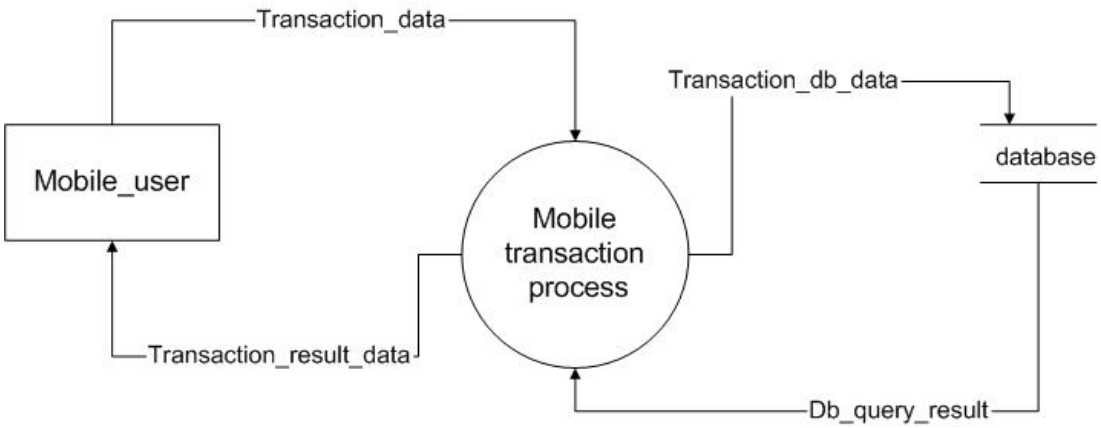
Appendix H: Sequence diagram for ‘UPDATE’ keyword



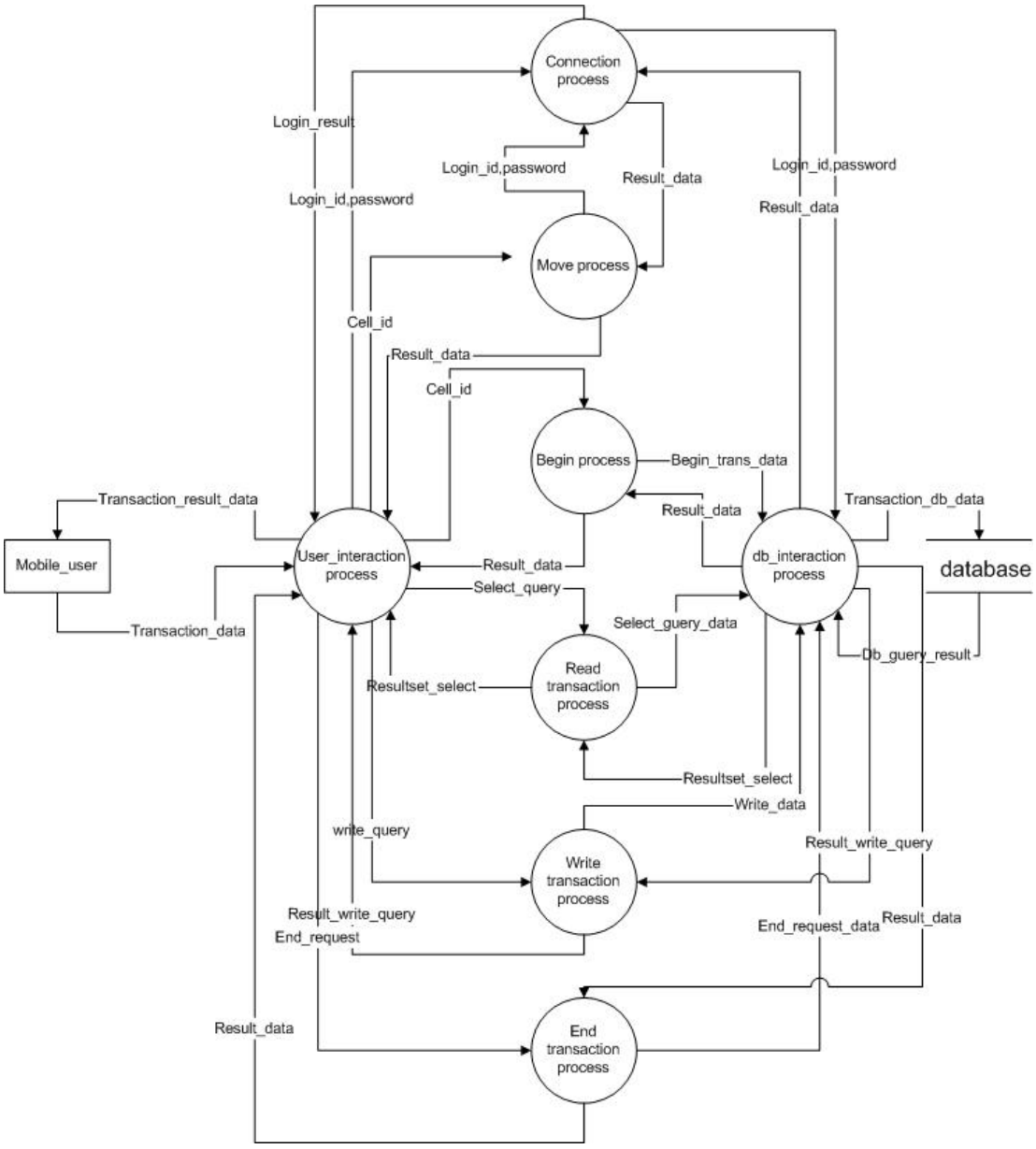
Appendix I: Sequence diagram for ‘SAVE’ keyword



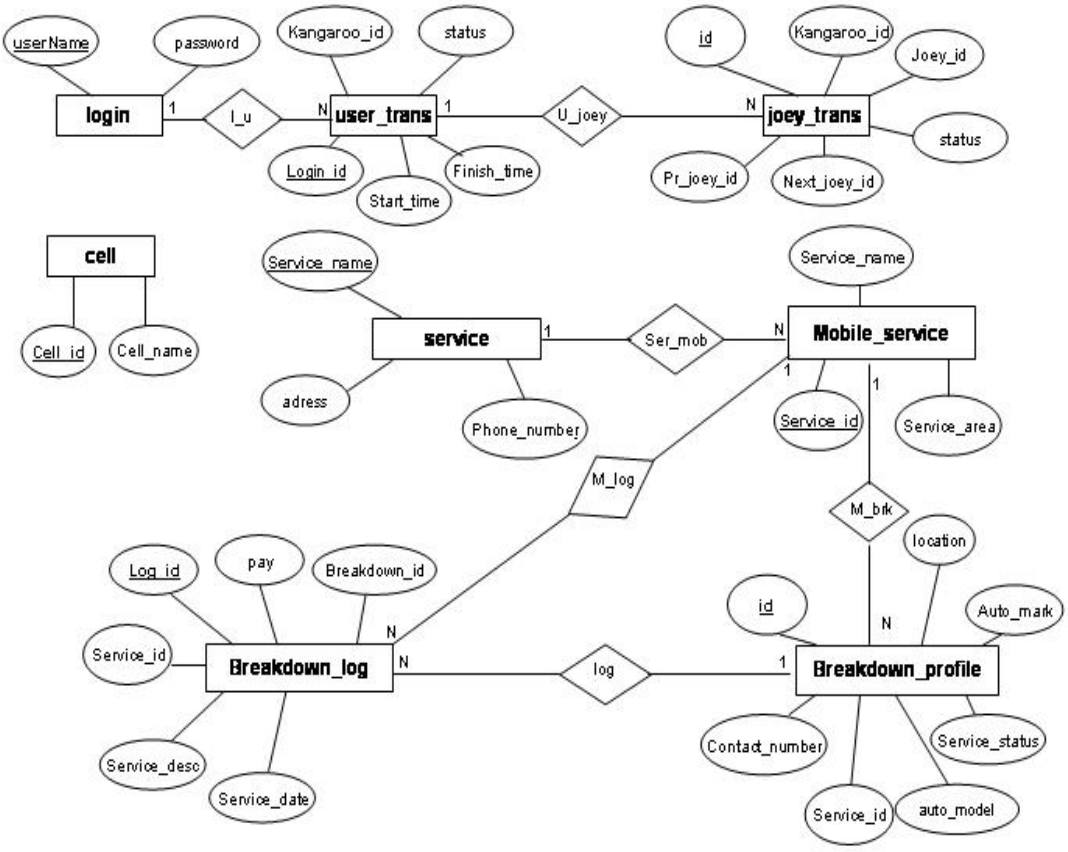
Appendix J: Data Flow Diagram (Level 0)



Appendix K: Data Flow Diagram (Level 1)



Appendix L: Entity-relationship (ER) Diagram of simulation tool's database



Appendix M: Physical Database Scheme Scripts

```
CREATE TABLE login
(
        userName        VARCHAR(6)  PRIMARY KEY,
        password        VARCHAR(6)
);

**********************************************************************

CREATE TABLE    user_trans
(
      login_id              VARCHAR(6)  NOT NULL REFERENCES login(userName),
      kangaroo_id           VARCHAR(15) NOT NULL,
      status                VARCHAR(1),
      start_time            DOUBLE,
      finish_time           DOUBLE,
      PRIMARY  KEY(kangaroo_id)
);

**********************************************************************

CREATE TABLE    joey_trans
(
      id                    INTEGER NOT NULL,
      kangaroo_id           VARCHAR(15) REFERENCES user_trans(kangaroo_id),
      joey_id               VARCHAR(15),
      status                VARCHAR(1),
      pr_joey_id            VARCHAR(15),
      next_joey_id          VARCHAR(15),
      PRIMARY  KEY(id)
);

**********************************************************************

CREATE TABLE service
(
     service_name       VARCHAR(100)  PRIMARY KEY,
     address            VARCHAR(200),
```

```
    phone_number        INTEGER
);

**********************************************************************

CREATE TABLE mobile_service
(
    service_id          VARCHAR(100)  PRIMARY KEY,
    service_name        VARCHAR(100)  REFERENCES service(service_name),
    service_area        VARCHAR(100)
);

**********************************************************************

CREATE table breakdown_profile
(
    id                  VARCHAR(6)   PRIMARY KEY,
    service_id          VARCHAR(100) REFERENCES mobile_service(service_id),
    location            VARCHAR(200),
    auto_mark           VARCHAR(25),
    auto_model          VARCHAR(25),
    contact_number      INTEGER,
    service_status      VARCHAR(1)
);
```