Hien Nam Le

A TRANSACTION PROCESSING SYSTEM FOR SUPPORTING MOBILE COLLABORATIVE WORKS

Thesis for the degree philosophiae doctor

Trondheim, October 2006

Norwegian University of Science and Technology
Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Computer and Information Science

NTNU
Norwegian University of Science and Technology

Thesis for the degree philosophiae doctor

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

ISBN 82-471-8059-6 (printed version)
ISBN 82-471-8058-8 (electronic version)
ISSN 1503-8181

Doctoral theses at NTNU, 2006:147

Printed by NTNU-trykk

To my wife ANH THU and son TAC TRI

Preface

This is a doctoral thesis submitted to the Department of Computer and Information Science (IDI), Norwegian University of Science and Technology (NTNU), in partial fulfillment of the degree of Doktor Ingeniør (PhD). The work has been carried out at the Database Systems Group in the years 2001-2005. The doctoral thesis was done in the context of the MObile Work Across Heterogeneous Systems (MOWAHS) project. The MOWAHS project is sponsored by the Norwegian Research Council's IKT 2010 programme and the Department of Computer and Information Science, NTNU.

Acknowledgements

First and foremost, I would like to thank my supervisor Professor Mads Nygård for the enormous time and effort that he spent with me during my PhD study. This work could not have been done without his excellent guidance, advice, encouragements and inspirations.

I would also like to thank the members of the MOWAHS project: Professor Reidar Conradi, Heri Ramampiaro, Carl-Fredrik Sørensen, and Alf Inge Wang for their valuable comments and helpful discussions on my research. I would further like to thank graduated students Rune Høivik and Gunnar Gauslaa Bergem for their contributions to the implementation of the mobile data sharing system.

I would even like to thank all the colleagues at the Database Systems group for supporting my work and research. I would further like to thank the technical and the administrative staffs at IDI for providing essential assistance.

I would finally like to thank my family, especially my father, for their constant encouragement. And last but not least, I express my deepest thanks to my wife Anh Thu and our son Tac Tri for their great love and inspiration.

July 2006
Hien Nam Le

Abstract

The theme of this research is mobile transaction processing systems, focusing on versatile data sharing mechanisms in volatile mobile environments.

The rapid growth of wireless network technologies and portable computing devices has promoted a new mobile working environment. A mobile environment is different from the traditional distributed environment due to its unique characteristics: the mobility of users or computers, the frequent and unpredictable disconnections of wireless networks, and the resource constraints of mobile computing devices.

On the one hand, the mobile environment promotes a new working model, i.e., people can carry out their work while being on the move. The environment for accessing and processing information is changing rapidly from stationary and location dependent to mobile and location independent. On the other hand, these unique characteristics of the mobile environment pose many challenges to mobile transaction processing systems, especially in terms of long delaying periods, data unavailability and data inconsistency.

Many research proposals that focus on supporting transaction processing in mobile environments have been developed. However, there are still major issues that have not been completely solved. One of the problems is to support the sharing of data among transactions in volatile mobile environments. Our solution is to provide the mobile transaction processing system with flexible and adaptable data sharing mechanisms that can cope with the dynamic changes of the surrounding environmental conditions while ensuring data consistency of the database systems.

The results of our research consist of three important contributions:

- The first contribution is a versatile mobile data sharing mechanism. This is achieved by the concepts of the *mobile affiliation workgroup* model that focuses on supporting mobile collaborative work in the *horizontal dimension.* The mobile affiliation workgroup model allows mobile hosts to form temporary and dynamic mobile workgroups by taking advantage of wireless communication technologies, i.e., the ability of direct communication among nearby mobile hosts. The data sharing processes among transactions at different mobile hosts are carried out by shared transactions, called *export* and *import* transactions. These shared transactions interact through a mobile sharing workspace, called an *export-import repository*. Data

consistency of the database systems is assured by either serialization of transactions or applying user-defined policies. Our mobile data sharing mechanism provides an adaptable way for increasing data availability, while taking into account all the important characteristics of mobile environments, which are: the mobility of computing hosts, the frequent and unpredictable disconnections of wireless networks, and the resource constraints of mobile computing devices. Therefore, it has the ability to increase the throughput of mobile transaction processing systems.

- The second contribution is a data conflict awareness mechanism that supports mobile transactions to be aware of conflicts among database operations in mobile environments. The data conflict awareness mechanism is developed based on the concepts of the *anchor transaction* that plays the role of a proxy transaction for local transactions at a disconnected mobile host. With the support of the data conflict awareness mechanism, the mobile transaction processing system has the capacity to minimize delay of transaction processes and to enforce consistency of the database systems.

- The third contribution is a mobility control mechanism that supports the mobile transaction processing system to efficiently handle the movement of transactions in mobile environments. We distinguish two types of transaction mobility in accordance with: (1) the movement of mobile hosts *through mobile cells*, and (2) the movement of mobile hosts *across mobile affiliation workgroups*. The mobility of transactions through mobile cells is handled by movement of the anchor transaction. While the mobility of transactions across mobile affiliation workgroups is controlled by the dynamic structure of export and import transactions.

We have developed a mobile transaction processing system for MOWAHS. Especially, we have successfully designed, implemented, and tested several important system components such as the mobile locking system and the mobile data sharing system.

Table of Contents

List of Figures

List of Tables

PART I

BACKGROUND and ORIENTATION

Chapter 1
Introduction

The theme of this thesis is transaction processing in mobile and heterogeneous environments. The main focus of this thesis is on developing a mobile transaction processing system that has the ability to support mobile data sharing and to cope with the dynamic changes of mobile environments. This chapter presents the motivation of the research, states the research questions, and remarks the important contribution results. At the end of the chapter, we outline the organization of the thesis to serve as a guide for the reader.

1.1 Motivation

At present, many types of mobile computing devices such as laptops and personal digital assistants (PDA) are available. The computing capacities of these mobile devices become more and more powerful in terms of processing speed, storage capacity and operating time. As a result, these mobile computing devices are becoming the essential work equipments. At the same time, many wireless network technologies are also developed and deployed, for example Bluetooth, wireless USB, wireless LAN or Universal Mobile Telecommunications System (UMTS).

The rapid expansion of both the wireless network technologies and the capacity of mobile computing devices has created a new work environment. With the support of wireless networks and mobile computing devices, people can carry out their work while being on the move. The environment for accessing and processing information is rapidly changing from stationary to mobile and location independent. This new work environment, called the mobile work environment, provides people a flexible and efficient work environment.

1.1.1 An application example

To illustrate the advantages of the mobile work environment, we will present and discuss a mobile IT (*I*nformation *T*echnology) support system. The mobile IT support system is a cooperative work system in which IT officers help users to deal with computer problems such as fixing a hardware problem or upgrading a software application (see Figure 1.1). The objective of the mobile IT support system is to solve as many computer problems as quickly as possible.

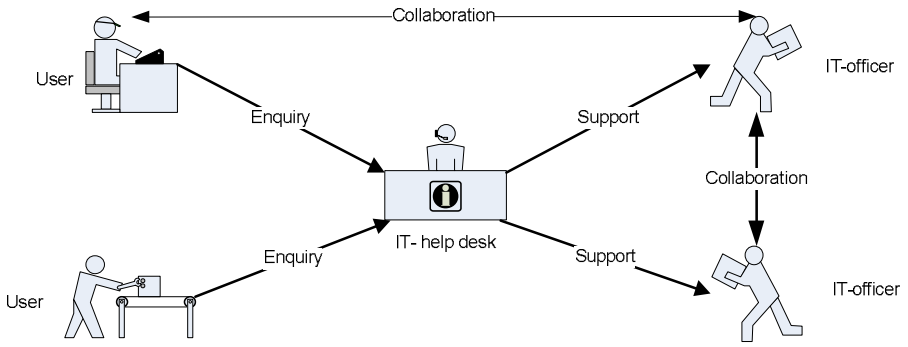

Figure 1.1: The mobile IT support system

Users report computer problems by sending enquiries to the IT help desk. These computer problems will be handled by IT officers. An IT officer has to prepare in advance all the necessary data and tools to solve a computer problem. The IT officer also has to move around to specific locations like offices or computer labs where the computers and equipments are located. While handling a computer problem, an IT officer may need to contact the user, who submitted the inquiry, to clarify the problem. For example, to prepare necessary equipments for a multimedia lecture, the IT officer needs to know what types of computers and applications are used. Furthermore, when working on a difficult problem, an IT officer may also want to consult other colleagues for additional support. For each computer problem, a logbook is written to keep track of its progress. When a computer problem is solved, its logbook will be archived in the IT help desk system for future reference.

Traditionally, all the contact among IT officers and users must be carried out through the IT help desk system. The users must connect the IT help desk system to report computer problems. The IT officers must connect the IT help desk system from stationary and wire-connected computers to cooperate with users or other IT officers, and to update logbooks. There are several disadvantages of this work environment when the IT officer is unable to connect the IT help desk system. First, the IT officer is not able to update logbooks to keep the status of computer problems up-to-date. Therefore, the number of unsolved computer problems in the IT help desk system may incorrectly increase, and it is difficult to manage the progress of these enquiries. Second, if the description of a computer problem, which is handled by the IT officer, is modified, the IT officer will not be aware of it. Consequently, the IT officer is not well prepared to work efficiently on the computer problem.

With the support of mobile computing devices and wireless network technologies, the mobile IT support system can be extended to attack the above disadvantages and improve its performance. First, the logbook of a computer problem, which an IT officer is currently working on, can be first updated in the mobile computer of the IT officer. The logbook will also be saved in the IT help desk system via the wireless networks. This way, the mobile IT support system can effectively manage the user enquiries. Second, the IT officer can contact the IT help desk system to retrieve the up-to-date information of computer problems, and communicate with other colleagues while they are on the move.

Consequently, the IT officer is well prepared to solve the computer problems. Third, the IT help desk system can be informed about the current location of the IT officers. Computer problems, whose locations are nearby the location of an IT officer, can be assigned to this IT officer, i.e., saving the traveling time of the IT officer. As a result, more computer problems can be solved in shorter time.

There are many challenges in mobile work environments. The wireless networks can be disconnected while an IT officer is working on a computer problem. Therefore, the status of this computer problem, which is currently stored in the mobile computer, can be different with the one stored in the IT help desk system. The mobile computers may not have the required capacity to support the IT officer to solve an enquiry. Consequently, some of the work may be delayed or suspended. Moreover, the collaborative activities among the IT officers and users can be carried out directly, i.e., without going through the IT help desk system. This leads to the demand for a new collaborative work model.

1.1.2 Challenges of transactions in mobile environments

Traditionally, database transactions with ACID (Atomicity, Consistency, Isolation, and Durability) properties have been used to enforce the integrity constraints of database systems in centralized or distributed environments [GR93]. However, due to the challenging characteristics of the mobile environments such as the mobility of mobile computers, the frequent disconnections of wireless networks and the limited processing power of mobile computers [PS98, Mad+02], the traditional database transactions may not be able to efficiently support transactions in volatile mobile environments.

There are many new transaction models [SRA04, Hir+01, Bar99] that have been developed to support transactions in mobile environments. One common approach is to provide for the transaction processing systems adaptability [Rak98] to deal with different environment conditions and to cope with the constraints of mobile computing resources. However, there are still several major limitations. For example, the architecture of mobile transaction environments [Mad+02] relies too much on the mobile support stations; a few research works focus on mobile transactions that are distributed among mobile hosts [SRA04]. The ability to support both the disconnection and mobility is still a major challenge for mobile transaction models [Hir+01]. In this thesis, we focus our research on two main issues – that are: (1) improving the data availability in mobile environments, and (2) supporting the mobility of transactions in mobile environments.

1.1.3 The MOWAHS project

This thesis is carried out as a part of the MObile Work Across Heterogeneous Systems (MOWAHS) project. The MOWAHS project is sponsored by the Norwegian Research Council's IKT 2010 programme. The project is jointly carried out by the Software Engineering and Database Technology research groups, at the Department of Computer and Information Science, Norwegian University of Science and Technology.

The two main goals of the MOWAHS project are [Con+01]:

- **G1.** Helping to understand and to continuously assess and improve work processes in virtual organizations.
- **G2**. Providing a flexible, common mobile work environment to execute and share real work processes and their artifacts.

The main objective of this thesis is to achieve the second goal of the MOWAHS project. The theme of the thesis is transaction processing in mobile and heterogeneous environments. We must deal with a variety and heterogeneity of electronic devices, equipments (e.g., laptops, PDAs, mobile phones) and database models. In addition, the mobility of mobile devices and the lack of connectivity of these mobile devices must also be taken into account.

1.2 Research questions

The rationale of this thesis is:

> *To be able to support a transaction processing system to efficiently deal with different surrounding conditions that are contextualized by the characteristics of the mobile environments.*

The main research question of this thesis is:

> ***How can we furnish a transaction processing system so that it can cope with the constraints of mobile resources and the variations of operating conditions in mobile environments?***

In order to be able to answer the research question, we define a set of refinement questions that direct the development of this work:

Q1: Current situation.

- What are the current ideas and concepts that have been developed to answer the main research question or to address part of it?

Q2: Characteristics and requirements of mobile transactions.

- What are the challenging characteristics of transactions in mobile environments?
- What are the requirements of a mobile transaction processing system that accomplishes the main research question?

Q3: Approach and solutions.

- What are the concepts and foundations for developing the required mobile transaction processing system?
- How should we design and implement the required mobile transaction processing system?

Q4: Evaluation.

- How well do the research results fulfill the requirements of the mobile transaction processing system?
- How do the research results compare with previous related works?

1.3 Research approach

The previous section presents the rationale and research questions for this thesis. Now, we need to identify a research methodology, and make research plans so that our research activities are effectively organized and coordinated.

1.3.1 Research methodology

Research methodology determines the system and the different stages in which the research is carried out [BCW95]. [PP00] categorized research into three types: *exploratory, testing out* and *problem solving*. Exploratory research focuses on handling new problems by either developing new concepts or conducting empirical studies. Testing out research uses the limitations of previous research as the starting point, and develops new theories to solve the problem. Problem solving research is finding a new methodology to solve a defined problem. Our research approach in this thesis is identified as the testing out research.

1.3.2 Research plans of this thesis

First, we approached the problem by studying new challenges that are the results of the changes of the transaction processing environments from centralized, via distributed to mobile environments. Then, we surveyed and analyzed existing transaction models and transaction processing systems that have been developed to attack these challenges. We addressed in detail the limitations of these reviewed transaction models. The first part of the thesis, which includes Chapters 1 (this chapter), 2, 3 and 4, is the results of this research phase.

Second, we proposed the concepts of mobile affiliation workgroups that focus on supporting data sharing among transactions at mobile hosts in a volatile mobile environment. Using this model as a starting point, we began developing a data sharing mechanism for mobile transactions and then formalized our mobile transaction processing system. The results of this research phase are presented in Chapters 5 and 6 of the second part of this thesis.

Next, we started designing and implementing the MOWAHS mobile transaction architecture that plays a role as a proof of concept of our theoretical research. We have selected and implemented two important system components of the MOWAHS mobile transaction architecture - that are: (1) the mobile locking system to deal with the disconnections of mobile hosts, and (2) the mobile data sharing mechanism to support sharing of data among mobile transactions. These practical works are addressed in Chapter 7 in the third part of this thesis.

Finally, the evaluation of our research results is presented in Chapter 8. Our research results are assessed based on: (1) the applicability of the mobile transaction processing system in mobile environments; (2) the consolidated advantages with other related works; and (3) the accomplishment of the main research question (see Section 1.2). Chapter 9 concludes our main research achievements and suggests several topics for the future work.

1.4 Research environments

The work conducted in this thesis is entirely carried out at the Department of Computer and Information Science, Norwegian University of Science and Technology. This thesis is part of the MOWAHS project that is jointly funded by the Norwegian Research Council's IKT 2010 programme for the first three years (2001 to 2004) and by the Department of Computer and Information Science for the forth year (2004 to 2005).

1.5 Requirements

In order to evaluate the research results (presented in Chapter 8), we have initiated a list of requirements for our mobile transaction processing system. These requirements will be further discussed in more detail in Section 3.5 of this thesis. Here, we briefly identify and describe nine requirements that a mobile transaction processing system must have in its capacity. These nine requirements are categorized into four groups: the mobility of transactions, the wireless networks and limited mobile resources, the customization of transaction properties, and the recovery of transactions.

The mobility of transactions

R1. *The mobile transaction processing system must be able to effectively handle the hand-over control of transactions*. Mobility is one of the main qualities of mobile transactions, and can be described in terms of hand-over processes [DHB97]. Therefore, the mobile transaction processing system must be able to capture and control these hand-over processes.

R2. *The mobile transaction processing system must support interactions among transactions at different mobile hosts.* Ad-hoc communication and collaborative activities can happen when mobile hosts are on the move. Therefore, the peer-to-peer interactive support is essential, especially for the sharing of data among transactions at different mobile hosts.

The wireless networks and limited mobile resources

R3. *The mobile transaction processing system must support disconnected transaction processing*. Due to long disconnection periods in communication between mobile hosts and database systems, the mobile transaction processing system must be able to support transaction processing in disconnected environments.

R4. *The mobile transaction processing system must support distributed transaction execution among mobile hosts and stationary hosts*. Due to the limitation of computing resources of mobile devices, the mobile transaction processing system must be able to move the execution of transactions from one mobile host to other non-mobile or mobile hosts.

The customization of transaction properties

R5. *The mobile transaction processing system must have the ability to customize the atomicity property of transactions.* The standard atomicity property of transactions may be too strict in mobile environments, especially for long-lived transactions. Therefore, the mobile transaction processing system must provide mechanisms to customize the atomicity property of transactions.

R6. *The mobile transaction processing system must support sharing partial states and status among transactions*. Here, we also customize the isolation property of transactions. This is to avoid long blocking periods among on-going mobile transactions, especially when the mobile hosts are disconnected from the database servers.

R7. *The mobile transaction processing system must assure the durability property of transactions*. In mobile environments, transactions are disconnectedly executed and locally committed at the mobile hosts, and globally committed at the database servers. Thus, the mobile transaction processing system must provide different methods to safely archive information in accordance with the commits of transactions.

The recovery of transactions

R8. *The mobile transaction processing system must provide efficient recovery strategies.* In mobile environments, the execution of transactions can be disrupted due to many factors, for example the disconnections of wireless networks or the exhaustion of battery energy. The transaction processing system must support different recovery methods to deal with the disruptions.

R9. *The mobile transaction processing system must support temporary data and transaction management.* The execution processes of transactions are performed at different computing (mobile or non-mobile) hosts that can be asynchronously connected or disconnected. Therefore, the non-permanence of data and transactions behavior must be managed. The temporary management must also take care of conflicting operations among transactions at different mobile hosts.

1.6 Publications

The research results of this thesis have already been published at several conferences. The published papers are presented in the order of importance.

1. Hien Nam Le and Mads Nygård: *Mobile Transaction System for Supporting Mobile Work*, **International Workshop on Database and Expert Systems Applications (DEXA)**, IEEE Computer Society, 2005, pp 1090-1094.

   This paper presents the export and import transaction model that supports peer-to-peer sharing of data among transactions at different mobile hosts. This paper contributes to Chapter 5 and 6 of this thesis.

2. Hien Nam Le and Mads Nygård: *A Mobile Affiliation Model for Supporting Mobile Collaborative Work*, **Ubiquitous Mobile Information and Collaboration Systems (UMICS), CAiSE Workshop**, FEUP Edições, 2005, pp 649-659.

   This paper presents a mobile affiliation workgroup model to support mobile collaborative work among mobile users. The paper discusses the concepts of vertical and horizontal collaboration among mobile users. This paper contributes to Chapter 5 of this thesis.

3. Hien Nam Le, Mads Nygård and Heri Ramampiaro: *A Locking Model for Mobile Databases in Mobile Environments*, **International Conference on Database and Applications (DBA)**, ACTA press, 2004, pp 49-55.

   This paper discusses a locking model for mobile databases, which is a part of the mobile transaction processing system, to deal with disconnections and long locking periods. The mobile locking model supports cooperative operations and conflict awareness in mobile working environments. The paper presents the design and implementation of prototypes. This paper contributes to Chapter 7 of this thesis.

4. Carl-Fredrik Sørensen, Alf Inge Wang, Hien Nam Le, Heri Ramampiaro, Mads Nygård, and Reidar Conradi: *Using the MOWAHS Characterisation Framework for Development of Mobile Work Applications*, **International Conference on Product Focused Software Process Improvement (PROFES)**, Lecture Notes in Computer Science 3547, 2005, pp 128-142.

   This paper describes an evaluation of the MOWAHS characterisation framework to analyse mobile work scenarios in order to make corresponding mobile software systems. This paper partly contributes to Chapter 5 of this thesis.

5. Heri Ramampiaro, Alf Inge Wang, Carl-Fredrik Sørensen, Hien Nam Le, Mads Nygård: *Requirement Indicators for Mobile Work: The MOWAHS Approach*, **IASTED International Multi-Conference on Applied Informatics (AI)**, ACTA Press, 2003, pp 1153-1160.

   This paper describes the requirement indicators derived from the MOWAHS mobile work characterization framework (MWCF). The requirement indicators are used to reveal the complexity of the different parts of a mobile support system (software and hardware). Further, these indicators can be a help to prioritize the non-functional and

functional requirements of the mobile system. This paper partly contributes to Chapter 5 of this thesis.

6. Carl-Fredrik Sørensen, Alf Inge Wang, Hien Nam Le, Heri Ramampiaro, Mads Nygård, and Reidar Conradi: *The MOWAHS Characterisation Framework for Mobile Wor*k, **IASTED International Multi-Conference on Applied Informatics (AI)**, ACTA press, 2002, pp 258-264.

   This paper describes a framework used to characterize mobile work scenarios in order to elicit functional and non-functional requirements for a mobile process support system. The framework is a tool for specifying and analyzing mobile scenarios in detail, resulting in a characterization of the mobile work scenarios. This paper partly contributes to Chapter 5 of this thesis.

1.7 Research contributions

The main contributions of the thesis are summarized as follows:

- *Providing fundamental concepts that extend and support mobile collaborative workgroup models for mobile users, called horizontal collaboration.*

  To our knowledge, there is no similar concept to the horizontal collaboration that supports collaborative work in mobile ad-hoc environments. The horizontal collaboration supports mobile users (that are currently being disconnected from the database servers) to dynamically form temporary mobile workgroups, called *mobile affiliation workgroups,* so that they can continue to carry out their collaborative operations. The concept of the mobile affiliation workgroup is presented in Chapter 5.

- *Providing concepts and models to support data sharing among mobile transactions in mobile environments, without any support from the database systems.*

  Mobile data sharing operations among transactions at different mobile hosts are carried out by the means of *export* and *import* transactions through a mobile sharing workspace, called *export-import repository*, that belongs to a mobile affiliation workgroup. These concepts and formalization of mobile data sharing are presented in Chapters 5 and 6 of this thesis, respectively.

- *Supporting conflict awareness among mobile transactions in mobile environments.*

  Conflict awareness among mobile transactions in mobile environments is supported by the concept of an *anchor transaction*. The anchor transaction plays the role of a proxy transaction for local transactions that are disconnectedly processed at disconnected mobile hosts. The anchor transaction also keeps track of conflicting database operations among mobile transactions in both the *data hoarding* and *transaction integration* stages. The concept of the anchor transaction is discussed in

Chapter 5, and the conflict awareness mechanism is presented in Chapter 6 of this thesis.

- *Supporting mobility of transactions in mobile environments.*

  The mobility of transactions in mobile environments is categorized into two types in accordance with the movement of mobile hosts: (1) mobility across mobile cells, and (2) mobility across mobile affiliation workgroups. The mobility of mobile transactions across mobile cells is captured by the movement of the anchor transactions; while the mobility of mobile transactions across mobile affiliation workgroups is taken care of via the dynamic structure of the export and import transactions. The mobility of transactions is addressed in Chapters 5 and 6 of the thesis.

- *Providing a new multiple-abort-dependency rule for mobile transactions in mobile environments.*

  The multiple-abort-dependency rule presents a flexible way to describe the dependencies among transactions in mobile environments. This rule is addressed in Chapter 6.

- *Designing and implementing a mobile transaction processing system prototype that supports mobile collaborative work.*

  We have chosen to design and implement two important system components of our mobile transaction processing system: (1) the mobile locking system, and (2) the mobile data sharing system. The mobile locking system supports mobile transactions to cope with disconnections and long locking periods. The mobile data sharing system supports data sharing among transactions at different disconnected mobile hosts. These designs and implementations of these two system components are presented in Chapter 7.

1.8 Organization of the thesis

This thesis consists of nine chapters that are divided into three parts, outlined as follows:

Part 1. Setting of the thesis, providing background concepts of transaction processing, and surveying the state-of-the-art of mobile transaction models and processing systems.

- *Chapter 1* (this chapter) contains the introduction of the thesis. The chapter outlines the goals and the achievements of this research.

- *Chapter 2* reviews the basic transaction concepts and the architecture of transaction processing systems.

- *Chapter 3* discusses in detail the characteristics of mobile environments and the impacts of these characteristics on mobile transactions. The characteristics of transactions in mobile environments and the requirements of the mobile transaction processing system are investigated and addressed in detail.

- *Chapter 4* is the literature review chapter. The chapter surveys existing traditional and mobile transaction models and transaction processing systems that are related to the theme of this thesis. The limitations of the related research also have been addressed.

Part 2. Discussing the concepts of horizontal collaboration, introducing new concepts and models for mobile transaction processing systems.

- *Chapter 5* presents the fundamental concepts of our mobile transaction processing system that includes the mobile affiliation workgroup, the export-import repository, the export and import transactions, and the anchor transaction. The mobile data sharing models are also presented in this chapter.

- *Chapter 6* formalizes the theoretical proposals of our mobile transaction processing system.

Part 3. Designing and implementing the MOWAHS mobile transaction processing system, and evaluating the research results.

- *Chapter 7* discusses the design and the current stage of the implementation of the MOWAHS mobile transaction processing system.

- *Chapter 8* evaluates the research results. This chapter discusses how the requirements of the mobile transaction processing system are achieved, and answers the main research question.

- *Chapter 9* concludes the main achievements of our research, and discusses topics for future works.

Further, the notations used in this thesis are listed and explained in a separate entry after the references entry.

Chapter 2
Transaction Processing

In this chapter, we first revisit the basic concepts of database transactions, and discuss how these concepts are achieved in practical systems. Next, we briefly go through the architecture of transaction processing systems in the centralized and the distributed environments.

2.1 Database and transaction concepts

A *database* is a collection of data items that is gathered over a period of time, and safety stored for further examination or analysis [GUW01]. A database is usually accompanied by a data structure and a set of constraint rules that specify what information a data item represents. For example, in an employee database, the employee age is an integer number and must be greater than eighteen and less than sixty five. *A database state* is a collection of all the stored data values of all the data items in the database at a specific time [Elm+92]. A *consistent state* of a database is a database state in which all the data values fulfill all the constraint rules of the database. A set of operations is usually provided to support users in retrieving or modifying data items in the database. These provided operations can be simple, for example read and write operations, or more complex operations, for example deletion or modification operations. To assist users to perform much more complex operations rather than reading from and writing to the database, a piece of specialized software called a *database management system* (DBMS) is accommodated to the database. In general, a DBMS not only provides an easy-to-use and friendly interface to users for accessing and manipulating the database, but also manages all the database operations. In addition, the DBMS also protects the database from unauthorized users.

2.1.1 Database transactions

Users can interact with the database by one or many database operations. The database operations can be gathered together to form a unit of execution program that is called a *transaction* [GR93]. In other words, a transaction is a logical execution unit of database operations. A transaction transforms the database from one consistent state to another consistent state. Figure 2.1 presents a programming model of a transaction.

```
Begin_transaction (initial_consistent_state)
        One or more database operations

if (reach new_consistent_state) then
Commit_transaction (new_consistent_state)
else
Abort_transaction (initial_consistent_state)
```

Figure 2.1: Transactional programming model

A transaction program starts from an initial consistent state of the database by invoking a *Begin_transaction* method call. After that, one or a set of database operations of the transaction program are executed. When these database operations are completed, i.e., a new consistent database state is established as designed, the transaction program saves this new consistent state into the database by calling the *Commit_transaction* method. The *Commit_transaction* call ensures that all the database operations of the transaction program are successfully executed and the results of the transaction are safely saved in the database. If there is any error during the execution of the transaction program, the initial consistent state of the database is re-established by the *Abort_transaction* call. The *Abort_transaction* call indicates that the execution of the transaction program has failed and this execution does not have any effect on the initial consistent state of the database. The transaction is said to be *committed* if it has successfully executed the *Commit_transaction* call, otherwise it is *aborted*. A transaction is called a *read-only transaction* if all of its database operations do not alter any database state.

2.1.2 The ACID properties

In a database system, there may be a large number of transactions that are executed concurrently, i.e., the shared data items in the database are read and possible written by many transactions at the same time. Each transaction must ensure that it always preserves the consistency of the database system. In order to retain and to protect the consistency of the database system, transactions will have the following ACID (Atomicity, Consistency, Isolation, and Durability) properties [GR93]:

- **Atomicity.** Either all database operations of a transaction program are successfully and completely executed, or none of the database operation of this transaction program is executed.

- **Consistency.** A transaction must always preserve and protect the consistency of the database, i.e., it transforms the database from one consistent state to another. In other words, the result of a transaction that has committed fulfills the constraints of the database system.

- **Isolation.** An on-going transaction must not interfere with other concurrent transactions, or be able to view intermediate results of other concurrent transactions.

In other words, a transaction is executed as if it is the only existing execution program on the database system at any given time.

- **Durability.** The result of a transaction that has successfully committed is permanent in the database. The consistent state of the database is always survived despite any type of failures.

The ACID properties of a transaction ensure that: (1) a transaction always keep the database in a consistent state, (2) a transaction does not disturb other transactions during their concurrent execution processes, and (3) the consistent state of the database system that is established by a committed transaction withstands software or hardware failures. In order to achieve the ACID properties, normally, two different sets of protocols named *concurrency control protocols* and *recovery protocols* are needed [Elm+92].

2.1.3 Concurrency control of transactions

In this section, we discuss the problems that can occur in a database system in which there are many transactions being executed concurrently. In other words, we answer the question of why there is a need of concurrency control in the database system. We also review different techniques that ensure the correctness of transaction execution.

To illustrate and to simplify the analyses without losing generality, we assume that each transaction possesses the following characteristics:

- Transaction T_i starts by a *Begin_transaction* call that is denoted by B_i.
- A database operation $Op_i(X)$ on a data item X is either a read operation $R_i(X)$ or a write operation $W_i(X)$. In general, more complex operations on a database system can be modeled via read and write operations.
- Transaction T_i ends by either a *Commit_transaction* call denoted by C_i, or an *Abort_transaction* call denoted by A_i.

Some typical problems which are caused by the concurrent execution of transactions are: lost update, dirty read, and unrepeatable read [GR93]. These problems are presented in Figure 2.2.

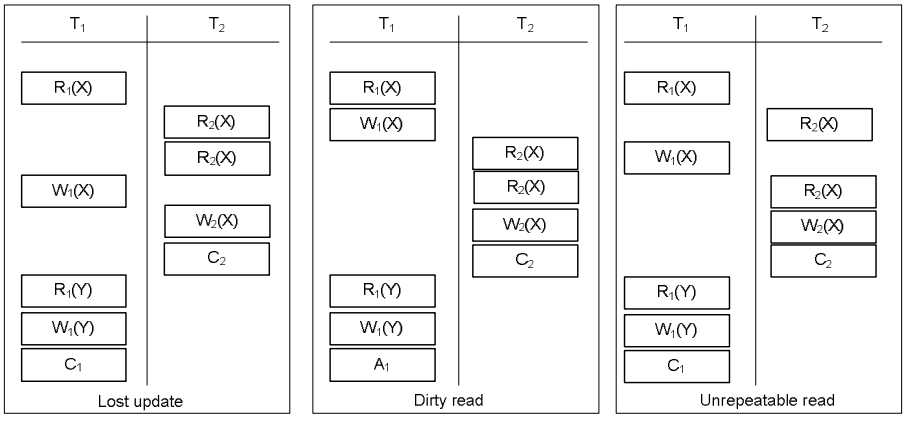

Figure 2.2: Concurrency problems

First, the lost update occurs when two transactions T_1 and T_2 try to write the same data item X. In the figure, transaction T_2 overwrites the value of data item X that was prior written by transaction T_1. The dirty read occurs when transaction T_2 reads the value of data item X that is written by transaction T_1 before the transaction T_1 commits. If the transaction T_1 aborts, the transaction T_2 has been operating on an invalid data value. Finally, the unrepeatable read happens if a transaction executes the same read operation at different times, and obtains different data values. In Figure 2.2, the read operations of transaction T_2 return two different values of X: before and after the write operation of transaction T_1.

The concurrency problems can be solved if the DBMS can schedule these database operations of transactions in an execution order in which no transaction interferes with other, i.e., fulfills the isolation property of transactions. The execution order that sequentially contains all the database operations of all concurrent transactions is called the *schedule* or *history* of transactions [BHG87]. The order of database operations of one transaction must be retained in the schedule of all transactions. A schedule is a serial schedule if, for any pair of transactions, all the database operations of one transaction follow all the database operations of another transaction. In other words, the isolation property of transactions is ensured in a serial schedule. Figure 2.3 (we omit the commitment and the abortion operations of transactions in the schedule) presents the possible serial schedules of transactions T_1 and T_2.

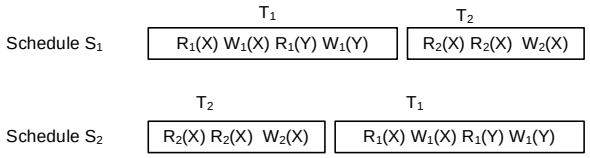

Figure 2.3: Serial schedules

The main disadvantage of the serial schedule is that transactions must be executed serially, i.e., the concurrent execution of transactions does not exist in a serial schedule. This may decrease the performance of the database system. To deal with this drawback, the concept of *serializable schedule* [BHG87] is normally used. A schedule is serializable if it is equivalent to a serial schedule. The remaining question is how to determine if a schedule is a serializable schedule. In other words, we need to clarify the "equivalent" term. Two examples of the equivalent serializability are: *conflict serializability* and *view serializability* [GUW01].

Conflict serializability

The conflict serializability is based on the concepts of conflicting operations. The idea behind the conflicting operations is that: for two sequentially executed operations Op_1 and Op_2 that belong to two transactions T_1 and T_2, respectively, if their order is interchanged, i.e., Op_2 Op_1, the results of at least one of the involved transactions will possibly be changed. In other words, two database operations that belong to two different transactions are conflicted if they access the same data item in the database and at least

one of them is a write operation [GUW01]. Two consecutive operations, which are not in conflict, can be swapped or interchanged in a schedule without any effect on the transaction behavior. Two schedules are said to be *conflict equivalent* if one can be turned into another by swapping the pairs of non-conflict operations [GUW01]. A schedule is *conflict serializable* if it is conflict equivalent to a serial schedule. Figure 2.4 illustrates some conflict serializable (CS) schedules. Both the schedules CS_1 and CS_2 (in Figure 2.4) are conflict serializable with the serial schedule S_1 (in Figure 2.3), while the schedule non-CS_3 is not conflict serializable. Moreover, the schedule CS_1 can be turned into the schedule CS_2 by sequentially swapping pairs of non-conflict operations ($W_2(X)$,$R_1(Y)$), ($W_2(X)$,$W_1(Y)$), and ($R_2(X)$,$R_1(Y)$).

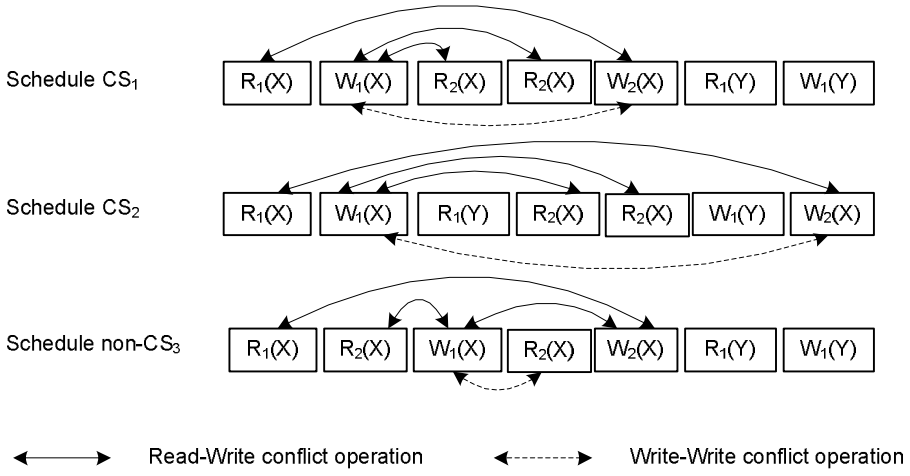

Figure 2.4: Conflict serializable and non-conflict serializable schedules

Verify conflict serializable

A schedule *S* can be validated if it is conflict serializable by analyzing a *serialization graph* [BHG87]. A serialization graph (SG) is a directed graph that is constructed in two steps as follows:

1. Each node labeled T_i in the SG represents an equivalent transaction T_i in the schedule *S*.
2. For any pair of operations, Op_i and Op_j, that are conflict in the schedule *S*, and Op_i precedes Op_j, add an edge from T_i to T_j in the SG.

The schedule *S* is conflict serializable if the constructed SG has no cycles [BHG87]. In Figure 2.5, the serialization graphs of schedules CS_1, CS_2 and *non-*CS_3 (in the Figure 2.4) are constructed. For schedules CS_1 and CS_2, the corresponding SG do not contain any cycle, i.e., the schedules are conflict serializable. On the other hand, the SG of the schedule *non-*CS_3 does contain a cycle $T_1 \rightarrow T_2 \rightarrow T_1$, i.e., it is not conflict serializable.

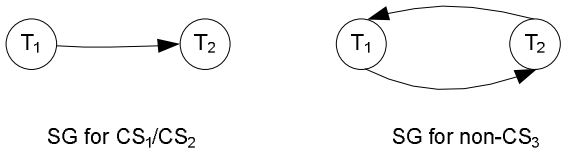

Figure 2.5: Serialization graph

View serializability

View serializablity is a weaker condition that guarantees that a schedule is serializable. Two schedules S_1 and S_2 are said to be view equivalent if the following conditions hold: (1) any read operation in either schedule returns the same data value, and (2) if a write operation $W_i(X)$ is the last operation on data item X in S_1, $W_i(X)$ must also be the last operation on X in S_2 [GUW01]. Thus, the view equivalent conditions ensure that (1) all the transactions read the same data values, and (2) the final database states are identical. If a schedule is view equivalent to a serial schedule, it is said to be *view serializable*.

Figure 2.6 illustrates a view serializable schedule. The serial schedule S_1 presents the sequential order schedule of transactions T_1, T_2, and T_3. The schedule VS_2 is not a conflict serializable schedule because of conflict operation pairs $((W_1(X), R_2(X))$ and $((W_2(Y), W_1(Y))$. However, the schedule VS_2 is a view serializable schedule because: (1) all the read operations $R_1(Y)$, $R_2(X)$ and $R_3(X)$ return the same data values of data items Y and X as in the serial schedule S_1; and (2) all the write operations $W_1(X)$ and $W_3(Y)$ are the last write operations on the data items X and Y as in the serial schedule S_1. The main disadvantage of view serializability is that, verifying view serializable schedule problem has been shown to be a NP-complete problem, i.e., it is not likely that a polynomial time algorithm for this problem will be found [EN00].

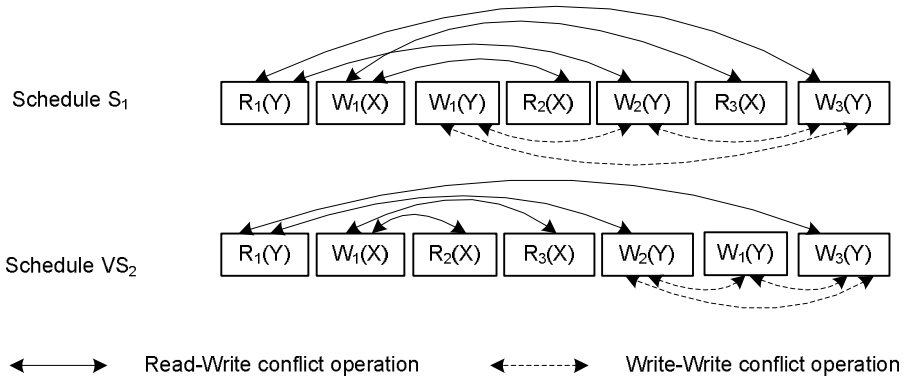

Figure 2.6: View serializable schedule

Concurrency control protocols

To assure that a schedule S is serial equivalent, the database system must keep track of conflict operations in the schedule S, constructs the SG of the schedule S, and checks for a cycle in the constructed SG. This process repeats every time when a new database

operation arrives to the database system, and requires a lot of computing resources and processing time. Due to the overhead of checking serialization graphs, one normally requires that a completion of the execution schedule of all committed transactions is available before the verifying algorithm can be carried out. This is not true in real-world transaction processing systems where transactions are dynamically and continuously submitted to the transaction processing system. Concurrency control protocols, in fact, do not check for serializability, but are used to ensure that a sequence of executable database operations submitted from on-going transactions can form a serializable schedule.

There are two main approaches for concurrency control protocols [GUW01]: *pessimistic* (also called *guard-before*) and *optimistic* (also called *guard-after*). For the pessimistic approach, a database operation is checked if it could cause a non-serializable schedule before it is executed. The database operation is rejected, i.e., the transaction is aborted, if it may potentially lead a schedule into a non-serializable schedule. For the optimistic approach, the submitted database operation is immediately executed as if there is no conflict between this database operation and database operations of other transactions. When a transaction begins to commit, a certification process, in which the transaction will be validated against other transactions, is carried out. If none of the database operations of this transaction breaks the serializability, the transaction is allowed to commit, otherwise the transaction is aborted.

Locking and timestamp ordering protocols are two common concurrency control protocols that are mostly used in the pessimistic approach. Concurrency control by the locking protocol requires that a transaction must request an appropriate lock on a data item before its database operation can be accepted for executing. In other words, a lock plays a role as an execution license for the database operation. One usually applies two types of lock: *shared* (read) and *exclusive* (write) [GR93]. A shared lock can be granted to many transactions at the same time, while an exclusive lock can only be assigned to one transaction at a time (see Table 2.1 for the lock compatibility matrix which shows what kind of lock combination are allowed or not). Serializability among transactions can be guaranteed by a *2-phase locking* (2PL) protocol [BHG87]. The 2PL protocol requires that a transaction must obtain all its locks (in *growing phase*) before it can release any lock (in *shrinking phase*). Strict 2PL is a locking protocol that only allows a transaction to release exclusive locks after it has committed or aborted.

Concurrency control by using timestamp ordering guarantees serializability among transactions based on the following time quantities: (1) the starting time or timestamp of each transaction *TS*, and (2) the read and write timestamp values for each data item *X*, denoted by *Read_TS(X)* and *Write_TS(X)* respectively. These read or write timestamp values correspond to the timestamp value of the latest transaction that successfully reads or writes the data item *X*. A timestamp can be a computer system clock or any logical counter maintained by the database system. When a transaction submits a database operation on a data item *X*, the timestamp *TS* of the transaction will be checked against the current read *Read_TS(X)* and write *Write_TS(X)* timestamp values of the data item. The outcome of this timestamp checking procedure is either the database system accepts

the submitted database operation and the new timestamp value is updated for *X*, or the transaction is aborted.

Table 2.1: Lock compatibility matrix

		Lock Hold	
		Shared	Exclusive
Lock Request	Shared	No conflict	Conflict
	Exclusive	Conflict	Conflict

The optimistic approach for concurrency control was first proposed in [KR81]. There are several methods to carry out the certification process of a transaction, for example the serialization graph testing (SGT) [BHG87] or the validation [Har84]. The SGT method dynamically builds a serialization graph SG between transactions when a conflicting operation is carried out. When a transaction T_i requests to commit, the SGT method checks if the transaction T_i belongs to a cycle of the SG. If it does, the transaction T_i is aborted; otherwise the transaction T_i passes the certification procedure and will be allowed to commit. The validation method is based on the concepts of conflicting operations to ensure that the scheduling of a transaction T_i is serializable in relation to all other overlapping transactions T_j, which have not committed when the transaction T_i begins [CDK00]. Figure 2.7 illustrates a validation process of transaction T_3 (time proceeds from left to right). When transaction T_3 requests to commit, the validation process will check to ensure that the database operations of transaction T_3 do not conflict with the database operations of transactions T_1, T_2 and T_4.

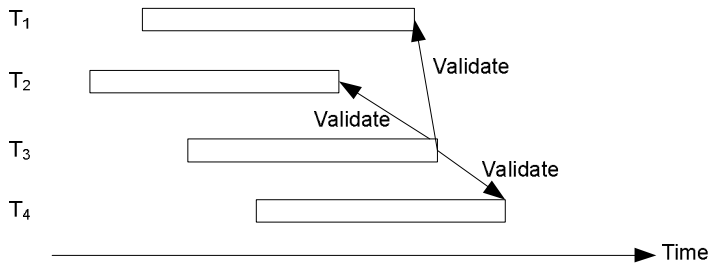

Figure 2.7: The validation procedure of a transaction

Every concurrency control protocol has disadvantages. Transactions in a database system that uses locking protocols can suffer from deadlocks or long blocking periods [GUW01]. Timestamp ordering protocols could have decreased the performance of the transaction processing system if there is a high conflict among transactions [Zha+99], i.e., many transactions must abort or roll back. For guard-after approach, works that have been done and system resources might be wasted if transactions are aborted. Concurrency control in a database system can apply either one or a combination of these concurrency control protocols.

2.1.4 Recovery concepts

The objective of recovery protocols is to enforce the atomicity and durability properties of transactions [Elm+92]. The atomicity property requires that either all or none of the database operations of a transaction is carried out. The durability property refers that the results of committed transactions, i.e., consistent database states, survive any kind of failure. In this section, we first study different types of failures that could happen in a database system. Later, we review different recovery techniques that allow the database system to recover from failures.

Type of failures

Normally, databases are stored on non-volatile media systems like magnetic or optical disks, and are further backed-up by one or more safe storage systems [EN00]. During the execution of transactions, data items are loaded and temporarily stored in computer memory that is volatile storage.

There are two main types of failures of a database system: *catastrophic* and *non-catastrophic* [GUW01]. A catastrophic failure happens when there is a breakdown in data storage systems, for example a hard disk crashes. A catastrophic failure can be recovered if there is a sufficient database system backup. Non-catastrophic failures do not affect the non-volatile database storage system, i.e., only data in the volatile storage such as memory is lost. The non-catastrophic failures include transaction and computer system malfunctions. Failures of transactions might be caused by logical faults of data or transaction programs or by the database system. Computer system malfunctions could be caused by errors in the operating systems or applications. A recovery support system will keep track of and record the progress of the execution of transactions by periodically writing important information like data modifications, commitments or abortions of transactions to a logbook, which is stored in the non-volatile storage system. These log records will be used to re-establish a consistent database state if any failure occurs.

Undo versus redo approaches

There are two main recovery techniques that are *undo* and *redo* [BHG87]. These two approaches support the database systems to reconstruct consistent database states when there is any failure in the database systems. However, they are different in *logging* strategies. The undo logging strategy records in the non-volatile logs the former consistent database states before these database states are changed by a transaction. The redo logging writes to the non-volatile logs the new consistent database states that the database systems will have after the updated transaction commits. Figure 2.8 compares these two logging strategies.

The undo technique supports the database systems to reconstruct the previous consistent database states when a transaction fails. The database system behaves as if none database operation of the aborted transaction has been executed. In other words, the undo technique is used to clean up the presence of data values of uncommitted transactions in

the database system. For the undo approach, the new database states must be written to the database systems after the undo logs have been written to the non-volatile storage [GR93]. Redo technique endorses the database system to re-produce the database states that are the results of successfully committed transactions. The redo approach, therefore, will ignore any uncompleted transaction. Before the new data values are written to the database systems, all the redo log records must be written to the non-volatile storage [GR93]. A recovery support system can combine (which is also the normal case) both undo and redo approaches so that it can decrease the work lost by failures.

Initial states	T_1	Undo Log	Redo Log
X = 10 Y = 20			
	Read(X)	START T_1	START T_1
	X = X + 10		
	Write (X)	<T_1, X, 10>	<T_1, X, 20>
	Read(Y)		
	Y = Y - 10		
	Write (Y)	<T_1, Y, 20>	<T_1, Y, 10>
	C_1	COMMIT T_1	COMMIT T_1

Figure 2.8: Undo logging against redo logging

In Figure 2.8, for the undo approach, if transaction T_1 aborts after it has modified the value of data item *Y*, the recovery system can re-establish the initial database states by two logging records $<T_1,X,10>$ and $<T_1,Y,20>$. For the redo approach, if a failure occurs after transaction T_1 has committed, the database system will re-produce the committed values of transaction T_1 based on two logging records $<T_1,X,20>$ and $<T_1,Y,10>$.

If a new failure happens when the database system is being recovered from previous failures, the recovery procedure has to be able to restart as many times as needed. This feature is called *idempotent* [GR93], i.e., the results of the re-executed recovery procedure are independent of the number of times that they are repeatedly executed.

Recoverability and cascading abort of transactions

When a transaction is aborted, its effect on the database system will be rolled back. If a transaction commits, its results are permanent by the durability property. In other words, a committed transaction does not rollback. A schedule *S* is said to be *recoverable* if no transaction *T* in *S* commits until all transactions *T'* that have updated data items that *T* reads have committed or aborted [BHG87]. A serial schedule is, therefore, always recoverable. Note that a serializable schedule does not forbid a transaction T_i to read from a data item *X* that is modified by an uncommitted transaction T_j (see Figure 2.2, *dirty read* problem). Recovery techniques make no attempt to support the serializability of transactions [GUW01]. Figure 2.9 illustrates the recoverable against serializable schedules. Schedule S_3 is a recoverable schedule because the transaction T_2 that reads new value of data item *X* modified by the transaction T_1 commits after the transaction T_1 has committed. Schedule S_4 is a serializable but non-recoverable schedule because transaction T_2 commits before T_1 commits.

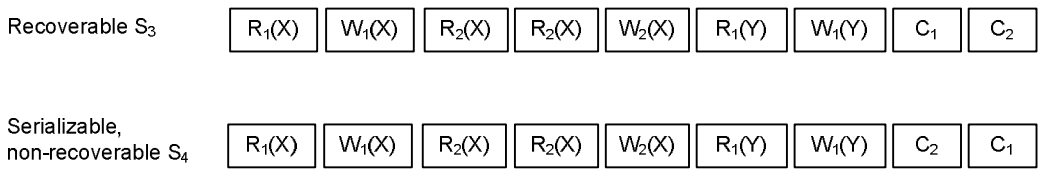

Figure 2.9: Recoverability versus serializability

In a recoverable schedule *S*, a transaction T_i reads data values that are written by an uncommitted transaction T_j, if transaction T_j aborts, T_i must also abort. The abortion of transaction T_i could subsequently cause other transaction T_k to abort if the transaction T_k has been reading data values that are modified by the transaction T_i. This abortion could recursively happen to many other transactions. This phenomenon is called *cascading abort* and is illustrated in Figure 2.10. Unfortunately, recoverable schedule does not prevent the cascading abort problem. Therefore, a stronger condition that only allows a transaction to read data values, which are modified by committed transactions, is needed. An *avoid cascading abort* schedule only allows a transaction to read data values that are written by a committed transaction. Furthermore, a *strict schedule* only allows a transaction to read or write data items that are modified by committed transactions [BHG87].

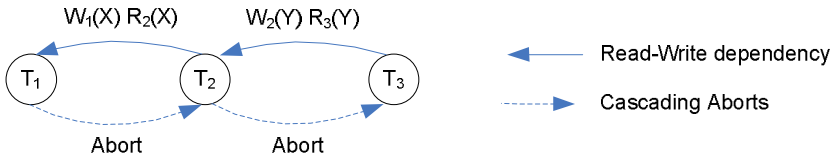

Figure 2.10: A cascading abort scenario

2.2 Transaction processing systems

In this section, we will first discuss the basic and essential components of a transaction processing system that manages the execution of transactions on a transaction-oriented database system. Later, we review the architecture of distributed transaction processing systems.

2.2.1 Essential components of a transaction processing system

A transaction processing system plays a role as a mediator that accepts transaction requests from users, dispatches these requests to the database system, coordinates the execution of the involved transactions, and forwards transaction results to the original acquirers. Figure 2.11 illustrates an interaction model for a transaction-oriented database system.

The common programming model for a transaction-oriented database system is the client-server model [GR93, JHE99]. Users or clients interact with the database system by submitting their transaction processes that consist of one or many database operations to

the transaction processing system. The transaction processing system will coordinate and manage the execution of these transaction processes by subsequently sending these database operations to the database system. The database system will carry out the actual execution of the submitted database operations. Finally, the transaction results that reflect the consistent states of the database system are returned to the clients.

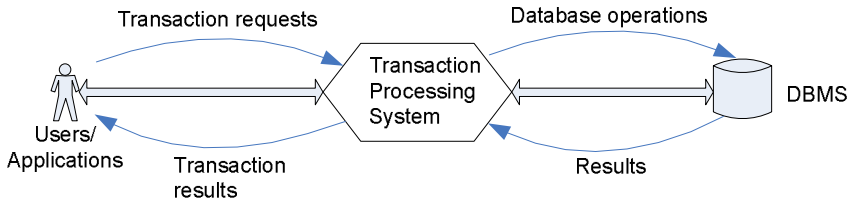

Figure 2.11: Dataflow of transaction-oriented database systems

To protect the integrity constraint of the database system, the transaction processing system must ensure that the ACID properties of transactions are fulfilled. In order to achieve this, a set of essential components that includes a transaction manager, a scheduling manger and a log manger are deployed [GR93]. Additional components such as communication manger or other resource managers can also be employed by the transaction processing system. However, in this section, we will focus our discussion on the three essential components. Figure 2.12 presents the roles of the transaction processing system components.

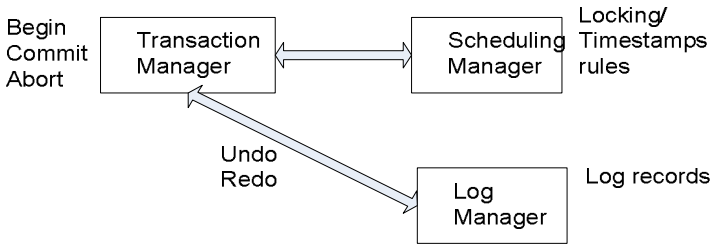

Figure 2.12: Transaction processing system components

The role of each transaction processing component is described as follows:

- **Transaction manager.** The role of the transaction manager is to orchestrate the execution of transactions [GR93]. Via the help of the scheduling and log managers (explained below), the transaction manger takes care of all important operations of transactions such as begin, read, write, commit, and abort (or rollback). If the execution of a transaction is distributed to many different resource managers, the transaction manager will act as the coordinator of the involved participants (explained in Section 2.2.2).

- **Scheduling manager.** The scheduling manger manages the order of execution of the database operations. Usually, the scheduling manager makes use of concurrency

control protocols, for example locking or timestamp protocols, in order to control the execution of transactions. Thus, the scheduling manger supports the isolation and consistency properties of transactions. Based on the applied concurrency control protocol, the scheduling manager will determine an execution order in which the submitted database operations will be carried out. For example, if a locking protocol is used, the scheduling manager will decide whether a lock request will be granted to the acquired transaction, or if a timestamp protocol is applied, the scheduling manager will assess if a submitted operation will be allowed to be carried out.

- **Log manager.** The role of the log manager is to support the database system to recover from failures. The log manager keeps track of the changes of the database states by recording the history of transaction execution. Depending on the deployed recovery strategies, for example *undo* and/or *redo*, the log manger will record necessary information in a non-volatile logbook. The log manager ensures the atomicity and the durability properties of transactions.

The cooperation among the transaction manager, the scheduling manger and the log manager will assure that the ACID properties of transactions in a transaction-oriented database system will be fulfilled.

2.2.2 Distributed transaction processing systems

In the previous section, we have discussed the essential components of a transaction processing system where data is stored in one database system. In this section, we will consider a distributed database system where data is distributed among different computers [OV99]. A distributed transaction processing system is a collection of sites or nodes that are connected by communication networks (see Figure 2.13).

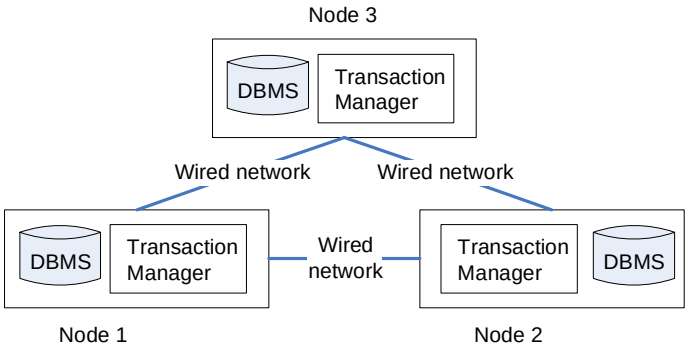

Figure 2.13: Distributed transaction processing systems

The communication networks are usually reliable and high speed wired networks, like LANs or WANs. At each node in a distributed system, there is a local database management system and a local transaction processing system (TPS) that operates semi-independently and semi-autonomously. An execution of a transaction in a distributed database system may have to spread to be processed at many sites. The transaction

managers at different sites in a distributed transaction system cooperate for managing the transaction execution processes.

Transactions in a distributed system can be categorized into two classes: local transaction and global transaction. Consequently, there are two types of transaction manager in a distributed transaction processing system: local transaction manager and global transaction manager [RC96]. Local transactions are submitted directly to local transaction managers (Figure 2.14). Local transactions only access data at one database system at one site, and are managed by the local transaction manager. On the other hand, global transactions are submitted via the global transaction manager. A global transaction can be decomposed into a set of sub-transactions; each of which will be submitted and executed as a local transaction at a local database system [DG00, RC96]. Therefore, the execution of a global transaction can involve accessing data at many sites, and be under control of many local transaction managers. A successful global transaction must meet both the integrity constraints of local databases and the global constraints of the distributed database system.

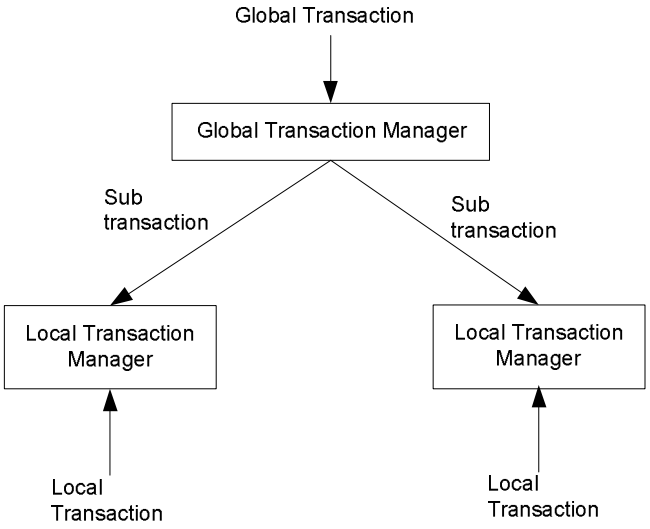

Figure 2.14: Local and global transactions

Some of the potential advantages of the distributed transaction processing system are: (1) higher throughput for transaction processing, and (2) higher availability than the centralized transaction processing system [GR93]. However, the distributed transaction processing system also introduces many challenging issues, for example disconnections in communication between computing sites or concurrency control across computing sites. These problems could cause data inconsistent among database systems, and abort on-going transactions. Consequently, more complicated concurrency control protocols or transaction commitment protocols are needed [BHG87], for example distributed 2-phase locking and 2-phase commit protocols. Moreover, the heterogeneous characteristic of the distributed system must also be taken into consideration [GR93, CDK00], for example different database systems or operating systems.

2.3 Summary

In this chapter, we have reviewed the basic concepts of database systems and database transactions, and discussed the architecture of transaction processing systems in distributed environments. In Chapter 3, we will shift our focus to transactions and transaction processing in mobile environments, which possess some unique characteristics such as the mobility of mobile computing hosts, the limitations of wireless communications and the resource constraints of mobile computing devices [PS98]. We will investigate two important topics: (1) how the distinguishing characteristics of the mobile environments impact transactions and transaction processing systems; and (2) what new requirements a transaction processing system must have in order to efficiently support transaction processing in the mobile environments.

Chapter 3
Requirements for Mobile Transaction Processing Systems

This chapter focuses on the main topic of this thesis: mobile transaction processing systems. The main objective of this chapter is to identify a set of requirements that must be fulfilled by a mobile transaction processing system in order to efficiently support transaction processing in mobile environments. This set of requirements plays a vital role in our research because: (1) it includes the objectives that must be achieved by our mobile transaction processing system, and (2) it contributes to the evaluation of our research results in Chapter 8.

3.1 Introduction

Unlike distributed environments, transaction processing in mobile environments must take into account three new challenging characteristics of mobile environment – that are: the mobility of mobile computing hosts, the limitation of wireless communications and the resource constraints of mobile computing devices [PS98]. These three challenging characteristics have a strong impact on the processing of transactions in terms of concurrency control, data availability, and recovery strategies [Mad+02]. Because of these unique characteristics of the mobile environments, the standard transaction ACID properties can be too strict to be applied in mobile environments. In other words, we need to define a set of requirements that broadens these properties in the context of the mobile environments.

The organization of this chapter is as follows. In Section 3.2, the characteristics of mobile environments and the behavior of mobile hosts are addressed in detail. Section 3.3 discusses transaction processing in mobile environments. Section 3.4 presents the general architecture of mobile transaction environments. The characteristics of mobile transactions are discussed in Section 3.5. Based on these characteristics, a set of requirements, which must be fulfilled by our mobile transaction processing system, is identified and addressed in Section 3.6. Finally, Section 3.7 concludes the chapter.

3.2 Characteristics of mobile environments

In this section, we discuss the characteristics of the mobile environments that could have strong impact on mobile transactions in terms of transaction specification and transaction processing. There are other important issues like authentication and security; however, they are not in the scope of this thesis. The main characteristics of the mobile environments that are addressed in this section include: the mobility of mobile computing hosts, the limitation of wireless communications and the resource constraints of mobile computing devices. In this chapter, we will use the *mobile transaction* terminology for specifying transactions in mobile environments.

3.2.1 Mobile hosts

Mobility is the main characteristic that distinguishes the mobile environments from the traditional distributed environments. In traditional distributed environments, computers are stationary hosts. In mobile environments, mobile computers are continuously moving from one geographical location to another.

The features of the mobility characteristic are discussed as follows:

- **Real-time movement.** The mobility of the mobile host is a real-time movement. Therefore, it is affected by many environment conditions. For example, the pre-planned travel route of a mobile host can be changed because of traffic jams or weather conditions. If there is a mobile task whose operations depend on the travel route of the mobile host, these operations can become invalid, or extra support is required. For example, a new route-map directory must be downloaded into the mobile host if the travel course is changed. Moreover, the movement of the mobile host can also depend on the objective of the mobile task. For example, an ambulance car wants to arrive at the accident scene by selecting the shortest route with fastest allowing speed, a bus must follow a strict time table on a bus-route, while a postman only wants to travel through each road once. During the movement, the mobile host can stop at some locations for some periods; therefore, the mobility of the mobile host includes both movement and non-movement intervals.

- **Change of locations.** The location of a mobile host changes dynamically and frequently in accordance with the speed and the direction of the movement. The faster the mobile host moves, the more frequently the location changes. The objective of mobile tasks can also specify the locations at which the mobile host must be, in order to carry out the mobile tasks. For example, a computer technician must come to customer locations to fix computer problems. A mobile support system must provide the utilities to manage the locations of mobile hosts (this demand is not needed in a distributed environment). Changes of locations can cause changes in the operating environments of the mobile hosts, for example network addresses, communication protocols, mobile services, or location dependent data [Ram+03, DK98].

The mobility of mobile hosts will have strong impact on the execution of transactions. The real-time movement of mobile hosts could pose timing constraints on the execution schedule of transactions. Furthermore, if mobile hosts change their locations frequently, additional time is required to reconfigure transaction application processes to the new environment conditions. Therefore, additional support is required for mobile transaction processing systems to cope with these challenges.

3.2.2 Wireless networks

Mobile hosts communicate to other hosts via wireless networks. Compared to wired networks, wireless networks are characterized by: lower bandwidth, unstable, disconnections, and ad-hoc connectivity [Sch02]. The characteristics of the wireless networks are described as follows:

- **Lower bandwidth.** The bandwidth of a wireless network is lower than a wired network. The wireless network does not have the capacity as the wired network. For example, a wireless network has bandwidth in the order of 10Kbps or a wireless local area network (WLAN) has bandwidth of 10Mbps; while gigabits (Gbps) are common in wired LAN [Sch02]. Therefore, it can take longer time for a mobile host to transfer the same amount of information via the wireless network than the wired network. Consequently, the wireless network introduces more overhead in transaction processing.

- **Unstable networks.** A wireless network has high error-rates, and the bandwidth of a wireless network is variable. Due to errors during data transmission, the same data packages are required to re-transmit many times, thus, extra overhead in communication and higher cost. Due to the varying bandwidth, it is hard to estimate the time required to completely transmit a data package from/to a mobile host. These problems will affect the data availability at the mobile hosts. As a result, the execution schedule of transactions at the mobile hosts can be delayed or aborted.

- **Disconnections.** Wireless networks pose disconnection problems. Disconnections in communication can interrupt or delay the execution processes of transactions. More seriously, on-going transactions could be aborted due to a disconnection. The disconnection in communication is categorized into two types: disconnection period and disconnection rate.

  Disconnection period. The disconnection period indicates how long a mobile host is disconnected. While being disconnected, the mobile host will not be able to communicate to other hosts for sharing of data. If the mobile host holds vital shared data, it can block transaction processes on other hosts. Furthermore, the duration of a disconnected period of a mobile host is not always as planned, i.e., it can be longer than expected. The mobile transaction processing system must be able to continuously support transaction processing while the mobile host is being disconnected from the database servers by caching the needed data beforehand.

Disconnection rate. The disconnection rate indicates how often the wireless communication is interrupted within a predefined unit of time. The execution of transactions on a mobile host can be affected when an interruption occurs. The more interruptions the many transactions are aborted or rollback. If the transactions on the mobile host are carrying out collaborative operations with other transactions on other mobile hosts, these collaborative activities can be suspended or aborted. To cope with this problem, the mobile transaction processing system must be able to support the mobile transactions to resume or recover from previous interrupted points.

- **Ad-hoc communication.** The wireless network technologies introduce a new way to support direct and nearby communications among mobile hosts, also called *any-to-any* or *mobile peer-to-peer* communication [Sch02, Rat+01]. For example, two mobile hosts can directly share information with the support of Bluetooth or infra-red technologies [PLZ05]. The characteristics of this peer-to-peer communication are: unstructured (i.e., ad-hoc), short-range, and mobility dependent [Rat+01]. Table 3.1 compares the communication ranges and bandwidth of different wireless technologies.

Table 3.1: Wireless communication technologies[1]

Wireless technology	IEEE standard	Range (m)	Bandwidth
IrDA[2]	N/A	0.1-1	100kbps – 16Mbps
Bluetooth	IEEE 802.15.1	10-100	1Mbps
Wireless USB	IEEE 802.15.3[3]	1-10	2Mbps-480Mbps
Wi-Fi	IEEE 802.11	45-90	11Mbps-540Mbps
WiMAX	IEEE 802.16	2km-10km	75Mbps

3.2.3 Computing devices

There are many types of mobile computing devices such as mobile phones, laptop computers, or personal digital assistants (PDAs). Mobile devices are subject to be smaller and lighter than stationary computers. Consequently, mobile computers have limited energy supply, less storage capacity, and limited functionality compared to stationary computers. Furthermore, the mobile computers are easily damaged, i.e., less reliable. The characteristics of mobile computing devices are elaborated as follows:

- **Limited energy supply**. The operation of mobile computers heavily depends on the electrical power of batteries. This limited power supply is one of the major disadvantages of mobile computing devices. The energy consumption of a mobile device depends on the power of electronic equipments installed on the mobile device, for example types of hard disks or CPU. Moreover, the battery life also depends on the number of applications and the application types that operate on the mobile

[1] Sources: www.irda.org, www.bluetooth.org, www.ieee802.org, and www.intel.com
[2] IrDA stands for Infrared Data Association
[3] Yet to be standard

devices [FS99, KU99]. For example, a mobile phone can live up to five days but a laptop can only be able to operate for several hours; text processing applications consume less power than graphical applications. Transaction processes that are being carried out at a mobile host can be interrupted or re-scheduled if the mobile host is exhausting its power supply.

- **Limited storage capacity**. The storage capacity of a mobile computer (i.e., hard disks or memory) is much less than a stationary computer and is harder to be expanded. Therefore, a mobile host may not be able to store the necessary data that is required for its operations in disconnected mode [PS98, Mad+02]. Consequently, transaction processes on the mobile host will be delayed due to data unavailability, or require longer processing time due to frequent memory swapping operations.

- **Limited functionality.** The functionality of mobile devices is also limited in terms of the graphical user interface, the application functionalities, and the processing power. Therefore, a mobile host may be unable to perform some of transaction operations, or requires longer processing time to perform these operations. For example, a small PDA may only be able to view black and white pictures. Table 3.2 compares the configurations of several PDA types.

Table 3.2: Personal digital assistant devices[4]

PDA type	Size and weight (cm, gram)	Screen size (inch, color bits)	Processor type (MHz)
HP iPAQ Pocket PC hx2110	7.7 x 1.6 x 11.9, 164 g	3.5”, 16 bits	Intel XScale 312
ASUS MyPal A620BT	7.7 x 1.3 x 12.5, 141 g	3.5”, 16 bits	Intel XScale 400
Fujitsu Siemens Pocket LOOX 720	7.2 x 1.5 x 12.2, 170 g	3.6”, 16 bits	Intel XScale 520

- **Unreliable equipments**. The data stored at a mobile host can be lost if a catastrophic failure happens. This could heavily impact the durability property of transactions because of the losing of the committed results of transactions that are stored at the mobile host. To avoid this problem, data stored at mobile hosts must be backed-up at stationary database servers as much and as soon as possible.

3.2.4 The behavior of mobile hosts in mobile environments

In mobile environments, mobile transactions are initiated [DHB97, KK00] and/or processed [WC99] at mobile hosts. The mobile hosts can participate in the mobile transaction execution processes in different ways. First, a mobile host can initiate a mobile transaction, submits the transaction to appropriate (non-mobile or mobile) hosts for processing, and receives the committed results. In this way, the mobile host plays a

[4] Sources http://www.komplett.no

role as a terminal transaction client [GR93]. Second, a mobile host can take part in the actual transaction execution process, i.e., the entire or part of a mobile transaction is carried out by the mobile host. The mobile host plays a vital role in the transaction execution process. Therefore, we need to answer the following question: How do the characteristics of the mobile environments affect the behavior of the mobile host?

The behavior of mobile hosts in mobile environments is categorized into two dimensions: *movement* and *operation* (see Figure 3.1).

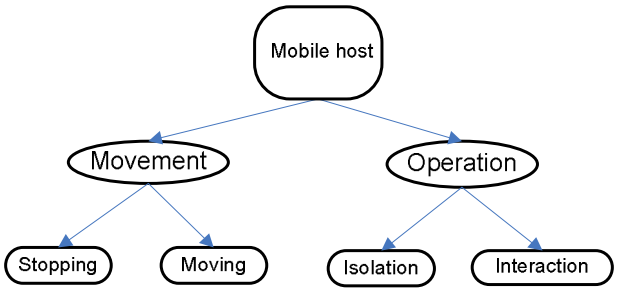

Figure 3.1: Behavior model for mobile hosts

First, the movement of the mobile host is affected by both the requirements of the mobile tasks and the environmental conditions [DK99, Sør+02]. Second, the operation of the mobile host depends on its internally designed capacity and externally associative factors. For example, the performance of computational operations depends on the available energy of the mobile host's battery, and the network operations rely on both the connectivity capacity of the mobile host and the provided network services. The behavior of mobile hosts is discussed in the following.

Movement of mobile hosts

The movement behavior of a mobile host describes the actual mobility states of the mobile host. While operating in mobile environments, the mobile host can be either in *stopping* or *moving* state. The two movement states are explained as follows:

- **Stopping.** A mobile host is said to be in stopping state either when its movement velocity is zero, or when the location of the mobile host is not considered changing within a period of time. For example, a bus stops at a bus-stop to pick up passengers, a salesman is selling products at a shopping centre, or two mobile hosts are always moving close to each other.

- **Moving.** A mobile host is in moving state either when its movement velocity has a value greater than zero, or when the location of the mobile host is considered changing over time. For example, a bus is moving along a road or a salesperson travels to several places during the day. While in moving state, the mobile host can continuously change its velocity and direction of movement.

On the one hand, the movement behavior of a mobile host can affect the mobile tasks that are carried out by the mobile host, e.g., a public transport vehicle needs to strictly follow a timetable. On the other hand, the movement of the mobile host can be affected by the surrounding environment conditions, e.g., traffic jam. The movement behavior of the mobile host demands additional supports such as location management [MRX03], and awareness of location dependent data [RD00, DK98].

Operations of mobile hosts

The operation behavior of mobile hosts depends on the availability of mobile resources such as network connectivity and battery energy. We distinguish two operation modes for mobile hosts in mobile environments: *isolation* and *interaction*. These operation modes of the mobile hosts are explained as follows:

- **Interaction.** When a mobile host is sharing data with other hosts, it is said to be in an interaction mode. The two essential prerequisite conditions for the interaction mode are: (1) the mobile host is operational, and (2) the network connectivity is available. It is not necessary that the mobile host always connects to other hosts all the times. This can help the mobile host to save the battery energy and to reduce communication cost. However, in an interaction mode, the communication channel between the mobile host and other hosts must always be available and establish-able whenever it is needed.

- **Isolation.** When the communication channel between a mobile host and other hosts is not available, the mobile host is disconnected from other hosts and is said to be in an isolation mode. There are many factors that contribute to disconnection of the mobile host, for example the mobile host moves out of the wireless communication range, or network services are not available, or the mobile host is running out of its energy. The isolation mode can be further refined to *autonomous* and *idle* sub-modes.

  *Autonomous***.** When a mobile host operates by itself, it is said to be in autonomous mode. In the context of mobile transaction processing, we refer this mode as disconnected processing mode (see Section 6.5).

  *Idle***.** In this mode, the mobile host is not able to operate or has to delay its operations.

The behavior of mobile hosts also illustrates the correlations among the three characteristics of the mobile environments. Disconnections in communication can be the results of the mobility of mobile hosts and/or the limitation of mobile resources. When mobile hosts communicate with others via short-range wireless network technologies, e.g., infra-red or Bluetooth or wireless LAN, the communication will be disconnected if the mobile hosts move outside the limited communication range. The mobile hosts can be disconnected for short periods, i.e., seconds or minutes, and more frequently when they are moving in and out of the shadow of physical obstructions such as high buildings. The disconnection period can be long, i.e., hours or days, when the mobile hosts stay in some locations in which the wireless network service is not available. The mobile hosts can

also volunteer to disconnect if the supplied energy is running out. On the other hand, the heavy use of network activities can shorten the battery life of the mobile host.

3.3 Transaction processing in mobile environments

The main differences between the mobile environments and distributed environments are: (1) mobile computing hosts, and (2) wireless networks. Table 3.3 compares the main different features between the distributed and mobile environments.

Table 3.3: Distributed environments versus mobile environments

	Distributed environments	**Mobile environments**
Computing hosts	Stationary sites Powerful computing capacity Reliable computing hosts	Mobile and non-mobile hosts Limited computing capacity of mobile hosts Less reliable computing hosts
Network connectivity	Wired and high-speed networks Reliable networks	Wireless, unstable and low speed networks Unreliable, error-prone, frequent and long disconnection periods

The mobile hosts usually have less computing resources and capacity than stationary hosts. For example, a laptop computer has lower processing speed and smaller storage capacity than a desktop computer, and its operation might depend on the limited battery energy. Consequently, it takes longer time for a transaction to be processed at a mobile host. Moreover, mobile computers are easily damaged, i.e., less reliable. The results of committed transactions, which are stored at a mobile computer, can be lost if the mobile computer is damaged, i.e., the durability property of transactions may not be fully guaranteed. Therefore, the committed results of transactions in mobile environments should additionally be saved at the stationary hosts as in distributed environments. The movement of mobile hosts brings additional requirements and demands that the mobile transaction processing system must handle, for example hand-over processes [DHB97] or locally dependent data [DK99]. In distributed environments, these demands do not exist.

Mobile computing hosts communicate with other hosts via wireless networks. Compared to a wired network, a wireless network is usually less reliable, i.e., disconnections can occur frequently; has lower bandwidth, i.e., megabits versus gigabits; and is limited in communication range, i.e., mobile hosts must stay within limited distance to be connected. Because of these unique features of wireless networks, it can take longer time to download necessary data into the local storage devices at the mobile hosts; or due to disconnections, the mobile hosts will not be able to obtain the needed data. Consequently, transactions in mobile environments may experience long blocking periods and inconsistent data.

In mobile environments, transaction processing systems consist of both mobile and non-mobile hosts [SRA04], and can be divided into two different layers (see Figure 3.2). The

distributed transaction processing layer corresponds to the execution of mobile transactions that are carried out on non-mobile hosts. The mobile transaction processing layer corresponds to the execution of mobile transactions that are carried out on a mobile host or distributed among mobile hosts. Due to the above distinguishing and challenging characteristics of mobile environments, transaction processing in mobile environments is more difficult than in distributed environments, especially in terms of concurrency control, data availability, and recovery mechanisms [Mur01]. These characteristics of mobile transactions will be discussed in Section 3.5.

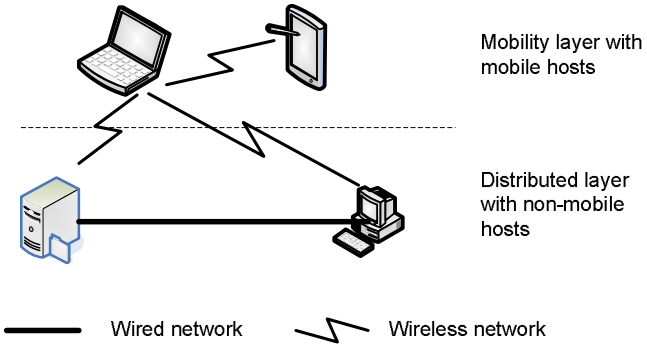

Figure 3.2: Transaction processing in mobile environments

3.4 Architecture of mobile transaction environments

In this section, we discuss the architecture of the mobile transaction environments. In general, the mobile transaction environments include three different components: mobile hosts (MH), mobile support stations (MSS) and fixed hosts where database servers (DB) reside [SRA04, Hir+01]. Figure 3.3 illustrates the mobile transaction environments.

A mobile environment is a geographical territory. The geographical territory is divided into a collection of areas called mobile cells. Wireless communications in each mobile cell is provided by a single low-power transmitter-receiver [Sch02]. There might be some areas in the mobile environments in which the wireless communication is not available. This could be caused by the limited service of the wireless communication providers or the structural of physical objects in the areas, for example concrete tunnels or remote islands. The geographical mobile environment, therefore, can be considered as a collection of mobile cells that are separated or overlapped with others. The size of mobile cells is not necessarily equal, due to the differences of operational power of the transmitter-receiver devices.

The wireless technologies that are provided in each mobile cell can be different, for example wireless LAN or wireless USB. As a consequence, network bandwidth, network latency, communication protocols and covered ranges are different among mobile cells. In each mobile cell, there is a special computing host called the mobile support station. The role of the mobile support station is to provide additional computing services to all the mobile hosts that currently reside in the mobile cell.

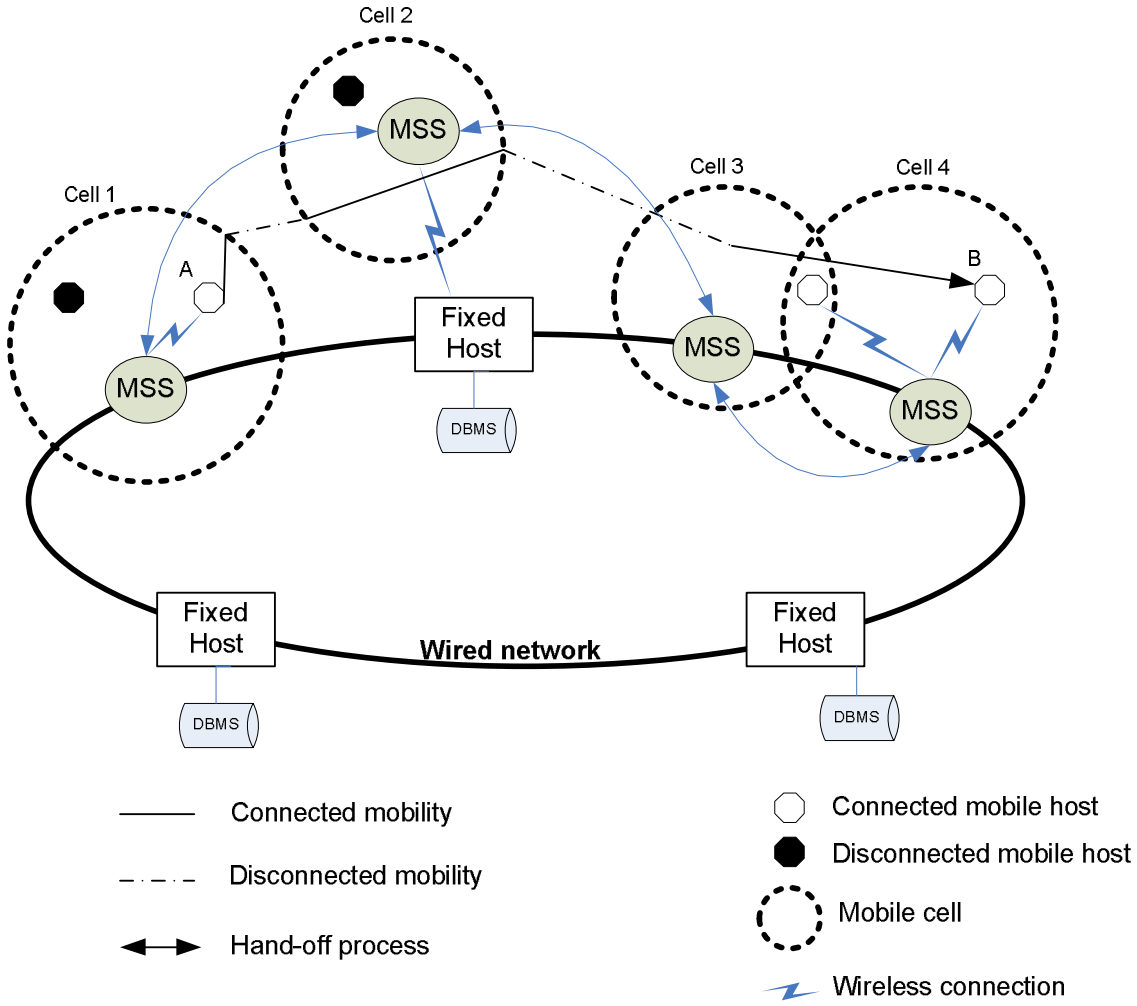

Figure 3.3: Mobile transaction environments

Mobile hosts are portable mobile computing devices which have the capability to cache a limited amount of information. Database servers are stationary computers that are connected via high speed wired-networks, and play roles as permanent data storage repositories. Shared data is distributed on these database servers. Mobile support stations (also called base stations) are stationary or mobile computers. Mobile support stations have higher processing power and data storage capacity than the mobile hosts. The role of the mobile support stations is to support mobile hosts communicating with other mobile hosts or database servers. Mobile hosts communicate with the mobile support stations via the wireless networks. Communications between the database servers and the mobile support stations are via wired networks or dedicated wireless connections.

Mobile hosts move in mobile environments while carry out mobile tasks. While being in a mobile cell, a mobile host can be either connected or disconnected with the mobile support station of this mobile cell. The mobile host may only connect to the mobile support station when there is a need for sharing of data. This will help to save the limited energy of the mobile host and to reduce the communication cost. On the other hand,

because of the limitations of wireless networks, a mobile host may not always be able to establish a communication channel with the mobile support station. If a mobile host is in the area that is an intersection of two or more mobile cells, it can connect to any mobile support station.

The mobile hosts can move within one mobile cell or across a large area covered by several mobile cells. When a mobile host is leaving a mobile cell and entering a new mobile cell, the communication channel and other related information between the mobile host and the previous mobile support station will be transferred to the next mobile support station. This process is called *hand-over* or *hand-off* process [SRA04]. The new mobile support station at the new mobile cell will continue carrying out the support to the mobile host. However, it is not necessary that hand-over processes must happen every time the mobile host enters a new mobile cell. For example, the mobile host can operate in an autonomous mode when the wireless network is not supported in the new mobile cell. Furthermore, a mobile host does not have to disconnect from the old mobile support station before it can connect to the new mobile support station. As shown in [CP98, TLP99], a mobile host can connect to a new mobile support station while connecting to the old mobile support station. The hand-off process can be planned beforehand if the travel route of the mobile host is known in advanced and strictly followed. Otherwise, the hand-off process can only be carried out after the mobile host has established a connection with the new mobile support station, i.e., after the new destination of the mobile host is known.

In Figure 3.3, there are four mobile cells in the mobile environments. Mobile cells one and two are separated, while mobile cells three and four are overlapped. A mobile host moves from position *A* in mobile cell one to position *B* in mobile cell four. The travel route of the mobile host passes through mobile cells two and three. When the mobile host is leaving cell one, it will enter a disconnected interval in the area between the mobile cells one and two. While in the mobile cell two, the mobile host will be supported by the mobile support station that is a dedicated mobile host. When the mobile host is in the mobile cell three, it may not connect to the mobile support station all the time. In the intersection region of the mobile cells three and four, the mobile host can connect to the mobile support station of either mobile cell three or mobile cell four. The hand-over processes occur when the mobile host moves from one mobile cell to another along the travel route.

3.5 Characteristics of mobile transactions

Transactions in mobile environments possess many challenging characteristics due to the characteristics of the mobile environments. In this section, we will discuss the characteristics of mobile transactions. The characteristics of mobile transactions are described as follows:

- **Mobility of transactions.** The execution of transactions in mobile environments is tightly coupled with the behavior of the mobile hosts. A mobile host can initiate mobile transactions or participate in the transaction execution processes. When a

mobile host moves from one location to another, all the transactions that are being carried out at that mobile host will also move. Consequently, many computing activities associated with these transactions are moved or changed, for example handling hand-over processes, establishing new communication channels, or updating the routing tables. In other words, the mobility of transactions causes the movement of related transaction resources, controls, and services.

- **Long-lived transactions.** Transactions in mobile environments have longer life (i.e., long-lived) than traditional ACID transactions. This is due to the overheads that are caused by two aspects: the data availability and the execution interruptions (see Figure 3.4).

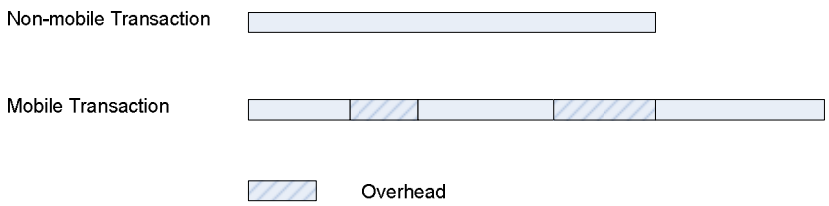

Figure 3.4: Transaction life-time in non-mobile and mobile environments

Data availability. In mobile environments, the data availability at a mobile host can be affected by many factors. First, the movement of the mobile host causes the movement of related information. This will cause additional overhead to the transaction execution time. Second, the bandwidth of wireless networks is limited; therefore it will take longer time to obtain the necessary data. Third, the mobile computing devices have limitations in storage capacity; therefore, the mobile host may not able to cache the required information to support disconnected transaction processing. In addition, due to the unexpected disconnections of the wireless networks, a transaction will not be able to release the controls on shared data to transactions at other hosts as scheduled; this means that this transaction blocks the execution of other transactions.

Execution interruptions. The execution of transactions can be interrupted while being carried out at the mobile host. The interruptions can be caused by either the surrounding environment conditions or the limitation of computing capacity of the mobile host. For example, a wireless network disconnection suddenly occurs during the execution of transactions, or the performance of the mobile host is slowing down due to heavy computing load. The interruptions can happen frequently and cause transactions to be suspended or aborted.

- **Adaptive transaction processing.** Due to the real-time movement of the mobile hosts, the limitations of the wireless networks, and the variation of the mobile resources, the execution plan of a transaction in mobile environments may not be as scheduled. Therefore, the mobile transaction processing system must have the ability to support adaptive transaction processing that includes: distributed and disconnected transaction processing.

Distributed transaction processing. Due to the limitations of processing capacity and resources, mobile hosts require additional support from other hosts to carry out transactions. For example, a transaction, which is initiated by a mobile host and accesses a large data set that is not cached at the mobile host, could be moved to stationary hosts for executing. This could reduce transaction processing time and avoid transferring a large amount of data through a slow wireless network, i.e., achieving higher throughput for the transaction processing system. Furthermore, the portable computing devices are easily damaged; therefore, the results of committed transactions must be saved at stationary database servers.

Disconnected transaction processing. A mobile host can be disconnected from the database servers for long periods; therefore, transactions that are executed at the mobile host may suffer from long blocking if the necessary data is not available at the mobile host. To deal with this problem, the mobile transaction processing system should have the capacity to cache enough data so that it can carry out the transactions while being disconnected from the database servers.

- **Temporary data inconsistency.** Due to long disconnection periods, shared data among different mobile hosts may not be fully consistent all the time. For example, a transaction at a disconnected mobile host can modify a shared data item that is currently being read-only cached in a local storage of another disconnected mobile host. Data synchronization processes will be carried out when the disconnected mobile hosts reconnect to the database systems so that the data consistency of the database systems will be achieved.

- **Heterogeneous processing**. Many types of mobile devices can be involved in transaction execution processes. Interactions or communications among participating parties are carried out via the support of different types of wireless network technologies and protocols. Furthermore, different database systems are accessed during the execution of mobile transactions. All these factors contribute to the heterogeneous processing characteristic of mobile transactions.

3.6 Requirements of transactions in mobile environments

In this section, we address in detail the requirements of a mobile transaction processing system that have been briefly mentioned in the Section 1.5 of this thesis. Because of the challenging characteristics of mobile transactions, the ACID properties of transaction are too strict to be applied in the mobile environments. More relaxing transaction properties have been introduced to support transaction processing in the mobile environments. A common approach is that the atomicity and isolation properties could be relaxed, while the consistency and durability properties must be preserved [RC96, SRA04].

In this thesis, in relation to the transaction properties, we will apply the same approach. However, we also propose additional requirements that take into account the characteristics of mobile transactions like the mobility of transactions, and the heterogeneous and adaptive transaction processing. In order to achieve the objectives, we

identify nine requirements that a mobile transaction processing system must have. The requirements are based on four categories: *mobility of transactions* (R1 and R2), *wireless networks* and *limited mobile resources* (R3 and R4), *customization of transaction properties* (R5, R6, and R7), and *recovery of transactions* (R8 and R9). The requirements are summarized in Table 3.4.

Table 3.4: Requirements of mobile transaction processing systems

Categories	Requirements
Mobility of transactions	**R1.** *The mobile transaction processing system must be able to effectively handle the hand-over control of mobile transactions*.
	R2. *The mobile transaction processing system must support interactions among transactions at different mobile hosts.*
Wireless networks and limited mobile resources	**R3.** *The mobile transaction processing system must support disconnected transaction processing*.
	R4. *The mobile transaction processing system must support distributed transaction execution among mobile hosts and stationary hosts*.
Customization of transaction properties	**R5.** *The mobile transaction processing system must have the ability to customise the atomicity property of transactions.*
	R6. *The mobile transaction processing system must support sharing partial states and status among transactions*.
	R7. *The mobile transaction processing system must assure the durability property of transactions*.
Recovery of transactions	**R8.** *The mobile transaction processing system must provide efficient recovery strategies.*
	R9. *The mobile transaction processing system must support temporary data and transaction management.*

The above requirements are elaborated as follows:

R1. *The mobile transaction processing system must be able to effectively handle the hand-over control of mobile transactions*. Mobility of hosts is one of the main challenging characteristics of mobile environments that cause the mobility of transactions. The mobility of a mobile transaction can be described in terms of hand-over processes that occur during the execution of the mobile transaction. Therefore, the mobile transaction processing system must be able to capture and control these hand-over processes. This can be achieved if the mobile transaction processing system is able to

identify (1) when a hand-over process occurs, and (2) which information is needed to move or to modify in accordance with the mobility pattern of mobile transactions.

R2. *The mobile transaction processing system must support interactions among transactions at different mobile hosts.* While being on the move and disconnected from the database servers, mobile hosts can directly communicate with others by using short-range and peer-to-peer communication technologies, for example infra-red, Bluetooth or wireless LAN. The mobile transaction processing system must be able to support direct interactions among transactions at different mobile hosts, i.e., without any support from the mobile support stations or the database servers.

R3. *The mobile transaction processing system must support disconnected transaction processing*. In mobile environments, the mobile hosts are frequently disconnected from the database servers. Therefore, the mobile transaction processing system must support disconnected transaction processing, i.e., to deal with the disconnections of the wireless networks, especially long disconnection periods. This will allow the mobile hosts to continue processing transactions in isolation mode and, hence, reducing the delay of local transactions.

R4. *The mobile transaction processing system must support distributed transaction execution among mobile hosts and stationary hosts*. Due to the limited computing resources of mobile devices, the mobile transaction processing system must be able to distribute the execution of transactions among available computing hosts. For example, if a mobile transaction requires a lot of processing capacity or the amount of requested data of the mobile transaction is large, the mobile transaction should be transferred to fixed hosts to be processed there. This approach, in addition, will avoid the problem of transferring a large amount of data from the database servers to the mobile host on the low bandwidth and frequently disconnecting wireless networks.

R5. *The mobile transaction processing system must have the ability to customise the atomicity property of transactions.* The standard atomicity property of transactions is too strict in mobile environments, especially for long-lived transactions. Therefore, the mobile transaction processing system must provide mechanisms to customize the atomicity level of transactions. In other words, the mobile transaction processing system must support transactions to partially roll back when failures occur. For example, a transaction will be partially rolled back (i.e., not totally aborted) due to a failure caused by the exhausting power supply at the mobile host or the disconnection of wireless networks. The mobile transaction can be continued when these mobile resources become available. Customizing the atomicity property of transaction also avoids losing of useful work done due to the failures of the mobile hosts.

R6. *The mobile transaction processing system must support sharing partial states and status among transactions*. Sharing partial results is essential in mobile environments. For example, if a shared data object is only accessible after the transaction that is being executed at a mobile host has finally committed at the database servers; other transactions can suffer long blocking periods. Furthermore, mobile transactions are long-lived

transactions, therefore, the mobile transaction processing system must allow partial results of on-going transactions to be shared.

R7. *The mobile transaction processing system must assure the durability property of transactions*. In mobile environments, mobile transactions are executed and locally committed at the mobile hosts, and globally committed at the database servers. Mobile computing devices are easily damaged; therefore, the results of committed transactions saved at mobile hosts can be lost if failures happen. Thus, the mobile transaction processing system must provide different methods to safely archive information in accordance with the commitment (i.e., locally or globally) of mobile transactions.

R8. *The mobile transaction processing system must provide efficient recovery strategies.* When a transaction fails, the recovery techniques support the database systems to restore consistent states. In mobile environments, failures are common due to many factors, for example the disconnections of wireless communications or the exhausting of the battery energy. Furthermore, cascading abort can happen if a transaction aborts after sharing their partial results to other transactions. Therefore, the transaction processing system must support different recovery methods to deal with different transaction failure situations. For example, if a transaction that shares consistent data to other transactions aborts, those transactions that have read the shared consistent data should not be aborted (see the concepts of shared transactions in Section 5.5 for more detail).

R9. *The mobile transaction processing system must support temporary data and transaction management.* The execution processes of mobile transactions can happen at different computing (mobile or non-mobile) hosts that can be asynchronously connected or disconnected. For example, a transaction at a disconnected mobile host reads a shared data object that is being modified at another mobile host. Therefore, the non-permanency of data and transactions behavior must be managed. The temporary management must also handle conflicts among transactions at different mobile hosts.

3.7 Summary

Because of the unique characteristics of the mobile environments (that are: the mobility of the mobile hosts, the limitations of wireless networks, and the resource constraints of the mobile computers), mobile transactions are very different from traditional transactions.

In [GR93], Jim Cray and Andreas Reuter gave a definition of transaction as:

> "*A transaction is a collection of one or more operations on the database that must be executed atomically*".

Serrano-Alvarado et al. [SRA04] defined a mobile transaction as:

> "*A mobile transaction is a transaction where at least one mobile host takes part in its execution*".

The focus has moved from the transaction design to where and how transactions are executed. Mobile transactions are more complicated than traditional transactions in both specification and execution, due to, for example disconnection in communications or hand-over processes. In order to support the development of our mobile transaction processing system, we have addressed and discussed the requirements that a mobile transaction processing system must face. These requirements not only focus on customizing the transaction properties, but also take into account other challenging characteristics of mobile transactions such as mobility of transactions, and disconnected and distributed transaction processing.

There are many mobile transaction models, analyzing tools and transaction processing systems [SRA04, Hir+01] that have been proposed and developed to support mobile transaction processing. However, there are still major limitations, especially to support both the disconnected processing and the mobility of transactions. These limitations will be investigated in Chapter 4.

Chapter 4
State-of-the-Art Survey

In this chapter, we survey existing mobile transaction models to answer the research question: *What are the current ideas and concepts that have been developed to answer the main research question or to address part of it?* Therefore, the objective of this chapter is to analyze what have been done and find out what are the limitations in the field of mobile transaction processing, focusing on both academic and practical research.

4.1 Introduction

In this chapter, we survey several selected transaction models and transaction processing systems that have been purposely developed to support transaction processing in mobile environments. We will also recap some traditional transaction models whose features could be used in the mobile environments. Based on the characteristics and requirements of mobile transactions that have been addressed in Chapter 3, we will comment on the implications, usefulness as well as the limitations of these models and systems.

The chapter is organized as follows. Traditional transaction models are reviewed in section 4.2. We discuss why they are important, and how these models can be used in mobile environments. Mobile transaction models and mobile transaction processing systems that are recently developed are surveyed and commented in section 4.3. Other related issues to mobile transactions like mobile databases, transaction commitment protocols, and data sharing workspaces will be considered in section 4.4. In section 4.5, we will look into some available commercial transaction systems. This is to find out what the gap between theoretical and practical research is. Summary of the literature review is given in section 4.6.

4.2 Traditional transaction models

As the transaction environment evolves from the centralized environment to distributed and mobile environments, the properties and the structure of transactions change. However, several basic transaction models are indispensable. In other words, they are still useful and applicable in the new mobile environments. In this section, we will review the following transaction models:

- *Flat* transaction model [Gra81, GR93]
- *Nested* transaction model [Mos85]
- *Multilevel* transaction model [Wei91, Elm+92]
- *Sagas* transaction model [GMS87]
- *Split and Join* transaction model [PKH88]

For each transaction model, we briefly describe the transaction model, the properties and discuss how the features of the transaction model could be used in the mobile environments.

4.2.1 Flat transaction model

Description. The flat transaction model [Gra81, GR93] presents the simplest transaction structure that fully meets the ACID properties. Figure 4.1 illustrates the structure of a flat transaction. The building block of a flat transaction, between Begin and Commit /Abort operations, contains all the database operations that are tightly coupled together as one atomic database operation. The flat transaction begins at one consistent database state, and either ends in another consistent state, i.e., the transaction commits, or remains in the same consistent state, i.e., the transaction aborts.

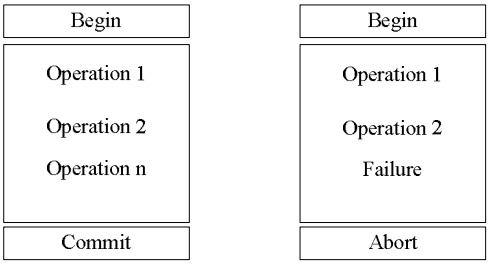

Figure 4.1: Flat transaction model

Transaction properties. The flat transaction model fully meets the standard ACID properties. The flat transaction is fully isolated during its execution, and any failure causes the whole transaction to abort. The results of a committed flat transaction are durable and permanent.

Usefulness for mobile environments. Due to the strict ACID properties, the flat transaction model is not suitable in mobile environments. However, the flat transaction model plays an important role for building more advanced transaction models. For example, a complicated transaction model can consist of a set of smaller flat transactions. The flat transaction model can be easily supported at the application programming level.

4.2.2 Nested transaction model

Description. The nested transaction model [Mos85] defines the concepts and the mechanisms for breaking up the large building block of a flat transaction into a set of smaller transactions, called *sub-transactions*. Thus, the nested transaction model has a

hierarchical tree structure that includes a top-level transaction and a set of sub-transactions (either parent or children transactions). Sub-transactions at the leaf level of the transaction tree are flat transactions.

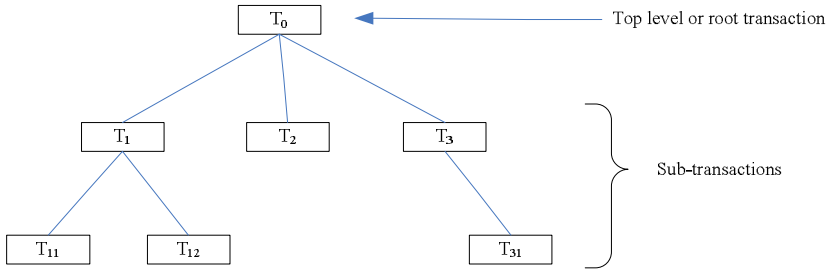

Figure 4.2: Nested transaction model

Transaction properties. The nested transaction model has the following characteristics. First, children transactions are flat transactions. Second, the children transactions start after their parent have started, and can autonomously commit or abort. However, the results of the committed children transactions do not take effect until their parent transactions commit. In other words, the nested transaction only commits when the top-level transaction commits. And third, when a child transaction commits, its results become visible to its parent transaction. If a parent or the top-level transaction aborts, all the sub-transactions must abort, regardless of their states.

Usefulness for mobile environments. The concept of the nested transaction model can be applied in mobile environments, especially for decomposing a large transaction into sub-transactions which can be carried out concurrently.

4.2.3 Multilevel transaction model

Description. The multilevel transaction model [Wei91, Elm+92] is looser than the nested transaction model in terms of the relationship between parent and children transactions. Sub-transactions in the multilevel transaction can commit or abort independently of their parents. This is supported by the concepts of compensating transactions. We will briefly discuss the concept of *compensating transactions,* and its opposed *contingency transactions* (see Figure 4.3).

Compensating transactions [GR93] are designed to undo the effect of the original transactions that have aborted. The compensating transactions are triggered and started when the original transactions fail. Otherwise, the compensating transactions are not initiated. Once a compensating transaction has started, it must commit. In other words, the compensating transactions can not abort. If a compensating transaction fails, it will be restarted.

Contingency transactions [Elm+92] are designed to replace the task of the original transactions that have failed. Contingency transactions are also triggered by the failures

of the original transactions. Note that it is not always possible to specify the compensating or contingency transactions for an original transaction.

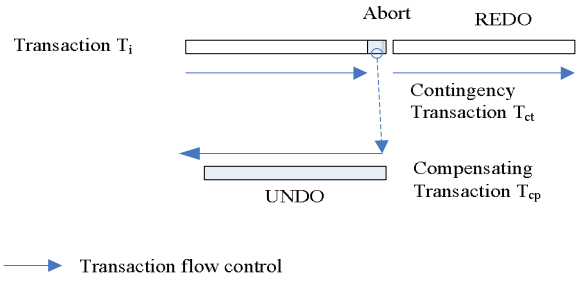

Figure 4.3: Compensating and contingency transactions

Transaction properties. The isolation property is relaxed in multilevel transaction model. The committed results of sub-transactions are visible to other transactions. The atomicity property is ensured by the means of compensating transactions.

Usefulness for mobile environments. The multilevel transaction model is applicable in mobile environments. Multilevel transaction model not only relaxes the isolation property of transactions but also provides a flexible recovery mechanism by the means of the compensating and contingency transactions.

4.2.4 Sagas transaction model

Description. The Sagas transaction model [GMS87] also makes use of the concept of compensating transactions to support transactions whose execution time is long. A Sagas transaction consists of a consecutive chain of flat transactions S_i that can commit independently. For each flat transaction S_i, there is a compensating transaction CP_i that will undo the effect of the transaction S_i if the transaction S_i aborts. A compensating transaction CP_i in the Sagas chain is triggered by the associated transaction S_i or the compensating transaction CP_{i+1}. If the Sagas transaction commits, no compensating transaction CP_i is initiated (see Figure 4.4), otherwise the chain of compensating transactions is triggered (see Figure 4.5).

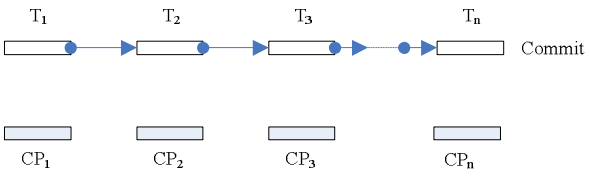

Figure 4.4: A successful Sagas

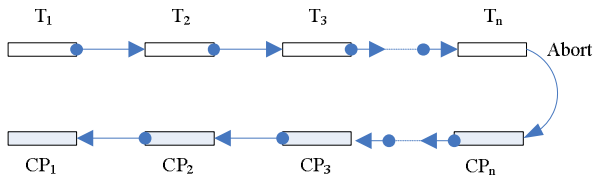

Figure 4.5: An unsuccessful Sagas

Transaction properties. The unit of control of a Sagas transaction is the whole transaction chain. Sagas relaxes the isolation property by allowing component transactions S_i to commit. The atomicity property of Sagas is achieved by the commitment of the last transaction component S_n in the chain or by the backward execution of the compensating transaction chain.

Usefulness for mobile environments. The Sagas transaction model is useful in mobile environments because of its ability for supporting transactions which are long-lived. The isolation property is also compromised. Therefore, the concept can be used to support sharing of data during the execution of mobile transactions. Moreover, it is possible to modify the Sagas model so that we can minimize the losing of useful work when a component transaction S_i aborts, for example by deploying contingency transactions instead of compensating transactions. The main drawback of Sagas is the sequential execution of component transactions in the chain.

4.2.5 Split and Join transaction model

Description. The Split and Join transaction model [PKH88] was proposed to support the open ended activities that associate with transactions. The Split and Join transaction model focuses on activities that have uncertain duration, uncertain developments, and are interactive with other concurrent activities. The main idea is to divide an on-going transaction into two or more serializable transactions, and to merge the results of several transactions together as one atomic unit. In other words, the Split and Join transaction model supports reorganizing the structure of transactions (as illustrated in Figure 4.6).

Transaction properties. The Split and Join transaction model divides the accessed data set of a transaction into different subsets that will be used by newly created and serializable transactions. The goal is to commit part of the original transaction and to make committed results or resources available to other transactions.

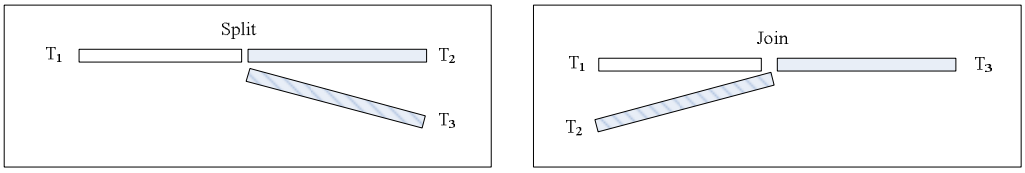

Figure 4.6: Split and Join transaction model

Usefulness for mobile environments. The Split and Join transaction model benefits transactions in mobile environments in terms of dynamic re-structuring of transactions.

4.3 Mobile transaction models

We have reviewed several traditional transaction models whose features are still useful in mobile environments. The traditional transaction models, however, do not have the ability to deal with other challenging requirements of mobile transactions, such as supporting the mobility of transactions and coping with disconnections. Consequently, there are many advanced transaction models that have been developed to particularly support mobile transactions. In this section, we will review several selected mobile transaction models that have the ability to efficiently support mobile transactions. The follows mobile transaction models will be surveyed:

- *Report* and *Co-transaction* model [Chr93]
- *Pro-motion* transaction model [WC99]
- *Two-tier* transaction model [Gra+96]
- *Weak-Strict* transactions model [PB99]
- *Pre-write* transaction model [MB98b, MB01]
- *Pre-serialization* transaction model [DG00]
- *Kangaroo* transaction model [DHB97]
- *Moflex* transaction model [KK00]
- *Adaptable mobile transaction* model (MTS) [Ser02]

For each model, we describe the transaction model and its properties, then we address how the model: (1) handles the *mobility of transactions*, (2) deals with *disconnections*, and (3) supports *distributed transaction execution* among mobile and non-mobile hosts.

4.3.1 Reporting and Co-transaction model

Description. *Reporting* and *Co-transactions* transaction model [Chr93] is based on a two-level nested transaction model (see Figure 4.7). A reporting transaction T_R shares its partial results to top-level transaction S by delegating its operations. The delegation process can happen at any time during the execution of transaction T_R. A co-transaction is a reporting transaction but it cannot continue executing during the delegation process. Thus, the co-transaction behaves as a co-routine, and resumes execution when the delegation process is completed.

Transaction properties. The top-level transaction is the unit of control, and atomic sub-transactions are compensable transactions. A Reporting transaction that is compensatable does not have to delegate all of the committed results to the top-level transaction when it commits. Sub-transactions that are non-compensable delegate all of their operations to the top-level transaction when it commits.

Mobility. The locations of mobile hosts are determined via the identification of mobile support stations. However, the model does not mention explicitly what happens when mobile hosts move from one mobile cell to another.

Figure 4.7: Reporting and Co-transaction

Disconnection. Delegation operations require a tight connectivity between the delegator (i.e., Report and Co-transaction) transactions and the delegatee transaction (i.e., the top-level transaction). Therefore, disconnection is not supported in this model.

Distributed execution. The model supports distributed transaction processing among mobile hosts and fixed hosts where the network connectivity among these hosts is assumed to be available when it is needed.

4.3.2 Pro-motion transaction model

Description. The Pro-motion transaction model [WC99] is a nested transaction model. The Pro-motion model focuses on supporting disconnected transaction processing based on the client-server architecture. Mobile transactions are considered as long and nested transactions where the top-level transaction is executed at fixed hosts, and sub-transactions are executed at mobile hosts. The execution of sub-transactions at mobile hosts is supported by the concept of *compact* objects (see Figure 4.8).

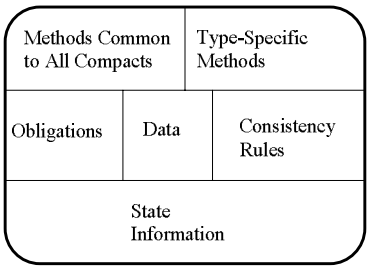

Figure 4.8: Compacts as objects

Compact objects are constructed by compact manager at database servers. Necessary information is encapsulated within a compact object. The compact objects are co-managed by the compact managers (resided at the database servers), the mobility managers (at the mobile support stations), and the compact agents (at the mobile hosts). The compact object plays a role as a contractor that supports data replication and consistency between mobile hosts and database servers. When a mobile host is disconnected, the compact agent takes responsibility for managing all local database operations of mobile transactions at the mobile host. When the mobile host reconnects to database servers, the compact objects are verified against global consistency rules before the locally committed mobile transactions are allowed to commit. Figure 4.9 shows the architecture of the Pro-motion transaction model. Transaction processing consists of four phases: *hoarding, disconnected, connected,* and *resynchronization.* Shared data is downloaded to the mobile host in the hoarding phase. When the mobile host is

disconnected from the fixed host, transactions are disconnectedly executed at the mobile host. If the mobile host connects to the fixed database, the transactions are carried out with the support of the compact manager. When the mobile host reconnects to a fixed host, the results of local transactions are synchronised with the database.

Transaction properties. The Pro-motion transaction model supports ten different levels of isolation. Transactions are allowed to locally commit at mobile hosts; the committed results of these transactions are made available to other local transactions. However, the local committed results must be validated when the mobile hosts reconnect to the database servers. Therefore, the durability property of transaction is only ensured when the transaction results are finally reconciled at the fixed database.

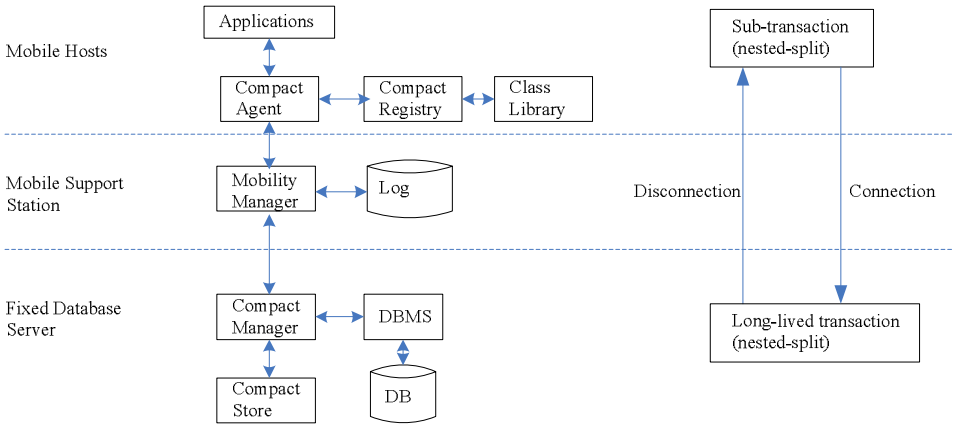

Figure 4.9: Pro-motion transaction architecture

Mobility. Though the mobility manager supports communications between the mobile host and the database servers, how the Pro-motion transaction model supports transaction mobility is not explicitly discussed.

Disconnection. Pro-motion transaction model supports disconnected transaction processing via the support of compact objects. When the mobile host is disconnected from the fixed database, the sub-transactions are split and executed at the mobile host (these split sub-transactions are not joined when the mobile host reconnects to the fixed database). Disconnected transaction processing is a dominant transaction processing mode in Pro-motion even when the mobile hosts are able to connect to the database server. Therefore, the Pro-motion transaction model requires high-capacity mobile resources at the mobile hosts.

Distributed execution. Transactions are mostly executed at mobile hosts and the results are reconciled at the database servers. Therefore, the distributed transaction processing is not strongly supported by the model.

4.3.3 Two-tier transaction model

Description. The two-tier (also called Base-Tentative) transaction model [Gra+96] is based on a data replication scheme. For each data object, there is a master copy and several replicated copies. There are two types of transaction: *Base* and *Tentative*. Base transactions operate on the master copy; while tentative transactions access the replicated copy version. A mobile host can cache either the master or the copy versions of data objects. While the mobile host is disconnected, tentative transactions update replicated versions. When the mobile host reconnects to the database servers, tentative transactions are converted to base transactions that are re-executed on the master copy. If a base transaction does not fulfill an acceptable correctness criterion (which is specified by the application), the associated tentative transaction is aborted. The two-tier transaction model is shown in Figure 4.10.

Transaction properties. Tentative transactions locally commit at the mobile host on replicated copies, and the committed results are made visible to other tentative transactions at that mobile host. The final commitments of those tentative transactions are performed at the database servers.

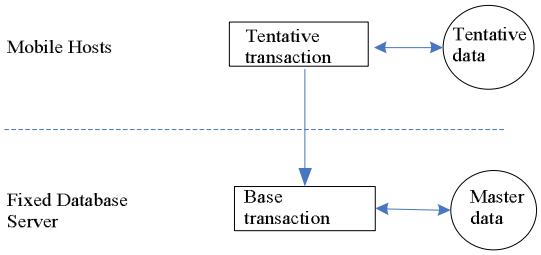

Figure 4.10: Two-tier transaction model

Mobility. Two-tier transaction model does not support the mobility of transactions.

Disconnection. While the mobile hosts are disconnected from the database servers, tentative transactions are locally carried out based on replicated versions of data objects.

Distributed execution. Two distinct transaction execution modes are supported: connected and disconnected. Transactions are tentatively carried out at disconnected mobile hosts, and re-executed as base transactions at the database servers.

4.3.4 Weak-Strict transaction model

Description. The Weak-Strict (also called Clustering) transaction model [PB99] consists of two types of transaction: *weak (or loose)* and *strict*. These transactions are carried out within the *clusters* that are the collection of connected hosts which are connected via high-speed and reliable networks. In each cluster, data that is semantically related is locally replicated. There are two types of a replicated copy: *local consistency (weak)* and *global consistency (strict)*. The weak copy is used when mobile hosts are disconnected or connected via a slow and unreliable network. Weak and Strict transactions access weak

and strict data copies, respectively. Figure 4.11 presents the architecture of this transaction model. When mobile hosts reconnect to database servers, a synchronization process reconciles the changes of the local data version with the global data version.

Transaction properties. Weak transactions are allowed to commit within its cluster, and results are made available to other local weak transactions. When mobile hosts are reconnected, the results of weak transactions are reconciled with the results of strict transactions. If the results of a weak transaction do not conflict with the updates of strict transactions, weak transactions are globally committed; otherwise they are aborted.

Mobility. The concept of transaction migration is proposed to support the mobility of transactions, and to reduce the communication cost. When the mobile host moves and connects to a new mobile support station, parts of the transaction that are executed at the old mobile support stations are moved to the new one. However, no further details about the design or implementation are given.

Disconnection. The Weak-Strict transaction model supports transaction processing in disconnected and weakly connected modes via weak transactions.

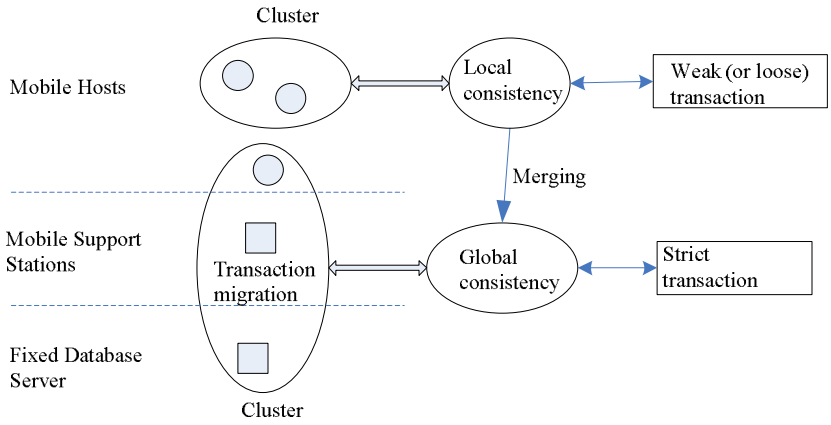

Figure 4.11: Weak-Strict transaction model

Distributed execution. Transaction execution processes can be distributed between the mobile host and the database servers within a cluster that the mobile host participates in. However, the distributed transaction processing among mobile hosts in a cluster is not discussed.

4.3.5 Pre-write transaction model

Description. The Pre-write transaction model [MB98b, MB01] was proposed to increase data availability in mobile environments. Mobile transactions are transactions that are initiated at the mobile host. Pre-write transaction model aims to increase the data availability at mobile hosts. This is achieved by allowing a transaction on a mobile host to submit pre-write operations that write the updated data values, and then issue a pre-

commit state to the mobile support station. After that, the rest of the mobile transaction can be carried out and finally committed at fixed hosts. The small variation, which is specified by the applications, between the pre-committed result and the final committed result is acceptable. Pre-committed data values are accessible to other transactions via pre-read operations. Two different types of lock, which are the *pre-read* and *pre-write*, are introduced to support the new operations. Mobile transactions are not allowed to abort after they have submitted pre-commit operations to the mobile support station. This mobile transaction model can be used to support mobile hosts which have little or no capacity for transaction processing.

Transaction properties. After a mobile transaction submits a pre-commit request, the pre-write values of the mobile transaction are made available to transactions. And the pre-committed mobile transaction is not aborted in any case. The final commitments of mobile transactions will be carried out by fixed hosts. The final committed and the pre-committed data values may not be identical.

Mobility. The roles of the mobile support station are to accept and to process pre-write and pre-commit operations submitted from the mobile host. When moving into a new mobile cell, a mobile transaction connects to the mobile support station in order to submit its pre-write and pre-commit operations.

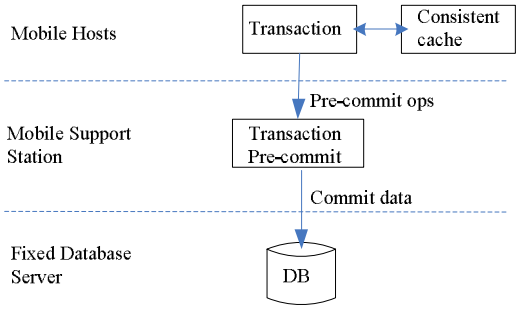

Figure 4.12: Pre-write transaction model

Disconnection. Disconnected transaction processing is supported in the Pre-write transaction model. The mobile transaction is executed at the mobile host until the pre-commit state is reached.

Distributed execution. The major part of the mobile transaction is migrated to the fixed hosts via the mobile support station to be executed there. The mobile host partly takes part in the execution process until the pre-commit states of the mobile transaction are achieved. After this, the mobile host plays no role in the execution of the mobile transaction.

4.3.6 Pre-serialization transaction model

Description. Pre-serialization transaction model [DG00] is built on top of local database systems. Mobile transactions (also called global transactions) are submitted from mobile

hosts through the global transaction coordinators that reside at the mobile support stations. The mobile transaction is entirely processed at local database systems (see Figure 4.13). At each node (or site), there is a site manager that administrates all the transactions executed at that node. When a global transaction is prepared to commit, a global transaction coordinator will carry out an algorithm, called Partial Global Serialization Graph algorithm, that detects any non-serializable schedule among the mobile transactions. If there is a cycle in the graph, i.e., the schedule is non-serializable, the mobile transaction is aborted.

Transaction properties. Each sub-transaction of a global transaction is managed by the local transaction manager. The global serializable graph of transactions is constructed by collecting sub-graphs from the local sites. The atomicity property of the global transaction is relaxed by the concepts of vital and non-vital sub-transactions. If a vital sub-transaction aborts, its parent transaction must abort. However, the parent transaction does not abort if a non-vital sub-transaction aborts. When a sub-transaction commits at the local database system, the results are made visible to other transactions at this local database system.

Mobility. The global transaction coordinators that reside at the mobile support stations support the mobility of mobile transactions. This is done by transferring the global data structure from one global transaction coordinator to another as the mobile host moves from one mobile cell to another.

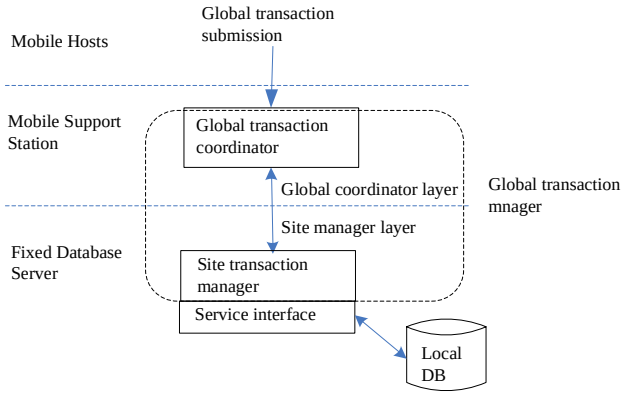

Figure 4.13: Pre-serializable transaction model

Disconnection. Mobile transactions are submitted from a mobile host, and sub-transactions are executed at local database servers. When the mobile host is disconnected, the global transaction is marked as disconnected if the disconnection is known and planned. The execution of the global transaction is still carried out at the local database servers. On the other hand, if the disconnection is unplanned, the global transaction is suspended. The global transaction is resumed when the mobile host reconnects to the mobile support station.

Distributed execution. Mobile transactions are submitted from mobile hosts, and the entire transactions are distributed among local database servers through the support of mobile support stations. The mobile hosts do not take part in the execution processes.

4.3.7 Kangaroo transaction model

Description. The Kangaroo transaction model [DHB97] is designed to capture the movement behavior and the data behavior of transactions when a mobile host moves from one mobile cell to another. This transaction model is built based on the concepts of global and split transactions in a heterogeneous and multi-database environment. The global transaction is split when the mobile host moves from one mobile cell to another, and the split transactions are not joined back to the global transaction. The Kangaroo transaction model assumes that the mobile transactions may start and end at different locations. The characteristics of the Kangaroo transaction model are (see Figure 4.14 for the architecture of Kangaroo transaction model):

- Mobile transactions that include a set of sub-transactions called global and local transactions are initiated by mobile hosts. These mobile transactions are entirely executed at the local database servers that reside on the fixed and wired connected networks.
- The execution of a Kangaroo sub-transaction in each mobile cell is supported by a Joey transaction that operates in the scope of the mobile support station. The Joey transaction plays role of a proxy transaction to support the execution of the sub-transactions of the Kangaroo transaction in the mobile cell.
- The movement of the mobile host from one mobile cell to another is captured by the splitting of the on-going Joey transaction at the old mobile support station and the creating of new Joey transaction at the new mobile support station. The execution of the Joey transaction is supported by the Data Access Agents (DAA) that act as the mobile transaction managers at the mobile support stations.

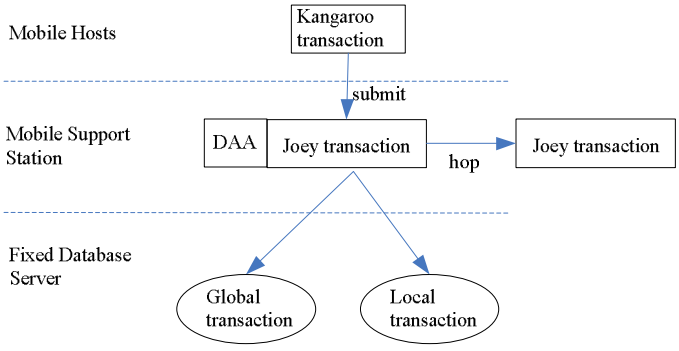

Figure 4.14: Kangaroo transaction model

Transaction properties. The Kangaroo transaction is the basic unit of computation in mobile environments. The serializability of mobile transactions is not guaranteed, and there is no dependency among Joey transactions, i.e., each Joey transaction can commit independently. Two transaction processing modes, which are *compensating* and *split*

modes, are supported by the model. For compensating mode, when a failure occurs, the entire Kangaroo transaction is undone by executing compensating transactions for all those Joey transactions. For split mode, the local DBMS takes responsibility for aborting or committing sub-transactions.

Mobility. The Kangaroo transaction model keeps track of the movement of mobile hosts via the support of the DAA that operates at the mobile support station. In other words, the mobility of mobile hosts is captured on the condition that the mobile hosts always may communicate with the mobile support stations. While mobile hosts move from one mobile cell to another, the hand-off processes are carried out by the DAAs.

Disconnection. Disconnected transaction processing is not considered in Kangaroo transaction model. The processing of Kangaroo transactions is entirely moved to the fixed database servers for executing.

Distribution. The mobile transactions are initiated at the mobile hosts, and entirely executed at fixed hosts. Transaction results are forwarded back to the mobile hosts. The Kangaroo transaction model has shown that the structure of mobile transactions at the specification and execution phases (with the dynamic support of Joey transactions) can be different because of the mobility behavior, i.e., fast or slow movements, of the mobile host.

4.3.8 Moflex transaction model

Description. The Moflex transaction model [KK00] is an extension of the Flex transaction model [Elm+90] to support mobile transactions. The Moflex model is built on top of multi-database systems and based on the concepts of split-join transactions. The main characteristics of a Moflex transaction are:

- A Moflex transaction that consists of compensable or non-compensable sub-transactions is initiated by the mobile host. These sub-transactions are submitted to the mobile transaction manager (MTM) that resides at the mobile support station. The MTM will send these sub-transactions to the local execution monitor (LEM) at local database systems for executing. Figure 4.15 presents the architecture of Moflex transaction model.
- Each Moflex transaction *T* is accompanied by a set of success and failure transaction dependency rules, hand-over control rules (see Table 4.1), and acceptable goal states. Dependent factors that include the execution time, cost and execution location of transactions are also specified in the definition of the Moflex transaction. Furthermore, joining rules are provided to support the join of the split sub-transactions (sub-transactions are split when the mobile host moves from one mobile cell to another).

Transaction properties. The mobile transaction managers make use of the two-phase commit protocol to coordinate the commitment of the Moflex transaction. The Moflex transaction commits when its sub-transactions that are managed by MTM have reached one of the acceptable goal states, otherwise it is aborted. A compensable sub-transaction

is locally committed, and the results are made visible to other transactions. For non-compensable sub-transactions, the last mobile transaction manger, which corresponds to the end location of the mobile host, plays the role as the committing coordinator.

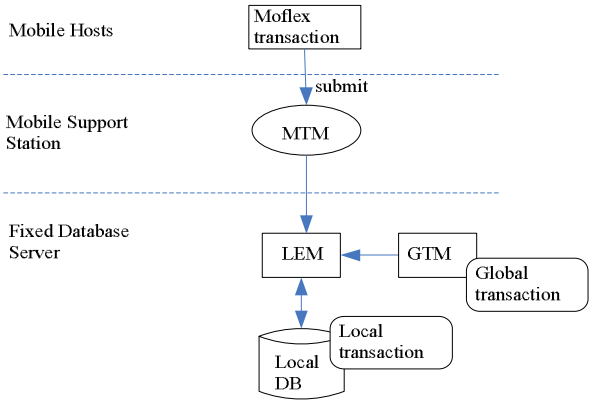

Figure 4.15: Moflex transaction model

Mobility. The mobility of transactions is handled by splitting the sub-transaction, which is executed on the local database at the current mobile cell, as the mobile host moves from one mobile support station to another (with the support of the mobile transaction manager). Hand-over control rules must be specified for each sub-transaction (see Table 4.2). If a sub-transaction is compensable and location independent, it will be split into two transactions; one will continue and commit at the current local database, the second will be resumed at the new location. If the sub-transaction is location dependent, at the new location, the sub-transaction must be restarted. If a sub-transaction is non-compensable, the sub-transaction is either restarted as a new one in the mobile cell if it is location dependent, or continued if it does not depend on the location of the mobile host.

Table 4.1: Hand-over control rules of sub-transactions

	Compensable	**Non-compensable**
Location-independent	split_resume	continue
Location-dependent	restart, split_restart	restart

Disconnection. Moflex transaction model does not support disconnected transaction processing. The Moflex transaction model requires network connectivity between the mobile host and the mobile support stations during the execution process.

Distributed execution. The execution of a Moflex transaction is transferred to local database systems at fixed hosts to be carried out there. Moflex transaction model provides a framework to specify the execution of transactions in mobile environments. The main drawback of the Molex transaction model is that the specification of mobile transactions must be fully specified in advance, therefore, the Moflex transaction model may not have the capacity to deal with un-expected or un-planned situations.

4.3.9 Adaptable mobile transaction model

Description. An adaptable mobile transaction model and a mobile transaction service (MTS) [Ser02] are proposed to support the adaptability of mobile transaction execution. The MTS architecture is a three-tier client/agent/server one in which the clients are mobile hosts, the agents reside at mobile support stations, and the servers are fixed database servers (see Figure 4.16). The main goal of the MTS is to adapt the transaction execution to different environment conditions. The adaptive mobile transaction consists of component transactions T_i, compensating transactions CT_i and the execution strategy *ES*. The execution strategy is a list of execution alternatives comprised of environment descriptors *ED* and component transactions. Changes of environment conditions are captured via an event notification service.

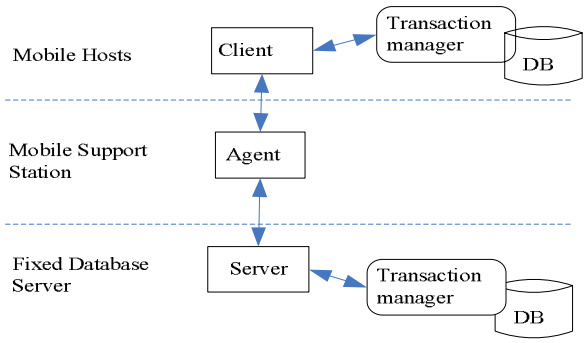

Figure 4.16: The architecture of the MTS

Transaction properties. Only one execution alternative of the adaptive mobile transaction is executed at any moment. The component transactions are ACID transactions which can belong to one or more execution alternatives. Changes in the execution alternatives may result in the abortion of the component transactions. If a component transaction belongs to different execution alternatives, the component transaction is continued with the new execution alternative.

Mobility. The mobility of transactions is not defined by the adaptable mobile transaction. However, the hand-off process is treated as a change of environment conditions via an *e-hand-off* event.

Disconnection. The disconnected processing of mobile transactions is specified in execution alternatives, and is applied when an *e-disconnection* event occurs.

Distributed execution. The mobile transaction service defines five different execution modes (see Table 4.2) that specify how a mobile transaction could be executed among the fixed database servers and the mobile hosts.

The adaptable mobile transaction takes into account dynamic changes of mobile environments, and supports different execution alternatives in accordance with the

environment conditions. The main disadvantage of the model is that the execution alternatives must be specified in advance.

Table 4.2: Execution models of adaptive mobile transaction

Modes	Distributed execution
1	Entirely at database servers
2	Entirely at the mobile host
3	At one mobile host and several DB
4	At several mobile hosts
5	At several mobile hosts and DBs

4.4 Issues related to mobile transaction processing systems

In this section, we discuss issues that are related to mobile transaction processing systems. The three issues are: mobile database replication, advanced transaction commitment protocols, and mobile data sharing mechanisms. These three issues contribute in a vital way to the performance of transaction processes in mobile environments.

4.4.1 Mobile database replication

In mobile environments, to cope with the disconnections of the wireless networks, the mobile hosts must be able to cache necessary data to support disconnected transaction processing. A database portion that is cached at a mobile host is called the mobile database [HAA02]. Mobile databases offer higher level of data availability at disconnected mobile hosts; thus, enhance the performance of mobile transaction processing systems. Figure 4.17 illustrates an example of the life cycle of mobile databases. Before the mobile hosts are disconnected from the database servers, shared data is cached at the mobile host. When the mobile host is disconnected from the database servers, cached data is modified. When the mobile hosts reconnect to the database server, shared data that has been modified at the local cache will be reconciled with the original versions.

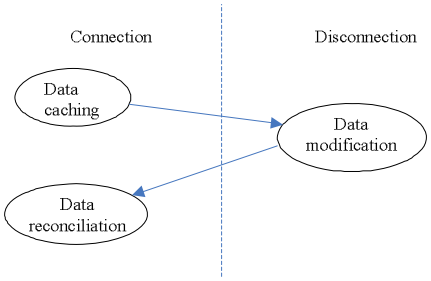

Figure 4.17: Life cycle of mobile databases

Keeping data consistency among these copies all the time is difficult. Therefore, the main issue is how to avoid or be aware of data inconsistency among such copies. This can be achieved by several ways, for example an advanced locking protocol [MB01] or sign-off/check-in/check-out operations [HAA02]. The pre-write lock [MB01] is an additional lock layer that is deployed at the mobile support station to support mobile transactions to access shared data, i.e., without connecting to the database server. Transactions can connect to either the database server or the mobile support station to access shared data. If a shared data is modified, it will first be stored at the mobile support station before being updated in the database server. For sign-off/check-in/check-out operations [HAA02], consistent shared data is downloaded from the database server to mobile hosts via the support of a proxy-transaction (called a pseudo-transaction) to support disconnected transaction processing. When the mobile hosts reconnect to the database server, mobile transactions will be checked to ensure that they are serializable with other transactions at the database server.

The mobile databases must be able to support mobile hosts to cope with different types of disconnections. There are two forms of disconnection: *planned* and *unplanned*. For planned disconnections, the mobile hosts inform the database servers about the disconnections so that the mobile databases can be well prepared. The strict mobile database replication model uses the standard shared and exclusive lock modes (see Table 2.1) for controlling conflicting database operations among replicated copies. Relaxed mobile database replication model allows transactions to concurrently access replicated database portions at different mobile hosts as long as there is an acceptable execution schedule among involved transactions. For example, check-out with mobile read, check-out with system read, or relaxed check-out modes [HAA02]. To our knowledge, there is no mobile database model that supports mobile databases to deal with unplanned disconnections (which will be dealt with in our mobile transaction processing system).

4.4.2 Advanced transaction commitment protocols

The standard 2PC protocol [Esw+76] may not be appropriate in mobile environments because it is a possibly blocking protocol and requires many messages. There are more advanced transaction commitment protocols that have been proposed.

The Timeouts Protocol is proposed in [Kum+02]. The transaction coordinator that resides at the mobile support station will decide to commit or abort transactions based on a *timeout* value. The timeout value is the total of *execution timeout* and *shipping timeout.* A mobile transaction will be allowed to commit if all of the updates of sub-transactions are received by the coordinator before the timeout value is expired; otherwise the transaction is aborted. The timeout commit protocol requires that mobile hosts always connect to the mobile support station and the database servers. The main drawback of this protocol is that it is hard to define or estimate the *execution* and *shipping* timeout values in mobile environments.

The Unilateral Commit Protocol [AC04] is proposed to support transaction commitment in disconnected mode. This protocol reduces the number of exchanged messages by

removing the second voting phase of the standard 2PC protocol (thus, this protocol is also called a one-phase commit protocol). If a mobile transaction reaches the prepare-to-commit phase, it will commit. There are other commit protocols that are developed by taking into account the special characteristics of mobile transactions. Some examples are the commitment of read-only transactions [GW82] that are carried out separately from updating transactions [KLH03, CLL03, LLK01] (by exploring the special consistency requirements of read-only transactions), and the pre-commit protocol [MB01] (by tolerating the difference between pre-committed and committed results which is specified by applications).

4.4.3 Mobile data sharing mechanisms

In this section, we address the mechanisms that support sharing of data in mobile environments. In general, shared data is stored at dedicated non-mobile database servers. Mobile hosts need to connect to these database servers to access shared data. However, due to the disconnections in communication, the mobile hosts may not always be able to connect to the database servers. This leads to the demand of a temporary sharing workspace that is stored at dedicated hosts. Existing models that have been designed to support sharing data in distributed environments, for example the common-local workspaces model [Ram01] or the sharing tuple space [PMR00], will not be suitable for mobile environments due to the static configuration and the lack of mobile and dynamic workgroup supports.

Recently, there are many research proposals that focus on supporting data sharing among mobile hosts in mobile environments. The two essential components that contribute to the mobile data sharing are: (1) the dynamic mobile workgroup management, and (2) the data access mechanisms. The dynamic mobile workgroup management [BCM05] focuses on the organization and management of temporary mobile workgroups that are the collection of mobile hosts. The data access mechanisms are based on either the client-server [BF03] or the peer-to-peer [PMR00] architecture. The Accessing Mobile Database (AMDB) architecture [BF03] is based on the concepts of mobile agents and the client-server model. The main idea is to form a Mobile Database Community (MDBC) in which mobile clients access mobile databases that are stored at dedicated mobile hosts. The LIME (Linda in Mobile Environments) [PMR00] architecture makes use of mobile agent technology to support sharing of data among different mobile hosts.

4.5 Survey of commercial products

In section 4.3, we have reviewed several mobile transaction models that are mostly used for academic research purposes. There is little information about how these mobile transaction models are deployed in real application products. In this section, we review mobile transaction processing in commercial products. The following products are surveyed: Microsoft SQL Server CE [Mic], Oracle Lite [Ora], and IBM DB2 Everyplace [IBM]. We focus on describing in detail how these commercial products support mobile transactions and how data consistency is achieved.

4.5.1 Microsoft SQL Server CE

Description. The Microsoft SQL Server CE (SSCE) [Mic] is a client-agent-server architecture that supports database applications on mobile hosts (see Figure 4.18). The database on a mobile host is a small replicated portion of the main database. When mobile hosts are disconnected, transactions are processed locally at the mobile hosts. When mobile hosts reconnect to the database server, synchronization processes are carried out to reconcile information. The client agent at the mobile host connects to the server agent through the Internet Information Server (IIS) that resides on the database server. This means that the role of mobile support stations is not an issue in SSCE systems.

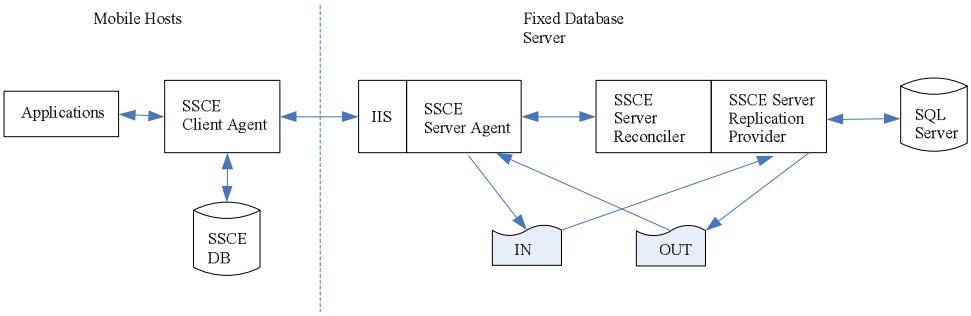

Figure 4.18: Microsoft SQL CE architecture

Transaction properties. The Microsoft SQL Server CE supports both flat and nested transactions at mobile hosts. Sub-transactions only reveal committed results to the parent transaction. When the top-level transaction commits, the results are visible to local transactions at the mobile host. Transactions at mobile hosts are executed sequentially.

Data consistency. When the mobile host reconnects to the database server, a synchronization process is performed to reconcile information. The client agent sends all changes in the local database to the server agent. The server agent, then, writes the updates to a new input file and initiates a reconciliation process at the SQL Server Reconciler. The reconciliation process will detect and resolve conflicts. Different conflict resolutions are supported in the SSCE system, for example priority based or user defined. When the reconciliation process completes, it will inform the SQL Server Replication Provider to finally write the successful updates to the database server. When there are updates at the database server, an inverse process is carried out to propagate these updates to the mobile host.

4.5.2 Oracle Lite

Description. Oracle Lite [Ora] is a client-server architecture that makes use of a replicated copy of the main database (which is called a snapshot) to support disconnected transaction processing at mobile hosts (see Figure 4.19). Oracle Lite does not include mobile support stations in its architecture. The replicated database system at the mobile host is called a *snapshot* that can be read-only or updatable. When the mobile host is

disconnected from the database server, transactions are processed locally. The snapshot is synchronized with the master copy at the database server when the mobile host reconnects.

Transaction properties. The Oracle Lite only supports flat transactions at mobile hosts.

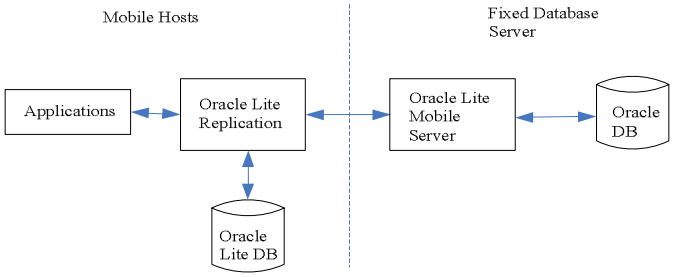

Figure 4.19: Oracle Lite architecture

Data consistency. When the mobile host connects to the database server, a refresh process will be performed to synchronize the snapshot with the master copy. If the snapshot is modified, the updates will be sent to the database server. All local transactions at the mobile host will be validated at the database server in the same order as they were executed at the mobile host. The refresh process is a blocking process. This means that no database operations will be allowed at the mobile host during the reconciliation process.

4.5.3 IBM DB2 Everyplace

Description. IBM DB2 Everyplace [IBM] is an architecture that consists of a relational database at mobile hosts and a *mid-tier* on fixed hosts. The mid-tier system supports data synchronization between the mobile databases that reside at the mobile hosts with the source databases on the fixed database servers. When mobile hosts are disconnected, transactions are processed locally at the mobile hosts. When mobile hosts reconnect to the database server, synchronization processes are carried out to reconcile data. As Microsoft SQL Server CE and Oracle Lite, IBM DB2 Everyplace does not discuss mobile support stations. Data synchronization processes are carried out directly between the mobile hosts and the fixed database servers.

Transaction properties. The IBM DB2 Everyplace only supports flat transactions.

Data consistency. When the mobile host connects to the database server, a synchronization process will be performed to synchronize data between the mobile hosts and the source database. IBM DB2 Everyplace differentiates the data synchronization processes between the mobile host and the source database. The data synchronization from the mobile host to the source database is illustrated in Figure 4.20. The synchronization request is submitted from the mobile host and placed in the input queue at the fixed database server. If the synchronization request is allowed to proceed, the data at the mobile host is temporarily saved in the Staging table then the Mirror table. If there

is any conflict, it will be resolved in the Mirror table. After this, the changes are stored in the DB2 log and sent to the source database through a Change Data table. For the data synchronization from the source database to the mobile host, an inverse process is performed (as illustrated in Figure 4.21). The main difference between these two data synchronization processes is that the data from the source database is immediately processed and transferred to the mobile host without any delay, i.e., without passing through the Staging table and Administration control.

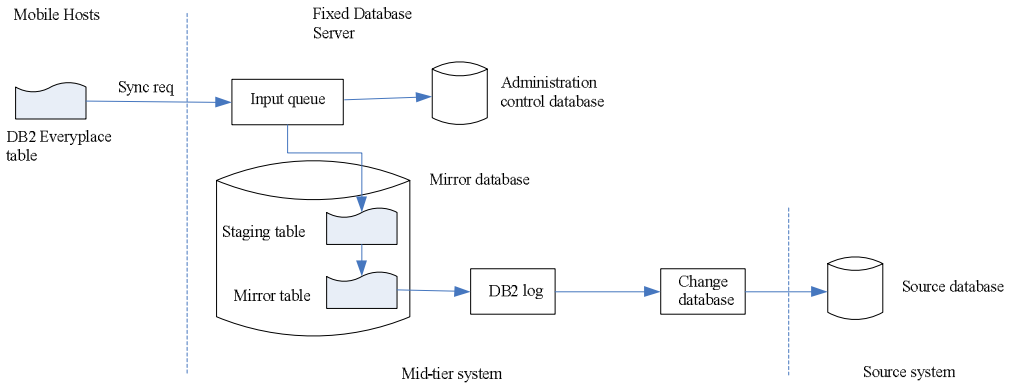

Figure 4.20: IBM DB2 Synchronize from mobile hosts to fixed hosts

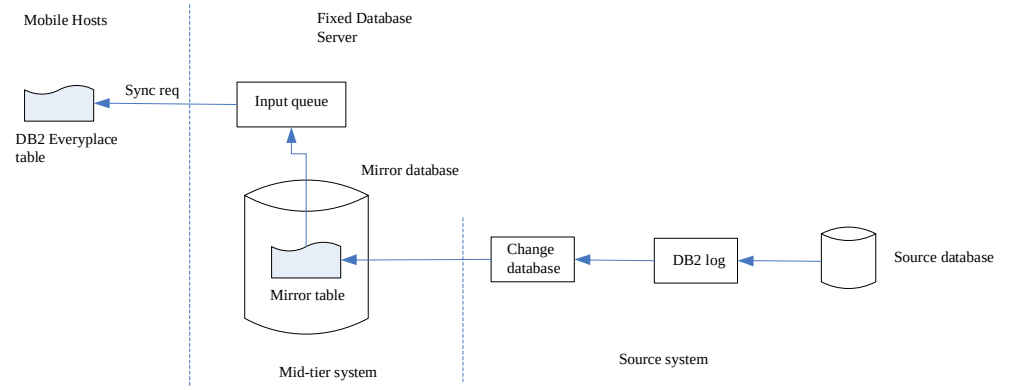

Figure 4.21: IBM DB2 Synchronize from fixed hosts to mobile hosts

4.6 Conclusions

In this chapter, we have reviewed several traditional transaction models that were developed to support transaction processing in centralized and distributed environments. These transaction models still benefit transactions in mobile environments in term of customized isolation property (e.g., Multi-level and Sagas transactions), and dynamic structure (e.g., Split and Join transactions). For dealing with other challenging requirements like mobility and disconnections, a number of advanced mobile transaction models have also been developed. The more general characteristics of these mobile transaction models are:

- Mobile transaction models are developed based on the concepts of nested and split-join transaction models. These models have the ability to relax the atomicity and isolation properties. The commitment of mobile transactions consists of two states: (1) local commit at the mobile hosts, and (2) final commit at the database servers. When a mobile transaction commits at a disconnected mobile host, its committed results are made available to other local transactions at the same mobile host. When the mobile host reconnects to the database server, these results of local transactions will be validated against the ones at the database server. If there is any inconsistency, some of the locally committed transactions are aborted.

- In order to capture the mobility of mobile transactions, when the mobile hosts move from one mobile cell to another, the mobile hosts must be able to connect to the mobile support stations of these mobile cells. Hand-off or hand-over processes are performed to transfer the transaction controls from one mobile support station to another.

- To cope with the limited computing capacity of mobile hosts, a part of or an entire mobile transaction that is initiated by a mobile host, is moved to fixed database servers for processing.

A part from these features, there are still major limitations:

- The lack of some fundamental support for mobile transactions is an issue. There are different views what a mobile transaction is. Many models consider mobile transactions as transactions that are submitted to or initiated from the mobile hosts [DHB97, KK00]. Other models require that mobile hosts must take part in the execution of mobile transactions [MB01]. These different attitudes cause incompatibility and incoherence between mobile transaction processing systems.

- The common architecture of mobile transaction environments relies heavily on the mobile support stations that are stationary and wired connected with the database servers. A difficulty is to extend the capacity of mobile transaction processing systems. For example, the bottleneck problem can occur when there are many mobile hosts within a mobile cell (one IEEE 802.11 WAP can support thirty wireless client systems within a radius of 100 meters[5]); and the distribution of the transaction processes among mobile hosts must be carried out through the mobile support stations.

- Sharing partial results among mobile transactions is not fully dealt with. The existing approaches like delegation operations [Chr93, Ram01] that support sharing of data among transactions may not be adequate because it requires a tight cooperation between delegator and delegatee transactions. Furthermore, the issue of distributed transaction execution among mobile hosts [SRA04] has not been addressed.

[5] Source http://www.wifiguide.org/

There is also a big gap between academic research and commercial products on mobile transactions. In academic research, the mobile support stations play very important roles in the processing of mobile transactions. While in commercial products, mobile hosts and database servers communicate directly, i.e., the role of the mobile support stations does not exist. Moreover, commercial products mainly focus on disconnected transaction processing, while the mobility of mobile hosts is not taken into consideration.

PART II

CONCEPTS, MODELS and FORMALIZATION

Chapter 5
Mobile Transaction Processing System: Concepts and Models

This chapter presents a method of approach and fundamental concepts of our mobile transaction processing system. The main focus is to support sharing of data and database operations among mobile transactions at different mobile hosts in mobile environments. This is achieved by the adaptable mobile data sharing mechanism via the support of *export* and *import* transactions, which operate in a mobile sharing workspace, called the *export-import repository,* which belongs to a temporary and dynamic workgroup of mobile hosts, named the *mobile affiliation workgroup*.

5.1 Introduction

In Chapter 4, we have reviewed several mobile transaction models that have been developed to support transaction processing in mobile environments. We also discussed the limitations of these mobile transaction models. The main disadvantage is the lack of adaptable support for mobile transactions to continue or to adjust their operations to different operating conditions. For example, the architecture of the mobile transaction environment requires that in order to contact other mobile hosts or database servers, a mobile host must connect to the mobile support station of the mobile cell in which the mobile host currently resides [SRA04]. In other words, in this restricted architecture of the mobile transaction environment, if a mobile host is not able to connect to a mobile support station, the mobile host has no means to interact with other hosts, and therefore on-going transactions at this mobile host may not be carried out.

Furthermore, the advantages of mobile computing devices and communication technologies are not fully exploited by the existing mobile transaction models. For example, the ability of wireless networks that support nearby and peer-to-peer communication among mobile hosts has not been taken into consideration. If this ability had been taken into account, it is possible to support the distributed transaction execution among mobile hosts. Furthermore, this new communication technology can also be used to enhance the data availability in mobile environments. For example, in stead of connecting to the database servers (via the mobile support stations) to obtain necessary

information, a mobile host can contact other nearby mobile hosts for replicated information.

In order to sufficiently and efficiently support a mobile transaction processing system, we must take into account not only all the challenging characteristics of mobile environments (see Section 3.5), but also the advanced mobile technologies. For example, to cope with disconnections in communication, the mobile transaction processing system must be able to support asynchronous, non-blocking and presumable interactive operations. Sharing of data or database operations must be carried out in accordance with the availability of wireless network resources. For example, a large chunk of shared data must be divided into a set of smaller chunks for transmitting when the wireless bandwidth is low and the connection time is short. The usage of mobile computing resources and the mobility behavior of the mobile hosts must also be taken into consideration. For example, a mobile host that has a large storage capacity should be configured to play a role as a temporary mobile proxy server to other nearby mobile hosts.

In this chapter, we present our method of approach and fundamental concepts that lead to the development of our mobile transaction processing system. The main objective is to develop a versatile mechanism to support the sharing of data and database operations among transactions at different mobile hosts. This is achieved by allowing mobile hosts to form temporary and dynamic workgroups, called the *mobile affiliation workgroups*, by taking advantage of wireless communication technologies, i.e., the ability of direct communication among mobile hosts within a limited range. For example, two mobile hosts can directly exchange data by using Bluetooth or wireless USB technologies. The sharing of data and database operations among transactions at different mobile hosts is carried out by the means of *export* and *import* transactions through a mobile sharing workspace, called the *export-import repository*, which belongs to a mobile affiliation workgroup.

This chapter is organized as follows. In Section 5.2, we illustrate our method of approach via a motivating mobile IT (*I*nformation *T*echnology) support scenario and discuss several interesting observations. This leads to a new mobile collaborative work model for mobile environments called *horizontal collaboration* that is introduced in Section 5.3. The concepts and model of the mobile affiliation workgroup, the export-import sharing workspaces as well as the export and import transactions are also discussed in this section. Section 5.4 addresses the properties of two different types of mobile transactions, called *shared* and *standard* transactions. Section 5.5 focuses on the mobile data consistency and the mobile data sharing mechanism. The issues related to the management of mobile sharing workspaces and the management of transaction execution behavior are discussed in Section 5.6 and 5.7 respectively. Finally, the important contributions of our mobile transaction processing system are summarized in Section 5.8.

5.2 Extending the support for mobile collaborative works

Mobile environments change the way in which people carry out their works. The environment for accessing and processing information is changing rapidly from

stationary to mobile and location independent. This leads to the demand for new organization and management models to support collaborative work in mobile environments. In this section, we discuss and analyze the characteristics of a mobile IT-support scenario. We also present several interesting observations that lead a new mobile collaborative work model called *horizontal collaboration* (presented in Section 5.3.1).

5.2.1 Motivating scenario

In the following, we discuss a mobile IT support scenario that has been studied thoroughly in our MOWAHS project [Sør+02, Ram+03, Sør+05] (this mobile IT support scenario was also briefly presented in Section 1.1). The mobile IT support scenario (see Figure 5.1) will be used as to exemplify our mobile transaction processing system.

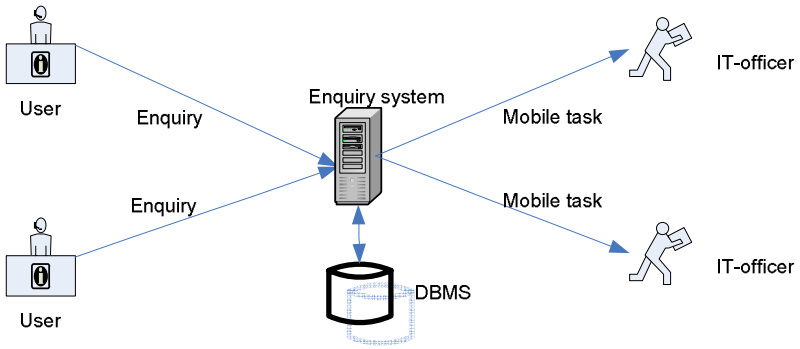

Figure 5.1: Mobile IT-support scenario

The mobile IT support system is a mobile collaborative support system in which IT officers work and collaborate to help users dealing with computing problems. The IT officers are equipped with mobile computers, and handle requests from users at different locations. The goal is to solve as many computer problems as quickly as possible. When a user encounters a computer problem, he or she will send a description of the problem to an enquiry system that consists of distributed database servers. The submitted enquiries from users may or may not fully describe the problem. This problem description is called an enquiry, and will be stored in the database servers. Each newly arrived enquiry will be assigned a state named *new* (see Figure 5.2). The IT officers regularly check the enquiry database for unsolved problems. An IT officer can self-select or be assigned (by the system administrator) an enquiry to work on. When a problem is selected to be solved, its state is changed to *active*, and is called a *mobile task*. The IT officer who takes the responsibility for a mobile task can contact the users who submitted the enquiry for further details; or other officers for additional consultations and discussions about the problem. When a mobile task is solved, it is saved in a *complete* state for future references.

To avoid work collision among IT technicians, one mobile task is allocated to one IT officer at any time. However, an IT officer can be assigned many mobile tasks. Furthermore, to prevent conflicts among database operations of mobile tasks that could

concurrently manipulate shared data, a part of or an entire mobile task must be covered by a transaction.

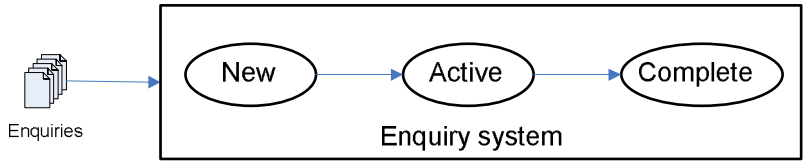

Figure 5.2: States of mobile tasks

The characteristics of the mobile tasks that require transactional support are summarized in Table 5.1.

Table 5.1: Mobile task characteristics requiring transactional support

Characteristics	Descriptions
Pre-planned	To what degree is the mobile task planned beforehand?
Data synchronization	When is the updated data of the mobile task synchronized with other tasks?
Data exchange rate	How often will the mobile task exchange data with other tasks within its lifetime?
Event-triggered	Is the mobile task triggered by an event?
Task resumption	Can the mobile task be halted for later to be resumed from where it left off without restart?
Task lifetime	What is the expected lifetime of the mobile task?
Location constraint	Is the mobile task executed at a specific location?
Time constraint	Is the mobile task executed at a specific time?
Temporary coordination	Is the mobile task timed with other activities?

The above characteristics of the mobile tasks are as follows:

- The *pre-planned* characteristic describes to what degree a mobile task is planned beforehand. A mobile task can be well-planned in detail or partially planned before being executed. In some extreme cases, a mobile task can not be planned at all. For example, in the mobile IT support scenario, the pre-planned characteristic of a mobile task depends on the knowledge of the user who submits the enquiry. A mobile task can be well pre-planned if it is described in detail, for example the yellow cartridge of a laser printer must be replaced. If the cause of a computer problem is not clear or a user has little knowledge about it, the description of the problem can be more general, for example a wireless connection in the lecture theatre has failed. Consequently, this mobile task is partially pre-planned.

- The *data synchronization* specifies when a mobile task has to synchronize or merge the updated data with other tasks. For a simple and short mobile task, data synchronization is not necessary or not required. However, a complicated and long mobile task can require data synchronization during its execution process. For example, a mobile task that installs a client application at a remote computer requires data synchronization with the server application in order to obtain the operational license.

- The *data exchange rate* specifies how often the data exchange between the current mobile task and other tasks takes place. During its execution process, a mobile task can require one or many interactions with other tasks. For example, a mobile task that installs an operating system at a remote computer demands many upgrading or bug fixing phases.

- The *event-triggered* characteristic decides whether a mobile task is triggered by an event or not. The execution of a mobile task can be affected when a resource or a service becomes accessible or inaccessible. The triggering event can cause re-scheduling or re-planning of the mobile task. For example, a network disconnection event causes the upgrading process of an application to be aborted or re-scheduled at later time; and the suspended process can resume executing when a connection event occurs.

- The *task resumption* characteristic describes if a mobile task can halt, and then later resume from where it left off, i.e., it is not required that the mobile host must completely restart. For example, a mobile task that upgrades a client application via wireless networks can be suspended if a network disconnection occurs. This mobile task can resume executing when the network connectivity becomes available. On the other hand, a mobile task may not have the ability to resume executing, i.e., this mobile task must always begin from scratch. For example, a network security scanning task must always start freshly to avoid missing any malicious bug. For mobile tasks that can be resumed at some specific states, additional services are required, for example check-point or logging services.

- The *task lifetime* describes the expected lifetime of a mobile task. If a mobile task is simple and well planned in advance, it is possible to estimate an approximate execution time. On the other hand, the task lifetime of a difficult mobile task can not be accurately estimated. For example, changing the ink cartridge of a printer can take minutes to complete; however, configuring a network service could take several hours.

- The *location constraint* specifies to what degree a mobile task must strictly follow a specific travel route or be executed at a specific location. For example, an IT technician must be in a specific room to repair a network connection. The location constraint characteristic also affects the pre-planned characteristic of the mobile task, for example the travel route must be well planned beforehand.

- The *time constraint* describes if a mobile task must be executed at a specific time or within a specific time interval. For example, a storage service upgrading task must be performed between 19:00 hours and 21:00 hours to avoid interrupting normal everyday work of employees. Those mobile tasks that must follow a time constraint must also be carefully planned.

- The *temporary coordination* identifies if a mobile task must be coordinated with other tasks. The temporary coordination characteristic has a strong impact on the execution of related mobile tasks. For example, a mobile task that replaces a network router of a wired network must be strictly executed after a re-direct router configuration task has been completed in order to avoid losing network connections. If the re-configuration of the routing table is not carried out as planned, the router replacement task will be delayed.

5.2.2 Interesting observations

In this section, we discuss several interesting observations of the mobile IT-support scenario in order to illustrate how we shall develop a mobile transaction processing system that meets all the requirements described in Chapter 3. These observations are not only applicable to this mobile IT-support scenario, but also applicable to other mobile work applications such as traveling salesmen, mobile learning and report production [Ram+03].

Observation 1: Encourage mobile works without support from database servers or mobile support stations

IT officers work in a mobile environment, and use wireless networks to communicate with the database servers and other IT officers. While working in the mobile environment, IT officers may have to travel to different locations to fix computer problems. The mobile IT support system must have the ability to support the movement of the IT officers so that their activities will not be disrupted. This means that requirement R1 - *the mobile transaction processing system must be able to effectively handle the hand-over control of mobile transactions* – must be fulfilled.

Furthermore, while working in mobile environments, IT officers can experience long disconnection periods, for example when they are working in a location in which the wireless network services are not available. The mobile IT support system must have the capacity to support the IT officers to continue carrying out the work while being disconnected from the database server for a long period of time. This means that requirement R3 - *the mobile transaction processing system must support disconnected transaction processing* – must be fulfilled.

Furthermore, when an IT officer completes a mobile task, the states of the mobile task will be temporarily saved at the mobile computer, and must be archived in the database servers later. This means that the mobile IT support system must provide a mechanism to safely record the states of a mobile task. In other words, requirement R7 - *the mobile*

transaction processing system must assure the durability property of transactions - must be fulfilled.

Due to the disconnections of wireless networks and the constraint of mobile computing resources, an on-going mobile task can be disrupted or suspended. In order to support the recovery of the mobile task when the wireless networks or mobile resources become available, the mobile IT support system must provide a mechanism to record the previous activities of the mobile task. This means that requirement R9 - *the mobile transaction processing system must support temporary data and transaction management* – must be fulfilled. Moreover, the temporary data and transaction management also supports IT officers to know which activities have been carried out or what data has been modified while they are disconnected from the database servers.

Observation 2: Cultivate additional support among co-mobile workers

While working on a mobile task, an IT officer could experience unplanned disconnections in communication. For example, the IT officer may be outside the area covered by the wireless networks, or may be moving behind shadowing objects like buildings. In these situations, the IT officer will not be able to contact the database servers, and the mobile work will be interrupted. However, the IT officer can communicate with other nearby mobile workers, i.e., within a limited communication range, via ad hoc wireless networks, for example Bluetooth or wireless USB. This way the IT officer can ask for support from other nearby workers. For example, an IT officer who is fixing a printer problem can ask for an electronic version of the printer manual which is available from a nearby colleague. In order to support collaborative work in this situation, the mobile IT support system must support interactions among nearby mobile hosts. This means that requirement R2 - *the mobile transaction processing system must support interactions among transactions at different mobile hosts* – must be fulfilled. To achieve this, our mobile transaction processing system allows disconnected mobile hosts to form temporary and dynamic workgroups, called mobile affiliation workgroups (see Section 5.3.2), so that they can continue carrying out collaborative operations while being on the move and disconnected from the database servers.

A mobile host can, at the same time, be able to connect to a mobile support station via a wireless LAN connection and to other nearby mobile hosts via short-range wireless technologies. Therefore, this mobile host can be dynamically configured to play the role of an additional mobile support station to other mobile hosts. It can act as a mobile relay host or a temporary mobile database server to support other mobile hosts that are currently unable to directly connect the mobile support station. In other words, the mobile IT support system must fulfill requirement R4 - *the mobile transaction processing system must support distributed transaction execution among mobile hosts and stationary hosts*. This way the mobile transaction processing system can cope with the limited computing capacity of mobile hosts, and avoid relying heavily on the support from mobile support stations.

Observation 3: Demand an adjustable collaborative work technique

Due to the complexity and difficulty of a mobile task, it may take longer time and more effort to carry out the mobile task. While being carried out, the mobile task can suffer from disruptions or failures, for example a mobile computer is running out of battery energy or parts of the mobile work are cancelled. Therefore, the mobile IT support system must provide a mechanism to prevent losing useful work that has been done, for example rolling back previously achieved parts. This means that requirement R5 - *the mobile transaction processing system must have the ability to customise the atomicity property of transactions* – must be fulfilled.

Additionally, the mobile IT support system must also support the recovery of a mobile task that has been affected by disruptions, i.e., providing the ability to adjust and continue from previously disrupted points. For example, a disconnected IT officer must be able to recover from a previously disconnected state when the communication channel is re-established at a later time, or part of the mobile task must be changed to be consistent with other parts. This means that requirement R8 - *the mobile transaction processing system must provide efficient recovery strategies* – must be fulfilled.

Furthermore, a mobile task may not always be carried out as planned. This can happen when the mobile task is complicated and requires more collaborative support from several IT officers. For example, an IT officer who currently works on a difficult mobile task should allow other IT officers the opportunity to share their expertise in the problem or to take over the task. This means that requirement R6 - *the mobile transaction processing system must support sharing partial states and status among transactions* – must be fulfilled. This way the problem has a higher chance of being solved in the shortest possible time, i.e., achieving higher throughput for mobile works. Note that in volatile mobile environments, the existing mechanisms that support sharing of data among transactions, for example altruistic locking protocols [SGS94], delegation operations [CR94, Ram01], or prewrite locking protocols [MB01] might not be adequate. This is due to two reasons: (1) these mechanisms require a tight cooperation among the participants, and (2) network connectivity is assumed to be available when it is needed. A mobile data sharing mechanism, therefore, must be able to handle unexpected events that are caused by variations in the surrounding environmental conditions, for example the varying network bandwidth or uncertain connection periods.

5.3 Mobile affiliation model for supporting mobile collaborative works

In this section, we propose a new workgroup model that focuses on supporting mobile collaborative works, called the *horizontal collaboration* (explained in Section 5.3.1). The fundamental idea behind the horizontal collaboration model is that it takes advantage of nearby communication technologies to encourage mobile users to form temporary and dynamic workgroups. By this way, mobile users can continuously carry out collaborative operations while being disconnected from the database servers. We focus our discussion on three important properties of the horizontal collaboration - that are: the *mobile affiliation workgroups* (Section 5.3.2), the mobile sharing workspace called *export-*

import repository (Section 5.3.3), and the mobile data sharing mechanism by the means of *export* and *import transactions* (Section 5.3.4).

5.3.1 Extending workgroup model for mobile work environments

There are many research proposals that have been developed to support collaborative work in distributed environments [RN99, Ram01]. Among these proposals, the common-private workgroup model has been widely applied. In this workgroup model, an organization consists of one or many workgroups each of which consists of one or many members. Each member can work independently and/or cooperate with other members to achieve designed goals. Users work on their own local workspaces, and share a pre-defined common sharing workspace (the common workspace can also be defined at different nested levels, see Figure 5.3). Information is first updated in the local workspace, and then propagated into the common workspace. The local workspaces can be stored at mobile computers or fixed computers. The common workspaces are usually stored together with the database servers or at specific computers. Shared data can be temporarily inconsistent across different local workspaces. In the common workspace, shared data must always be consistent. In the mobile IT support scenario (Section 5.2), while dealing with mobile tasks, an IT officer first works on the local workspace at the mobile computer, and then integrates the results into the common workspace at the database servers.

The private-common workspace model has the capacity to support both synchronous and asynchronous communication among collaborative workers. Users can share their data, and obtain needed information by accessing the common workspace via predefined operations like *sign-off, check-in* and *check-out*. However, the organization of the private-common workgroup model (we shall call this workgroup model the *vertical collaboration*) may not be suitable in mobile environments. This is due to the static configuration of the common workspaces, and the strictness of the communication paths between the private and common workspaces (see Figure 5.3 for illustration). Consequently, there is a need to expand the existing workgroup organization model so that it can exploit the benefits of the new mobile work environment (we shall call this expansion the *horizontal collaboration*). The extended workgroup model takes into account the mobility characteristic of mobile hosts and the wireless communication technologies.

From a collaborative work perspective, the collaboration among mobile users can be carried out in two dimensions: *vertical* and *horizontal*. These collaboration dimensions are illustrated in Figure 5.3 and elaborated as follows:

- **Vertical collaboration.** Collaborative work among mobile users, who belong to static and pre-defined workgroups, is called vertical collaboration. Each workgroup has its own group workspace that is predefined, organized and allocated. Collaborative operations among users must strictly follow the pre-defined hierarchical communication paths.

- **Horizontal collaboration.** Collaborative work of a temporary and dynamic mobile workgroup that is formed from a collection of mobile hosts that belong to one or many pre-defined mobile workgroups is called horizontal collaboration. Nearby and peer-to-peer communication is the main characteristic of the horizontal collaboration. To our knowledge, there is no similar concept (in relation to mobile workspace sharing) that has been defined for this type of collaboration.

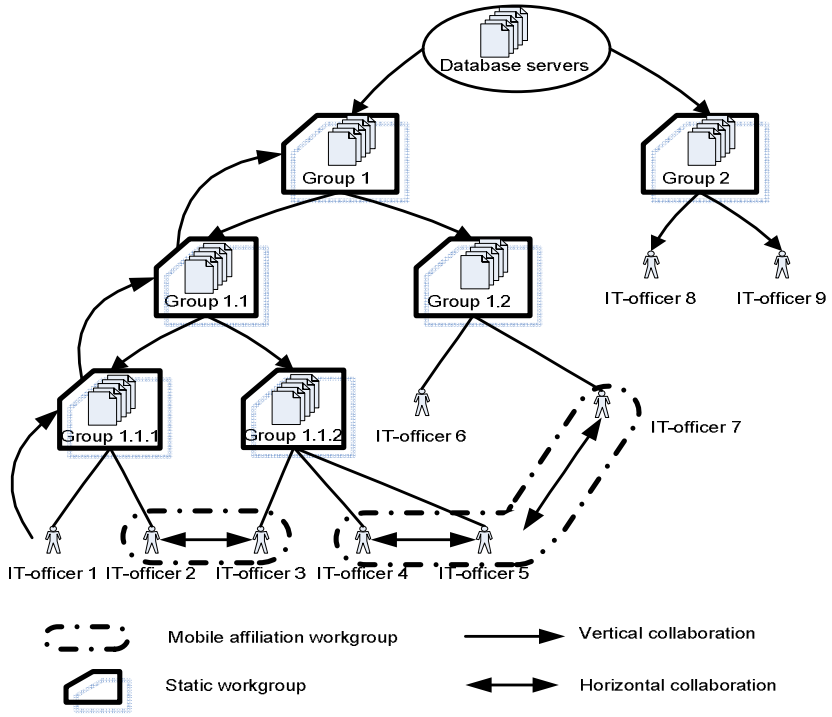

Figure 5.3: Extending collaborative work model in mobile environments

Figure 5.3 illustrates the collaboration work in both dimensions. For vertical collaboration, IT-officers are divided into two main groups: one and two. Group one is divided into sub-groups 1.1 and 1.2. Group 1.1 is further partitioned into sub-groups 1.1.1 and 1.1.2. Group 1.1.1 consists of IT-officer 1 and 2; and IT-officer 3, 4 and 5 are the members of group 1.1.2. Updates by IT-officer 1 are first integrated into the sub-workspace of group 1.1.1, then group 1.1, then the common workspace of group 1. After that, these updates can be downloaded into the sub-workspace of group 1.2, and can be accessed by IT-officer 6. For horizontal collaboration, IT-officer 2 and IT-officer 3 can form a dynamic *mobile affiliation workgroup* so that updated data by IT-officer 2 can be made available to IT-officer 3 without being integrated through the common workspace of group 1.1. Interactions between these two IT-officers in the mobile affiliation workgroup will be supported through an *export-import repository* (explained in Section 5.3.3) and *export* and *import transactions* (addressed in Section 5.3.4).

The extended workgroup model in the horizontal collaboration dimension promotes the benefits of mobile work environments by allowing direct data sharing among mobile hosts. This work model increases the data availability at mobile hosts that can not

connect to the database servers or the common workspace to obtain needed data. Furthermore, as explained in the next subsections, this work model also takes into account the mobility characteristic of mobile hosts, and utilizes the advantages of wireless network technologies.

5.3.2 Mobile affiliation workgroups

An *affiliation workgroup* is a dynamic group of mobile and non-mobile computing hosts that agree to form a temporary workgroup so that they can exchange information or support each other. A computing host in an affiliation workgroup must be able to communicate with other hosts in the workgroup. A *mobile affiliation workgroup* (MA) is an affiliation workgroup where all hosts are mobile hosts. Figure 5.4 illustrates the mobile affiliation groups.

A mobile host will be removed from the mobile affiliation workgroup if it is disconnected from other hosts that are the members of the mobile affiliation workgroup. This could be caused by the disconnections of wireless networks, the exhaustion of battery energy, or the mobile host moves outside the communication range of the mobile affiliation workgroup. A mobile host can participate in more than one mobile affiliation workgroup. A mobile host in a mobile affiliation workgroup can also connect to a mobile support station or database servers. For example, in Figure 5.4, the mobile host MH_1 connects to the mobile support station MSS_2, and joins two different mobile affiliation workgroups MA_1 and MA_2.

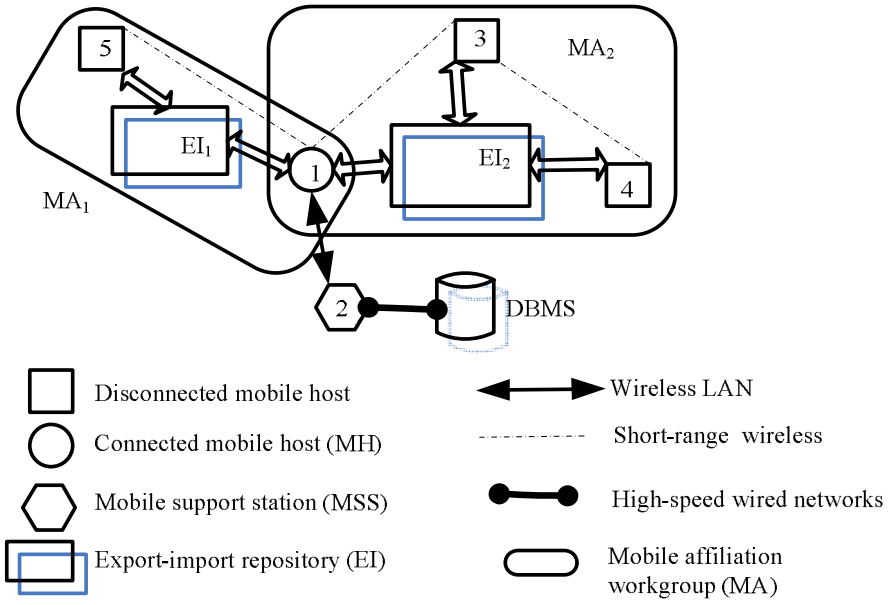

Figure 5.4: Mobile affiliation model

The advantageous characteristics of the mobile affiliation workgroup model are as follows:

- **Represent temporary and dynamic workgroups.** The mobile affiliation workgroup is created when a group of mobile hosts, which are disconnected from the database servers and whose locations are nearby each other, need to collaborate or share data. These mobile hosts will utilize short-range wireless technologies to establish a temporary mobile workgroup. One mobile host can initiate a mobile affiliation workgroup, and a varying number of mobile hosts can join the mobile affiliation workgroup. A mobile host can join or leave the mobile affiliation workgroup at any moment. When the cooperative activities among mobile hosts are completed, the last mobile host in the mobile affiliation workgroup will dispose of the mobile workgroup. This means that there is no central management of the mobile affiliation workgroup, and the disconnection of a mobile host will not destroy the mobile affiliation workgroup.

- **Capture the mobility of mobile hosts.** In a mobile affiliation workgroup, a mobile host uses wireless technologies to connect with nearby mobile hosts. If a mobile host wants to join a mobile affiliation workgroup, it must be within the communication range of the other members. In other words, the distance between mobile hosts impacts their connectivity ability. Therefore, the movement of mobile hosts has a strong impact on the mobile affiliation organization. The mobile affiliation workgroup model also provides a level of mobility transparent to mobile users or applications. A group of mobile hosts can be considered as a group of non-movement hosts as long as they belong to one mobile affiliation workgroup, i.e., their relative distances always comply with the scope of the communication range. For example, if a group of mobile hosts is always moving closely together, it would appear to a mobile user or a mobile application that there is no change in the group organization and surrounding environments.

- **Take into account the constraints of mobile resources.** While participating in a mobile affiliation workgroup, a mobile host interacts with other mobile hosts. This means that the operation mode of the mobile host is the interaction mode. As we have discussed in Section 3.2.4, the behavior of mobile hosts depends on the availability of mobile resources. For example, when a mobile host is running out of battery energy, it can disable its network connectivity and leave the mobile affiliation workgroup. Thus, the mobile affiliation workgroup model takes into consideration the constraints of the mobile resources.

5.3.3 Mobile sharing workspaces

An export-import (EI) repository is a dynamically configurable mobile sharing workspace that belongs to a mobile affiliation workgroup. The mobile sharing workspace provides a means for transaction processes at mobile hosts to share data while being on the move and disconnected from the database servers (see Figure 5.4 above). The advantageous characteristics of the export-import repository are as follows:

- **Dynamic sharing workspace**. The export-import repository is created when there is a need for sharing of data among transactions at different mobile hosts. A transaction T_i^k at the mobile host MH_i will initiate an export-import repository if it reaches the synchronous point (at which there is a need for exchanging shared data) before its associated transaction T_j^l that is being executed at the mobile host MH_j. Otherwise, the export-import repository can also be initiated by the transaction T_j^l. An export-import repository is initiated by a transaction at a mobile host, but a varying number of transactions at different mobile hosts can join the mobile sharing workspace for different purposes, for example sharing or obtaining necessary data. When the data sharing activities among transactions at different mobile hosts are completed, the export-import repository will be disposed.

- **Temporary persistent sharing workspace**. The export-import repository is dynamically created to support the data sharing, which could be partial state (see Section 5.5.4) or status (see Section 5.5.5), among transactions at different mobile hosts. The shared data in the mobile sharing workspace will eventually be integrated into the database servers by the participating transactions. Therefore, its content must be saved in a persistent storage. Moreover, this information can also be used to support recovery processes if there is any failure or conflict among the participating transactions (see Section 6.7).

- **Distributed sharing workspace**. The export-import repository is dynamically allocated and distributed among the mobile hosts in the mobile affiliation workgroup. For example, in Figure 5.4, the export-import repository EI_2 can be entirely allocated at the mobile host MH_1, or distributed among three mobile hosts MH_1, MH_3, and MH_4. This also enhances the scalability of the export-import repository and the availability of shared data in the mobile environment. If a mobile host is exhausting its energy and going to be disconnected from the mobile affiliation workgroup, the shared data in the mobile sharing workspace partition that is currently allocated at this mobile host will be reallocated to other available mobile hosts so that this shared data is still available to other transactions. For example, if the mobile host MH_1 is going to be disconnected from the mobile affiliation workgroup MA_2, the shared data that is currently stored at the mobile host MH_1 can be moved to either the mobile host MH_3 or MH_4.

A mobile host can participate in more than one mobile affiliation workgroup. Consequently, a transaction at the mobile host can join and access more than one export-import repository. In Figure 5.4, transactions at the mobile host MH_1 can access both export-import repositories EI_1 and EI_2, while transactions at the mobile host MH_3 can only access the export-import repository EI_2.

5.3.4 Export and import transactions

In this section, we present a flexible and adjustable mechanism to support the sharing of data among transactions at different mobile hosts, which are the members of a mobile affiliation workgroup. The idea behind our data sharing mechanism is: *using separate*

transactions to support data sharing among transactions at different mobile hosts. The data sharing among transactions in mobile environments is autonomously carried out by special transactions (called shared transactions – as discussed below) that interact through an export-import repository. By this, the data sharing can be carried out in both a synchronous and an asynchronous manner, i.e., coping with the volatile environmental conditions.

We differentiate two types of transaction: *standard transaction* and *shared transaction* (see Figure 5.5*)*. A standard transaction that shares data to or obtains data from other transactions is called a *delegator* or *delegatee* transaction, respectively. In some cases, a standard transaction can play roles as both delegator and delegatee transaction. Shared transactions include *export* and *import* transactions that support the delegator and delegatee transactions to share data (from now, we assume that the delegator and delegatee transactions belong to different mobile hosts). Export transactions interact with import transactions in export-import repositories. We also differentiate two types of data sharing: *sharing data state* and *sharing data status*. Sharing data state of data item X between a delegator and a delegatee transaction means that the delegator transaction shares the value V_X of data item X to the delegatee transaction. For sharing data status, the delegator transaction shares the lock (which is either a read X_R or write X_W lock – see more details in Section 5.5.3) on data item X to the delegatee transaction. To ease the discussion, we use the following notations: T_i^k denotes a transaction T^k at mobile host MH_i; an export transaction and an import transaction of a standard transaction T_i^k are denoted by $T_i^{k.E}$ and $T_i^{k.I}$ respectively.

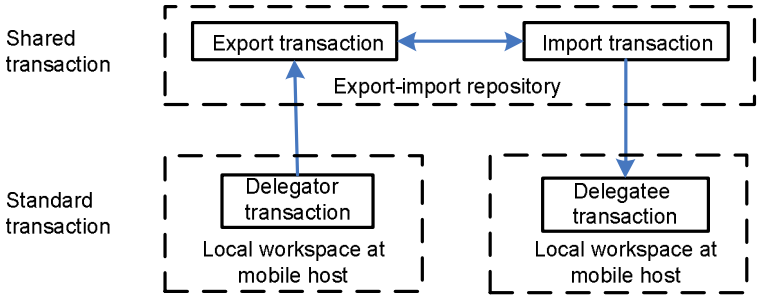

Figure 5.5: Standard and shared transactions

The roles of the export and import transactions are as follows:

- **Export transaction**. The role of an export transaction $T_i^{k.E}$ is to support a delegator transaction T_i^k: (1) to share its partial or committed results with delegatee transactions; (2) to transfer locks on shared data to delegatee transactions; and (3) to save partial results, i.e., avoid losing useful work due to failures of mobile hosts. The delegator transaction will initiate one or more export transactions when it wants to share information with other delegatee transactions. The correlation between a delegator transaction and its export transaction is an *abort-dependency* [CR94], see Section 6.2 for further discussion.

- **Import transaction**. An import transaction $T_i^{k.I}$ supports a delegatee transaction T_i^k at a mobile host to obtain needed information that can be either data states or data status from other delegator transactions. The delegatee transaction can initiate one or more import transactions to acquire the necessary information from other transactions. The correlation between a delegatee transaction and its import transactions is either an *abort-dependency* [CR94] or a *multiple-abort-dependency*. These transaction dependencies will be discussed in detail in Section 6.2.

Note that the idea of this mobile data sharing mechanism is not completely unknown in other research fields, like operating systems or parallel processing systems. For example, a process may use different threads to handle inputs and outputs or to communicate with other processes. The Linda parallel computing system [PMR00] also applied transaction concepts to support data sharing among parallel processes. However, there is a crucial difference: in our model, shared transactions are not strictly under control of the original standard transactions, i.e., the shared transactions can independently continue executing even if the original standard transactions fail.

The export and import transactions provide a flexible and adaptive mechanism to support mobile data sharing. This data sharing mechanism has the ability to deal with the dynamic changes of surrounding mobile environmental conditions and the constraints of mobile resources. The mobile data sharing mechanism also has several qualities that are as follows:

- **Cope with interruptions of synchronous data sharing.** The sharing of data among standard transactions T_i^k and T_j^l can be carried out in a synchronous manner if these two transactions are simultaneously connected to each other. In mobile environments, however, interruptions can happen any time during the synchronous data sharing process. Thus, the data sharing mechanism must have the ability to recover from the interruptions to ensure that the data sharing process is correctly carried out.

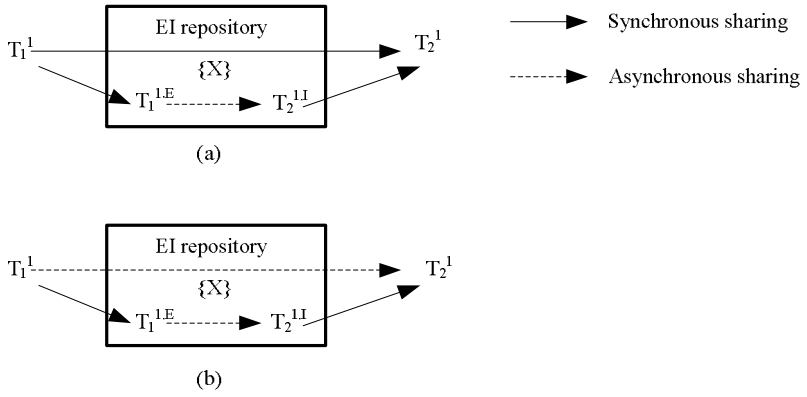

Figure 5.6: Adaptive mobile data sharing mechanism

In Figure 5.6(a), during the synchronous data sharing between two transactions T_1^1 and T_2^1, an export transaction $T_1^{1.E}$ and an import transaction $T_2^{1.I}$ are initiated and executed as back-up shared transactions in parallel with the transactions T_1^1 and T_2^1.

If a disconnection occurs, the data sharing process (via the export and import transactions) between the transactions T_1^1 and T_2^1 can continue in an asynchronous manner (see discussed below). In other words, the data sharing mechanism has the ability to withstand failures of connectivity.

- **Support asynchronous data sharing.** Due to the disconnections and interruptions in communication, asynchronous data sharing mechanisms must be supported. Pairs of export and import transactions are used to support asynchronous data sharing among disconnected standard transactions. In Figure 5.6(b), two standard transactions T_1^1 and T_2^1 are disconnected; however, the delegator transaction T_1^1 can connect to the export-import repository and share data item X to the delegatee transaction T_2^1 via its export transaction $T_1^{1.E}$. Asynchronously, the delegatee transaction T_2^1 can connect to the export-import repository to obtain this data item via its import transaction $T_2^{1.I}$.

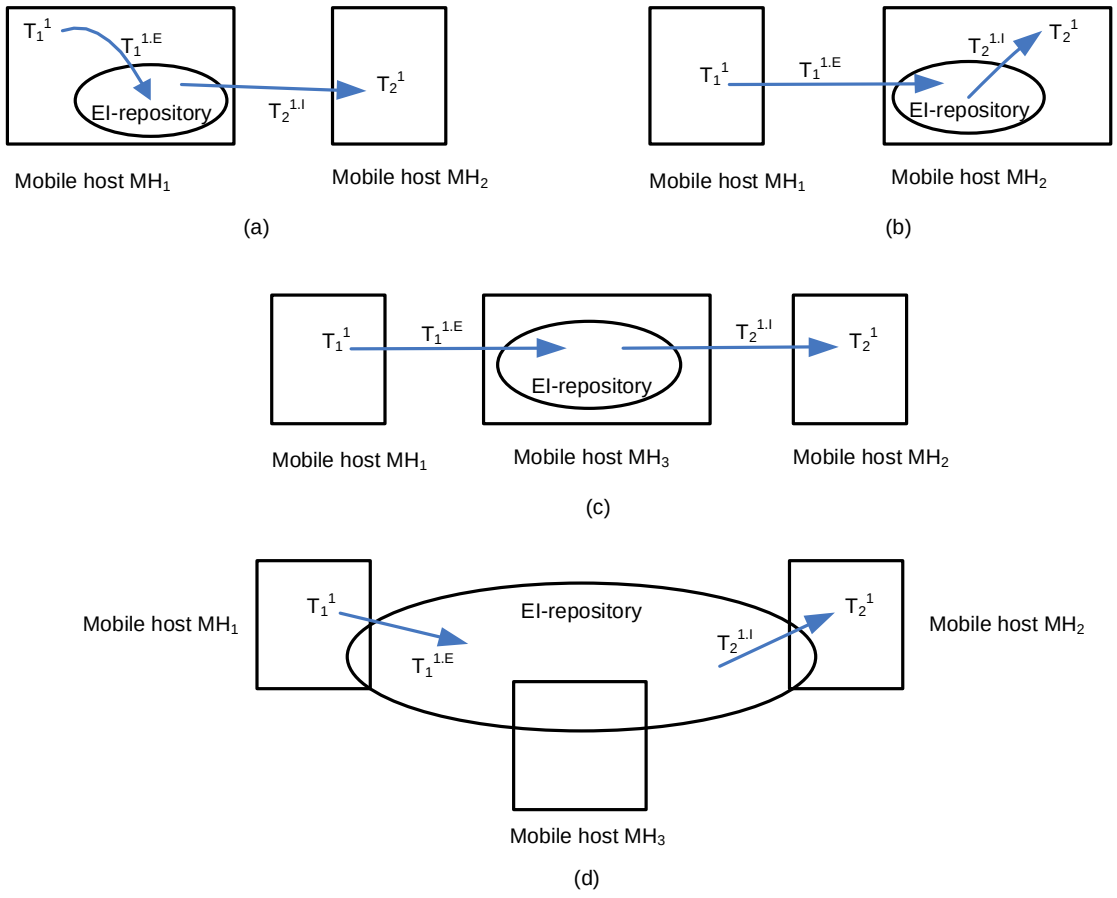

Figure 5.7: The physical distribution of the export-import repository

Note that the export-import repository illustrated in Figure 5.6 is a logical mobile sharing workspace. As we have discussed in Section 5.3.3, the real physical export-import repository can be allocated among different mobile hosts. The distribution of the physical mobile sharing workspace among mobile hosts is illustrated in Figure

5.7. In the figure, the delegator transaction T_1^1 and the delegatee transaction T_2^1 are executed at the mobile hosts MH_1 and MH_2, respectively. If the export-import repository is allocated at either mobile host MH_1 or MH_2, in order to share data, either the import transaction $T_2^{1.I}$ must connect to the export-import repository at the mobile host MH_1 (see Figure 5.7 (a)) or the export transaction $T_1^{1.E}$ must connect to the export-import repository at the mobile host MH_2 (see Figure 5.7 (b)). In other words, connectivity between these two mobile hosts MH_1 and MH_2 is required. However, if the export-import repository is allocated at other hosts, e.g., the mobile support station MH_3 (see Figure 5.7 (c)), synchronous connectivity between the mobile hosts MH_1 and MH_2 is not necessarily required. If the export-import repository is physically distributed among mobile hosts (see Figure 5.7 (d)), the shared transactions can connect to any partition of the export-import repository to share data. When the export-import repository is physically allocated among different mobile hosts, there is a need for support management of the mobile sharing workspace and the shared data (see Section 5.6 for further discussion).

- **Separate data sharing processes from the main transaction processes.** The data sharing processes are separated from the main transactions that might be large and long-lived. Furthermore, a large shared data amount can be divided into smaller sets and shared via a number of shared transactions. By this way, the mobile data sharing mechanism can deal with the low bandwidth and short connection time of the wireless networks. For example, in Figure 5.8, a delegator transaction T_1^2 uses two export transactions $T_1^{2.E1}$ and $T_1^{2.E2}$ to share data items Y and Z in the export-import repository. These sharing processes can be carried out by one export transaction if both the data items are ready to be shared at the same time, and both the network bandwidth and connection time are suitable for the data transmission.

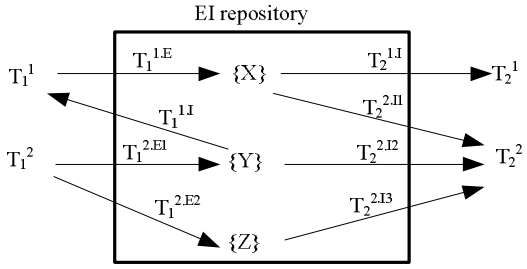

Figure 5.8: A general data sharing scenario

- **Provide a flexible mobile data sharing system.** Via the support of export and import transactions, the data sharing among transactions through an export-import repository is flexible. One delegator transaction can share information with one or many delegatee transactions, many delegator transactions can share data with one delegatee transaction, and even recursive data sharing is possible (explained in Section 5.5.6). For example, in Figure 5.8, a delegator transaction T_1^1 shares the data object X to both delegatee transactions T_2^1 and T_2^2 via one export transaction $T_1^{1.E}$; the delegatee transaction T_2^2 can obtain shared data from both delegator transactions T_1^1 and T_1^2; and the transaction T_1^1 plays roles as both delegator and delegatee transaction.

- **Support the mobility of transactions.** During their execution processes, standard transactions can participate in more than one export-import repository when the mobile host joins many mobile affiliation workgroups. The dynamic structure of shared transactions (see Section 5.7.3) will support the mobile transaction processing system to handle the mobility of standard transactions across many export-import repositories.

5.4 Discussions of mobile transaction properties

In the previous section, we have presented our proposal to extend the collaborative work model in the horizontal dimension in order to support mobile collaborative work. This extension leads to the development of an adaptable mobile data sharing mechanism among standard transactions at different mobile hosts via the support of shared transactions. In this section, we first discuss the domain of data consistency related to collaborative work in mobile environments. Then, we discuss the transaction properties of the shared and standard transactions.

5.4.1 Domains of data consistency

For a mobile information system that supports mobile collaborative work, there are four domains of data consistency: (1) *local consistency*, (2) *group consistency*, (3) *mobile affiliation consistency*, and (4) *global consistency*. The local consistency is applied for data objects that reside in a private (or local) workspace. This means that in mobile environments, the local consistency is applied to data that is being cached at a mobile host. For the vertical collaboration dimension, the group consistency [Ram01] represents the consistency of shared data items in the group workspace. The states of these shared data items are the results of the integration of local workspaces into the static group workspace. For the horizontal collaboration dimension, the mobile affiliation consistency is applied for data items which are shared by the standard transactions. In other words, the mobile affiliation consistency represents the consistency of data items that are shared in the export-import repository. Finally, when shared data items in local workspaces, group workspaces and mobile sharing workspaces are successfully integrated into the database servers, these data items are said to be in the global consistency domain.

For the vertical collaboration dimension, only three domains of data consistency are applied: the local consistency, the group consistency, and the global consistency. However, for the horizontal collaboration dimension, all the four domains of data consistency are used. The group consistency is applied for the horizontal collaboration when several mobile hosts that belong to one mobile affiliation workgroup are statically organized into sub-workgroups, i.e., vertical collaboration within a horizontal collaboration. Table 5.2 summaries the correlation between the collaboration dimensions and the domains of data consistency.

There are many research works that have been focusing on achieving data consistency in the vertical collaboration dimension [Ram01]. These works usually support collaborative work in non-mobile environments, thus, they may not be adequate for mobile

Table 5.2: Collaboration dimensions and consistency domains

		Data consistency			
		Local	*Group*	*Mobile affiliation*	*Global*
Collaboration Dimension	*Vertical*	Relevant	Relevant	N/A	Relevant
	Horizontal	Relevant	Partial relevant	Relevant	Relevant

environments. For example, the mobility of mobile hosts and the limitations of network connectivity have not been taken into consideration. For the rest of the thesis, we will concentrate our research on the three main data consistency domains in the horizontal collaboration dimension, i.e., without the group consistency. Figure 5.9 illustrates the relationship among the domains of data consistency in the horizontal collaboration dimension.

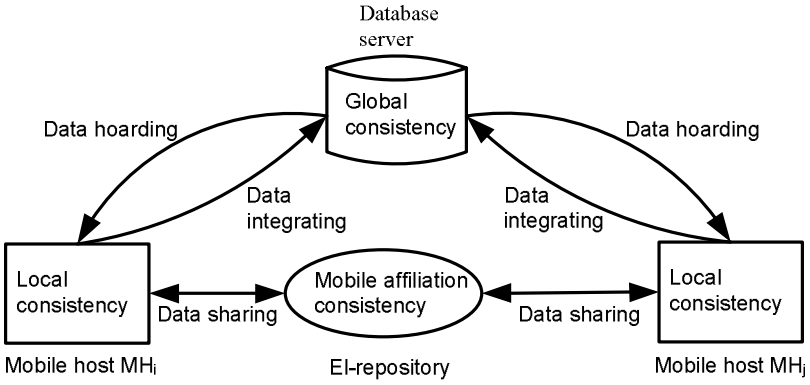

Figure 5.9: Domains of data consistency in horizontal dimension

As discussed in the previous section, the main objective of the horizontal collaboration is to enhance the data availability at disconnected mobile hosts via the support of the adaptable mobile data sharing mechanism. The local consistency is achieved through a data hoarding stage with the assistance of anchor transactions and a mobile data sharing stage with the support of shared transactions (see Sections 6.3 and 6.4). The mobile affiliation consistency is assured via the support of shared transactions (described in Sections 5.4.2 and formalized in Section 6.4), while the global consistency is accomplished through a transaction integration stage (see Section 6.6).

5.4.2 Shared transactions

In this section, we discuss the ACID properties of shared transactions. To recap, the shared transactions are export and import transactions that support the mobile data sharing among standard transactions through an export-import repository. For the shared

transactions, the important events [CR94] are *begin, commit,* and *abort.* Table 5.3 summaries the behavior of export and import transactions in relation to the important events.

Table 5.3: Behavior of shared transactions

Event	Export transaction	Import transaction
Begin	Initiated by a delegator transaction from local workspace	Initiated by a delegatee transaction from local workspace
Commit	Committed in the export-import repository	Committed in the local workspace
Abort	Aborted or restarted	Aborted or restarted

An export transaction $T_i^{1.E}$ is initiated by the delegator transaction T_i^1 to share data in the export-import repository. This means that the export transaction $T_i^{1.E}$ is initiated from the local workspace, and commits in the mobile sharing workspace (see illustration in Figure 5.10). If there is any failure during the execution of the export transaction, either the export transaction will be restarted based on the log records in the local workspace (see Section 6.4 for further discussion), or if the delegator transaction has disconnected from the export-import repository, the export transaction will be aborted. Furthermore, if the delegator transaction wants to withdraw its shared data, the corresponding export transaction will also be aborted by the delegator transaction. If the corresponding export transaction has committed, it will be compensated.

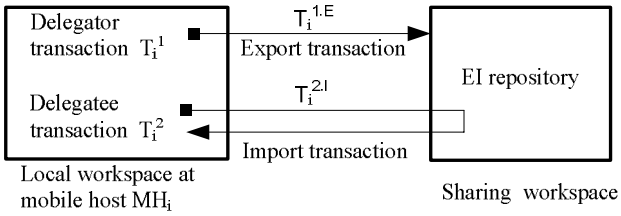

Figure 5.10: Behavior of export and import transactions

An import transaction $T_i^{2.I}$ is initiated by the delegatee transaction T_i^2 to obtain shared data from delegator transactions through the export-import repository. The import transaction is initiated from the local workspace, collects shared data from the export-import repository, and finally commits in the original local workspace. In other words, the execution of the import transaction involves both the mobile sharing workspace and the local workspace. If the delegatee transaction decides that the wanted shared data is no longer needed, the import transaction will be aborted. On the other hand, if there is a failure during the execution of the import transaction and the delegatee transaction still connects to the export-import repository, the import transaction will be restarted.

The following discussion addresses in detail the properties of the export and import transactions. Table 5.4 summaries the properties of export and import transactions.

Table 5.4: Properties of shared transactions

Properties	Export transaction	Import transaction
Atomicity	Fulfillment in the export-import repository	Relaxation in the local workspace
Consistency	Fulfillment in the export-import repository	Fulfillment in the local workspace
Isolation	Fulfillment in the export-import repository	Relaxation in the local workspace
Durability	Fulfillment in the export-import repository	Fulfillment in the local workspace

Atomicity property

The export transaction fulfills the standard atomicity property. This fulfillment ensures that information is either successfully shared or no information is shared. The export transaction has the ability to unilaterally commit or abort. When the export transaction commits, the shared data is successfully written into the export-import repository so that other import transactions can start reading these shared data. If the export transaction is aborted due to execution errors, then no information is shared.

The import transaction relaxes the atomicity property. The import transaction obtains shared data from the export-import repository. If there is a failure during the execution of an import transaction, the import transaction can partially roll back and some of the already collected shared data can be saved in the local workspace. This relaxation can help the delegatee transaction to make use of some needed data, especially if the collected data is read-only and consistent. For example, a delegatee transaction T_j^l initiates an import transaction $T_j^{l.I}$ to collect a set of read-only shared data. The import transaction $T_j^{l.I}$ will continuously read the needed data from the export-import repository and save these shared data in the local workspace. If the import transaction $T_j^{l.I}$ fails, it should be allowed to partially roll back, i.e., some of the collected data can be saved in the local workspace.

Consistency property

The standard consistency property means that committed transactions will transfer a database from a consistent state to another consistent state. In our mobile transaction processing system, the shared transactions support the standard transactions to carry out the mobile data sharing processes across different local workspaces. In terms of data consistency, this means that when a shared transaction commits, the shared data is consistent across the local workspaces and the mobile sharing workspace.

The export transaction fulfills the consistency property within the scope of the export-import repository. This means that when an export transaction commits, the state of the shared data written into the mobile sharing workspace is consistent with the state of this shared data in the local workspace in which the delegator transaction is being executed. If the delegator transaction aborts after the export transaction has committed, the export

transaction will be compensated so that the invalid shared data will be withdrawn from the export-import repository. If there is an import transaction that has read this invalid shared data, the mobile transaction processing system must provide mechanisms to correct the problem. This can be done by explicitly defining abort-dependency rules [CR94] between the standard and shared transactions (see Sections 5.4.3 and 6.2 for more detail).

For import transactions, the consistency property is fulfilled within the scope of the local workspace at the mobile host. This means that when an import transaction commits, the state of the collected shared data written into the local workspace is consistent with the state of this shared data currently owned by the delegator transactions. In other words, the shared data is consistent across the local workspaces in which the delegator and the delegatee transactions are being carried out. If the shared data being read by an import transaction is invalidated (i.e., the delegator transaction aborts and the export transaction is compensated), the import transaction will be compensated. Consequently, delegatee transactions that also have read invalid shared data (in the local workspace) must be aborted.

Isolation property

For export transactions, the standard isolation property is fully met. In other words, any related import transactions can only gain access to shared information after the export transaction has committed in the export-import repository. To assure this, strict two-phases locking can be applied or explicit *commit-begin-dependency* [CR94] rules may be defined by the mobile transaction processing system.

For import transactions, the isolation property is relaxed. The relaxed isolation property of import transactions avoids blocking of data availability in the local workspace if the commitment of the import transaction is being postponed. This can happen due to the disconnection of wireless networks or the mobility of the delegatee transaction. So, it should be feasible for the import transaction to reveal intermediate results to the delegatee transaction before its commitment. Note that the intermediate results of the import transaction may only be visible to the original delegatee transaction. In Figure 5.11, the delegatee transaction T_1^1 must have the right to access shared data item X that is collected by its import transaction $T_1^{1.1}$ before a local transaction T_1^2.

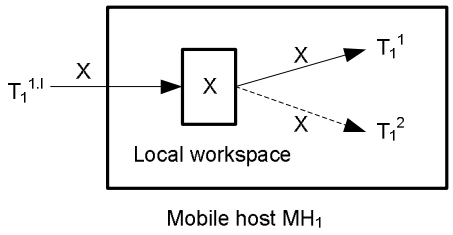

Figure 5.11: Access privilege of a delegatee transaction to imported data

The remaining question is how the relaxation of isolation property is achieved. The answer depends on the structure of the delegatee transaction, i.e., flat or nested structure (see Table 5.5).

If the delegatee transaction has a flat structure, either the import transaction can be merged into the structure of the delegatee transaction by the concepts of Split-Join transactions [PKH88], or the import transaction can delegate its partial results to the delegatee transaction by the concepts of Reporting and Co-transactions [Chr93]. This can be done because the import and delegatee transactions are tightly coupled in the local workspace. If the delegatee transaction has a nested structure, the import transaction can be adopted as a sub-transaction of the delegatee transaction (see Section 5.7.2 for further discussion).

Table 5.5: Relaxing the isolation property of import transactions

Structure of delegatee transaction	Relaxation mechanism
Flat structure	Merge or delegate the import transaction results to the delegatee transaction.
Nested structure	Adopt the import transactions as sub-transactions of the delegatee transaction.

Durability property

The standard durability property safeguards the results of committed transactions so that these results will be recovered when failures occur. When an export transaction commits, the shared data is persistent in the export-import repository of the mobile affiliation workgroup. The export-import repository will be disposed when the mobile affiliation workgroup is no longer existing. Therefore, the delegator transaction must log the information associated with its export transaction in the local workspace at the mobile host before dispatching the export transaction to the mobile sharing workspace (see Section 6.4 for further detail and formalization). This means that the durability property of export transactions is assured by the delegator transaction.

When an import transaction commits, the collected shared data is durable in the local workspace. The durability of shared data is assured by the logging facility that is provided by the transaction manager at the mobile host. Furthermore, related information such as the identification of the delegator and export transactions will also be recorded in the local log at this mobile host.

5.4.3 Standard transactions

In this section, we discuss the properties of standard transactions. To recap, the standard transactions are delegator or delegatee transactions that are executed locally within the scope of the local workspace at a mobile host. Standard transactions are normally long-lived transactions, with a complex structure; and demands additional support such as

disconnected and distributed transaction processing (see Section 3.5). Due to these characteristics, the standard ACID properties may be too strict for the standard transactions. For example, the atomicity property requires that either all transaction operations or no operation must be completed. For long-lived transactions, this standard atomicity property may waste useful work that has been done. The standard isolation property prevents an on-going transaction to share the available information with others; therefore it could block the execution processes of other transactions.

Transactions in mobile environments require less strict properties, and this is the approach that has been applied in many mobile transaction models [SRA04]. For example, relaxing the atomicity property allows transactions to partially rollback when there is a failure. Relaxing the isolation property makes it possible for the immediate results of an on-going transaction to be accessible to other concurrent transactions. This way, these transactions have an opportunity to be executed faster. For the consistency property, it is important that database states must be consistent at specific domains and time. For example, before a mobile host is disconnected, the data, which is cached in the local workspace, must be consistent with the one at the database servers so that local transactions at the mobile host can be performed correctly in the disconnected mode. During the disconnected transaction processing stage, the cached data at the mobile host could have been modified and thus, be different from the one stored at the database servers or at other mobile hosts. When the mobile host reconnects to the database servers, these different data states will have to be reconciled to achieve a global consistent state. For the durability property, the results of a committed transaction must be durable only after the transaction has committed at the database servers.

The remaining question is: *how much relaxation of the transaction properties could a mobile transaction processing system support?* The following analysis of the properties of standard transactions will answer this question (see Table 5.6).

Table 5.6: Properties of standard transactions

Properties	**Standard transaction**
Atomicity	Relaxation in local and global workspaces
Consistency	Fulfillment in the global workspace
Isolation	Relaxation in local and across local workspaces
Durability	Fulfillment in the global workspace

To ease the following analysis, we recap the important characteristics of our mobile transaction processing system:

- There is no constraint in roles and structure of a standard transaction at the mobile host, i.e., a standard transaction can have a flat or nested structure, and can play role as either a delegator transaction or a delegatee transaction or both.
- The execution process of a standard transaction involves a local workspace and a global workspace. This means that a standard transaction could have either (1) first

committed in the local workspace and then in the global workspace; or (2) committed directly to the global workspace.

- During its execution process, a standard transaction can involve one or many export-import repositories (i.e., the mobile host can join one or many mobile affiliation workgroups) with corresponding export and import transactions.

The following discussion addresses in detail the properties of the standard transactions.

Atomicity property

The atomicity property of standard transactions must be relaxed. This allows local transactions at a mobile host to partially rollback when failures occur. In mobile environments, the relaxation of the atomicity property is essential because: (1) it supports transactions to cope with interruptions, for example disconnections of wireless networks or exhausting battery energy; and (2) it prevents losing useful work done under the constraints of mobile resources, especially for long-lived transactions.

In non-mobile environments, there are several approaches to support customizing the atomicity property of transactions. For transactions with a flat structure, the relaxation of atomicity property can be achieved by save points or allowing transactions to partially commit [GR93]. For nested transactions, this can be achieved by explicitly defining *abort-dependency* rules among related transactions [Ram01]. These approaches can also be applied in our mobile transaction processing system to support the relaxation of the atomicity property of local transactions in the scope of the local workspace at the mobile host and in the global workspace (i.e., when the local transactions are integrated to the database server).

In our mobile transaction processing system, shared transactions are used to support the data sharing among standard transactions that are carried out in different workspaces. A delegator transaction initiates export transactions to share or save partial results in an export-import repository. When a delegator transaction aborts (in the local workspace or global workspace), it is not necessary that all export transactions must also be aborted (because the export transactions have shared consistent data – see Section 5.5.4 for further discussion). Therefore, a delegator transaction can partially rollback and restart when failures occur.

A delegatee transaction initiates import transactions to collect necessary data from the export-import repository. If a delegatee transaction aborts in the local workspace, its import transactions can still be carried out so that the collected data can still be used either when the delegatee transaction recovers from failures or by other local transactions at the mobile host. If a delegatee transaction is aborted when it is integrated in the global workspace, it is not necessary that all the associated import transactions must also be aborted. For example, in Figure 5.12, if the delegatee transaction T_1^1 aborts, two of its import transactions $T_1^{1.I2}$ and $T_1^{1.I3}$ are aborted, but not the import transaction $T_1^{1.I1}$. The relaxed atomicity property can be achieved by defining *abort-dependency* rules between a standard transaction and its shared transactions (see Section 6.2 for more detail).

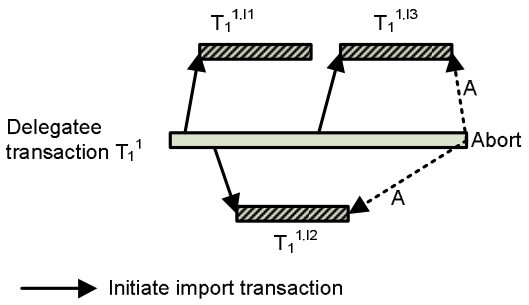

Figure 5.12: Dependencies between delegatee and import transactions

Furthermore, if the standard transaction has a nested structure, the relaxed atomicity can also be achieved by defining an *abort-dependency* between the parent transaction and shared transactions of children sub-transactions. For example, in Figure 5.13, if the sub-transaction T_1^{21} aborts, the export transaction $T_1^{21.E}$ and the import transaction $T_1^{21.I}$ will not be aborted. These shared transactions of the sub-transaction T_1^{21} will only be aborted if the parent transaction T_1^{2} aborts. Similarly, the import transaction $T_1^{1.I}$ is only aborted when the root transaction T_1^{0} aborts.

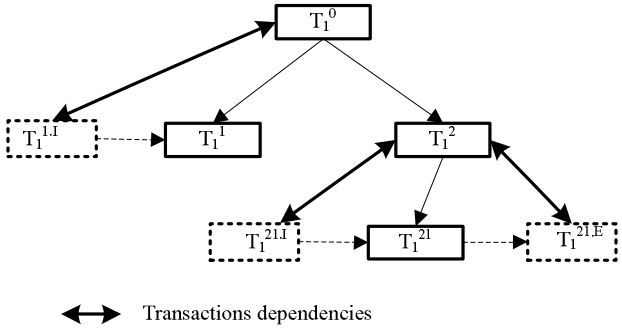

Figure 5.13: Dependencies between parent and children's shared transactions

Consistency property

The execution of standard transactions involves two workspaces: (1) the local (or private) workspace at the mobile hosts, and (2) the global workspace at the database servers. When the mobile hosts are disconnected from the database servers, the standard transactions are locally executed within the scope of the local workspace at the mobile host. The consistency in the domain of a local workspace is ensured by the correctness criterion of local transactions, i.e., a serializable schedule of local transactions.

The data consistency, however, is not always guaranteed among different local workspaces at different mobile hosts. In our mobile transaction processing system, the data conflict awareness among standard transactions in different local workspaces is supported by the concept of *anchor transactions* (see Section 5.5.2 for description). When the mobile hosts reconnect to the database servers, transaction integration processes are carried out to determine the global execution order of local transactions. If

a global serializable schedule is achieved, the local transactions are finally committed at the database servers and global consistency is achieved (see Section 6.6 concerning the transaction integration stage).

Isolation property

In our mobile transaction processing system, the isolation property of standard transactions is relaxed in both local and across local workspaces. Relaxing the isolation property allows standard transactions to share their intermediate results to others. It would not be a problem if the transaction will never abort. However, if a standard transaction whose intermediate results have been shared aborts, we have to ensure that these shared intermediate results will not cause data inconsistency problems, i.e., those transactions that have read the invalid shared data must be aborted too.

Local transactions are tightly coupled together in the local workspace at a mobile host. Therefore, within the scope of the local workspace, a local transaction can share its partial results to other local transactions via existing data sharing mechanisms, for example delegation operations [CR94, Ram01]. For standard transactions that are executed in different workspaces, the intermediate results of a standard transaction can be shared via export and import transactions through the export-import repository. The data sharing process among standard transactions at different local workspaces consist of three phases (see Figure 5.14): (1) between the standard delegator and export transactions, (2) between export and import transactions in the export-import repository, and (3) between the import and delegatee transactions. The mobile transaction processing system must ensure that all these three steps are taken into consideration when the delegator transaction aborts. In other words, it is necessary to explicitly define *abort-dependency*, *commit-dependency* or *multiple-abort-dependency* (see Section 6.2) rules among the involved transactions for each of the three data sharing phases. For example, if a delegator transaction aborts and withdraws the shared data, its export transactions must be aborted or compensated. Consequently, the import transactions that have read the shared data from the export transaction have to abort too. The abortion of an import transaction may lead to the abortion of the associated delegatee transaction (see Section 6.2 for detailed discussion). The dependency between a delegator and a delegatee transaction in the global workspace, then, will be transitively determined via the intermediate transaction dependencies.

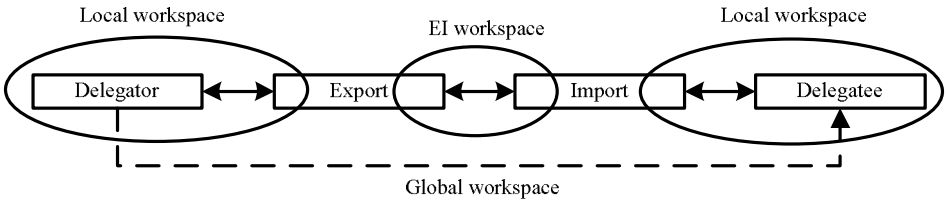

Figure 5.14: Data sharing stage between delegator and delegatee transactions

However, it is not practical that all the intermediate transaction dependencies must be defined at the beginning of the mobile data sharing process. This is due to several

reasons: (1) information related to shared transactions is not known in advance and (2) the mobile data sharing processes might not be carried out as planned. For example, the delegator and export transactions do not know about the import or delegatee transactions that will read the shared data. The actual transaction dependencies may be dynamically injected to or withdrawn from the mobile transaction processing system in accordance with the actual interactions among the participating shared and standard transactions. Dynamic transaction dependencies are adequate for transactions in mobile environments because these transactions are normally long-lived and interactive transactions (as discussed in Section 3.5).

Durability property

In mobile environments, the commitment of a transaction is divided into two stages: local commit in the local workspace, and final commit at the global workspace. A local transaction that has committed in the local workspace at the mobile host could be aborted when it is integrated at the database servers due to conflicting with other transactions. If there is no conflict, the locally committed transactions are finally committed in the global workspace, and global durability is enforced. Moreover, if a local transaction is carried out at the disconnected mobile host with consistent data (see Section 6.5), this transaction must be guaranteed to finally commit in the global workspace when the mobile host reconnects to the database servers.

5.5 Management of mobile data sharing mechanisms

One of the main limitations of the existing mobile transaction models is the lack of customizable mechanisms to support the mobile data sharing in accordance with the changes of the mobile environmental conditions and the behavior of mobile hosts. In this section, we address the issue of mobile data sharing among standard transactions at different mobile hosts via the support of shared transactions. First, we present the operational model of the mobile transaction processing system (Section 5.5.1). Second, we present the concept of an *anchor transaction* (Section 5.5.2) that supports conflict awareness among different local workspaces. Next, in Section 5.5.3, we argue that it is necessary to differentiate between sharing data state and sharing data status. We focus our discussion on the mobile data sharing mechanism that includes sharing of data states (Section 5.5.4) and data status (Section 5.5.5). Finally, in Section 5.5.6, we discuss the issue of recursive data sharing.

5.5.1 Operational model of the mobile transaction processing system

Formally, our mobile transaction processing system consists of a large database *DB* that is distributed among several fixed and wire-connected database servers S_i. Database operations can be performed at any database server, and the results are immediately propagated to other servers via the eager replication protocol [CDK00].

We also distinguish two classes of transactions in mobile environments: *online transaction* and *offline transaction*. An *online transaction* is a transaction that directly

accesses data at the fixed database servers. In other words, an online transaction directly interacts with the transaction manager at the fixed database servers to perform read or write operations on shared data. An *offline transaction is* a transaction that is executed in the local workspace and managed by the mobile transaction manager at the disconnected mobile host.

For online transactions, the transaction and database management systems at fixed database servers make use of standard lock modes, i.e., read and write locks, and the two phase locking protocol (2PL) [BHG87] to enforce data consistency, i.e., by a serializable execution schedule of transactions. An offline transaction that is executed at a disconnected mobile host can acquire read or write locks on shared data with the help of an proxy transaction, called an *anchor transaction* (informally, an anchor transaction is an online transaction that is never aborted, see further explanation in Section 5.5.2). The transaction manager at a mobile host also makes use of standard 2PL to ensure data consistency in the local workspace, i.e., by a serializable execution of local (offline) transactions.

Transactions at a mobile host can connect to any database server to acquire consistent data or to synchronize data that is asynchronously modified. The database servers grant read or write locks on shared data items that are requested by the anchor transaction, which represents offline transactions which are going to be executed at the mobile hosts.

In the following sections, lock and unlock actions on shared data item X are denoted by l_X and ul_X. The read and write locks on shared data item X are denoted by X_R and X_W, respectively. R_X and W_X represent the read and write operations on the shared data item X. Furthermore, to distinguish transactions that belong to different mobile hosts, T_i^k represents a local transaction T^k at the mobile host MH_i.

5.5.2 The anchor transaction

Before a mobile host disconnects from the database servers, shared data is cached in the local workspace at the mobile host to support the disconnected processing of local transactions. The shared data item can be cached for read-only or updating. At the same time, these shared data can also be acquired by transactions at other mobile hosts; therefore, there is a potential conflict among shared data items that are cached in different local workspaces. For example, a shared data item X is modified by an offline transaction at the mobile host MH_i while it is being cached as read-only in the local workspace at the mobile host MH_j.

For each mobile host MH_i, there is a special online transaction called the *anchor transaction* T_i^A that plays role as a proxy transaction to local (i.e., offline) transactions at this mobile host (see Figure 5.15). The anchor transaction will be managed by the transaction manager at fixed database servers. The anchor transaction of a mobile host will: (1) request and hold all the granted locks of the shared data items that are being cached in the local workspace at the mobile host, and (2) keep track of the potential conflicting operations and dependencies among transactions in mobile environments.

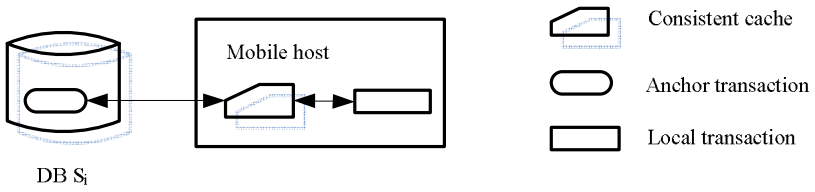

Figure 5.15: An anchor transaction in a mobile transaction processing system

The following discussion explains the operations of the anchor transaction T_i^A.

- *Requesting and holding locks.* Before the mobile host MH_i is disconnected, the anchor transaction T_i^A sends lock action requests to a database server S_i to acquire read or write locks on the set of shared data items that are needed for the disconnected transaction processing of local transactions T_i^j. If these lock requests are granted by the database server, the corresponding shared data items are cached in the local workspace at the mobile host. The set of granted locks will be held by the anchor transaction T_i^A. When the mobile host is disconnected from the database servers, the granted lock set will be replicated in the local workspace at the mobile host. A local transaction at the disconnected mobile host will acquire the corresponding read or write lock on a shared data item before its read or write operation on the shared data is carried out. Figure 5.16 illustrates this role of the anchor transaction. The local offline transactions T_i^1 and T_i^2 at the mobile host MH_i are considered sub-transactions of the anchor transaction T_i^A. For these local transactions, the transaction manager at the mobile host makes use of standard 2PL to ensure data consistency of the local workspace. Transactions T_i^1 and T_i^2 acquire the needed read and write locks, which are held by the anchor transaction at the database servers, before accessing the cached data item X. Note that the local transaction T_i^k can be either planned in advance or dynamically created.

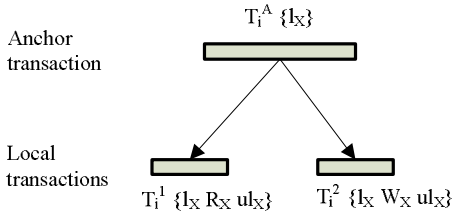

Figure 5.16: An anchor transaction acts as a proxy transaction

- *Keeping track of potential conflicting operations.* The anchor transaction is executed at the database server, and is managed by the fixed transaction manger at the database servers. The anchor transaction will not be forced to abort in any circumstance. This can be achieved by writing a log record for each anchor transaction at the database server, and if an anchor transaction fails, it will be restarted. While the mobile host is disconnected from the database server, the anchor transaction will keep track of potential conflicting operations that occur among transactions at different mobile hosts (i.e., read and write conflicting

operations). In Figure 5.17, before the mobile host MH_i is disconnected from the database servers, the anchor transaction T_i^A holds a read lock X_R on shared data item X. After this, the anchor transaction T_j^A of the mobile host MH_j acquires a write lock X_W on the data item X. Both anchor transactions T_i^A and T_j^A will keep track of the conflicting operations on shared data item X among transactions that are executed in the local workspaces at mobile hosts MH_i and MH_j. At this time, the local transaction T_i^1 at the being disconnected mobile host MH_i will not be aware of the conflict because this conflict occurs after the mobile host MH_i is disconnected from the database server. On the other hand, the local transaction T_j^1 at the mobile host MH_j will be aware of this conflict because this conflict occurs before the mobile host MH_j is disconnected from the database server. When the mobile host MH_i reconnects to the database servers, the transaction T_i^1 will be notified about the conflict via the conflict record held by the anchor transaction T_i^A. By this, the anchor transaction supports conflict awareness for offline transactions (see Section 6.3.4 for conflict awareness) by notifying the offline transactions about these potential conflicts when the mobile host reconnects to the database server.

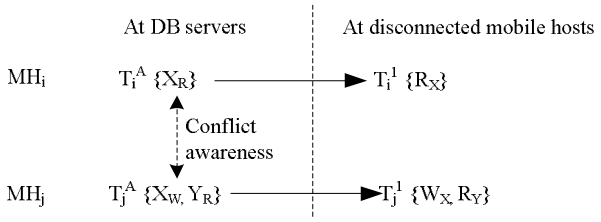

Figure 5.17: Anchor transactions support conflict awareness

Conflicting database operations can also happen when transactions at different disconnected mobile hosts share data status with each other. In Figure 5.18, before the disconnections of the mobile hosts, there is no conflict between the anchor transactions T_i^A and T_j^A. While being disconnected from the database servers, the delegator transaction T_i^1 at the mobile host MH_i shares the write lock X_W on the data item X to the delegatee transaction T_j^1 at the mobile host MH_j (see Section 5.5.5 for sharing data status). Therefore, the lock sets at the disconnected mobile hosts are changed and different from the initial lock sets held by the anchor transactions.

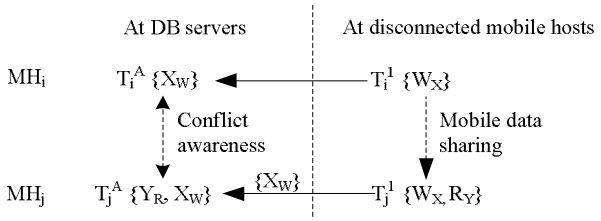

Figure 5.18: Conflict awareness caused by mobile data sharing

When the mobile hosts MH_i and MH_j reconnect to the database servers, the initial lock set hold by the anchor transactions T_i^A and T_j^A will be synchronized with the lock set at the mobile hosts to resolve any newly conflicting operations (it is not necessary that both the mobile hosts reconnect to the database servers at the same time, see Section 6.6). After this, the results of local transactions T_i^1 and T_j^1 will be integrated to the database servers. When the transaction integration stage is completed, the anchor transaction will commit.

The concept of proxy transactions (or pseudo-transactions) have been introduced and applied in mobile databases [HAA02]. However, our anchor transaction is different. There are four main differences between our anchor transaction and proxy transaction. First, the set of locks held by an anchor transaction can be modified when the mobile host disconnects from the database servers. Second, it is not necessary that an anchor transaction of a mobile host must always be created before the mobile host is disconnected from the database server (the proxy transaction must always be created before the disconnection of the mobile host). The reason is that this mobile host does not hold any shared data from the database servers at the beginning, but only receives shared data from other mobile hosts through the mobile sharing workspace (while being disconnected from the database servers). Third, the anchor transaction keeps track of potential conflicting operations among transactions at different local workspaces, i.e., supports conflict awareness among transactions in mobile environments. And fourth, the anchor transaction can support the mobility of transactions (explained in Section 5.7.3).

5.5.3 Distinguishing between sharing data states and sharing data status

In Chapter 4, we have surveyed several mobile transaction models that have been developed to support transaction processing in mobile environments. These mobile transaction models do not fully support the mobile data sharing among transactions at different mobile hosts (that are currently being disconnected from the database servers). For example, the mechanisms that support the sharing of data among transactions in mobile environments mainly focus on the sharing of data status (i.e., locks) via delegation operations [Chr93, Ram01] or additional lock modes [MB01]. We argue that a mobile transaction processing system must differentiate and support the sharing of both data state and data status.

In Figure 5.19, at the mobile host MH_1, the shared data item X is cached with read lock X_R. Local transaction T_2^1 at the mobile host MH_2, which cooperates with the transaction T_1^1, wants to read the shared data item X (the shared data item X is not cached at the mobile host MH_2). If the transaction T_1^1 is the only local transaction at the mobile host MH_1 to access the shared data item X (see Figure 5.19(a)), this transaction T_1^1 can delegate the read lock X_R of the shared data item X to transaction T_2^1, and the transaction T_2^1 will take control over the delegated lock X_R. However, if there is another transaction T_1^2 at the mobile host MH_1 that also needs to access the shared data item X (see Figure 5.19(b)), the transaction T_1^1 cannot delegate the read lock X_R on the shared data item X to the transaction T_2^1. Instead, the transaction T_1^1 can only let the transaction T_2^1 to view the state (i.e., the value V_X) of the shared data item X. The transaction T_2^1 can read the shared

data item X without holding the actual read lock X_R (we call this a *pseudo-read* operation). In other words, the anchor transaction T_2^A of the mobile host MH_2 does not hold a read lock on the shared data item X, but the local transactions at the disconnected mobile host MH_2 can perform read operations on this data item. This way, blocking of transactions at mobile host MH_2 is minimised.

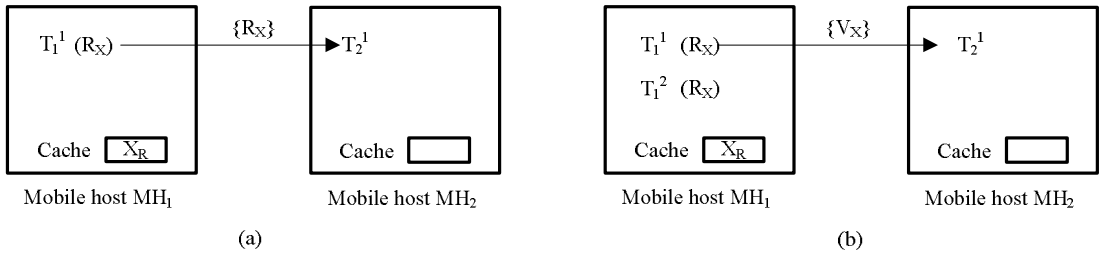

Figure 5.19: Sharing data status versus sharing data state

In mobile environments, we distinguish two types of mobile data sharing mechanisms (see Figure 5.20): (1) sharing data state and (2) sharing data status.

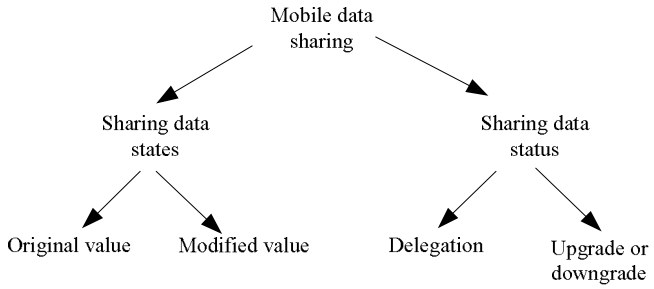

Figure 5.20: Mobile data sharing variants

For sharing data state, the shared value of the shared data item depends on the behavior of the delegator transaction (i.e., read-only or updating transaction) and the type of shared data item (i.e., with a read lock or write lock at the mobile host). If a delegator transaction is a read-only transaction or a shared data item is read locked at the mobile host, the delegator transaction can only share an original data value (i.e., non-modified) to a delegatee transaction. On the other hand, if a delegator transaction is an updating transaction, it can either share the original data value (i.e., before it is going to modify this shared data) or the updated data value (i.e., after it has modified the shared data) of a shared data item to a delegatee transaction.

For sharing data status, a delegator transaction can delegate locks on the shared data item to a delegatee transaction. Furthermore, we differentiate two sub-categories of sharing locks between transactions. First, the delegator transaction can completely relinquish its locks to the delegatee transaction. This means that the delegator transaction no longer holds any authority over the shared data, and the delegatee transaction will take full responsibility for the control of this shared data. Second, the delegator transaction can carry out a *downgrading* lock process to diminish its control over the shared data item

from a write to a read-only level. And the delegatee transaction can perform an *upgrading* lock process to raise the access right on the shared data item from the read to write permission. A detailed discussion on these types of mobile data sharing is presented in the following Sections 5.5.4 and 5.5.5.

5.5.4 Sharing data states

In this section, we focus on the issue related to sharing data state among transactions at different local workspaces.

For sharing data values, only the value of a shared data item is revealed to other delegatee transactions. The delegator transaction (to recap, the delegator transaction is a standard transaction that shares data to other delegatee transactions at different workspaces) must hold the lock of the shared data item. When a delegator transaction T_i^k at mobile host MH_i wants to share the value V_X of the data item X, it will initiate an export transaction $T_i^{k.E}$ that writes the value V_X into the export-import repository on behalf of transaction T_i^k. The export transaction $T_i^{k.E}$ is said to write on "behalf" of the delegator transaction because the transaction T_i^k still holds the read or write locks on the original data item. Delegatee transactions T_j^l at other mobile hosts MH_j are only allowed to read these shared values via corresponding import transactions. In other words, sharing data states are read-only.

A delegator transaction can share either an *original unmodified* data state or an *updated* data state. Table 5.7 summaries these sharing data state options.

Table 5.7: Locks and equivalent shared data state of delegator transactions

		Lock on X	
		Read	Write
Shared data state	Original value V_X	Relevant	Relevant
	Modified value $V_{X'}$	N/A	Relevant

If a delegator transaction T_i^k at mobile host MH_i holds a read lock on a data item X, the shared data value V_X will be identical to the value cached at the mobile host, i.e., the original data state is shared. However, if the delegator T_i^k at mobile host MH_i holds a write lock on data item X, the shared value V_X can be either an old value V_X (i.e., before the delegator transaction updates X) or an updated value $V_{X'}$ (i.e., after the delegator transaction has updated X). The shared data values that are exchanged between the delegator and delegatee transactions contribute to the transaction dependencies and execution constraints (see Section 6.5 for detail). Moreover, the delegatee transaction can either obtain the shared data value as a new shared data item; or, if it has already held the original data value V_X, it can modify its cached data to the up-to-date value $V_{X'}$.

A delegatee transaction can obtain the shared data value from the export-import repository via its import transactions. When the import transaction commits in the local workspace at the mobile host MH_j, the newly collected shared value V_X is read-only available to local transactions at this mobile host. A read operation on the shared value V_X in the local workspace at the mobile host MH_j is called a *pseudo-read* operation to

distinguish it from the "real" read operation that is preceded by a real read lock. This means that the database servers and the anchor transaction T_j^A of this mobile host do not know about these imported read-only data and pseudo-read operations until the mobile host reconnects to the database servers. A pseudo-read operation, therefore, allows an offline transaction to read a shared data before it can acquire the corresponding real read lock from the database server. This is one of the novel advantages of our mobile data sharing mechanism to increase the data availability in mobile environments.

To illustrate, Figure 5.21 presents a sharing data value scenario among three transactions at mobile hosts MH_1, MH_2 and MH_3. Data item X is acquired by a transaction T_1^1 at the mobile host MH_1. The value V_X is updated to $V_{X'}$ by this transaction and temporarily saved at this mobile host. The transaction T_1^1 shares this new value $V_{X'}$ to the export-import repository via an export transaction $T_1^{1.E}$. Similarly, delegator transaction T_2^1 at mobile host MH_2 shares the value V_Y via its export transaction $T_2^{1.E}$. Transaction T_2^1 at mobile host MH_2 also imports the shared data value $V_{X'}$ via its import transaction $T_2^{1.I}$. This means that transaction T_2^1 plays roles as both delegator and delegatee transactions. Delegatee transaction T_3^1 at mobile host MH_3 obtains the shared data values $V_{X'}$ and V_Y via its import transactions $T_3^{1.I1}$ and $T_3^{1.I2}$, respectively. The number of import transactions of transaction T_3^1 depends on the availability of the shared data items and mobile resources. For example, if both data items X and Y are available at the same time and the network bandwidth is adequate, one import transaction can be used to obtain both data values. Delegatee transaction T_3^1 and other local transactions at mobile host MH_3 can then pseudo-read these shared data values, i.e., without requesting corresponding read locks from the database server.

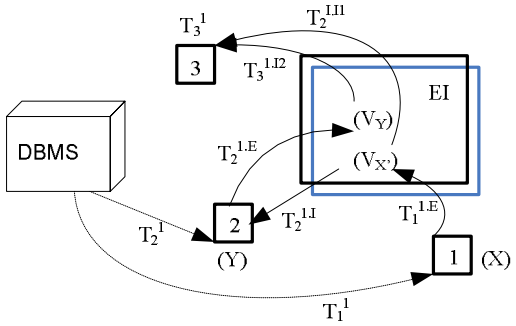

Figure 5.21: Sharing data states among transactions at different mobile hosts

5.5.5 Sharing data status

For sharing data status, a delegator transaction shares its locks on shared data to a delegatee transaction. Sharing lock is performed when a delegator transaction T_i^k at a mobile host MH_i wants to delegate its own read or write locks to a delegatee transaction T_j^l at a mobile host MH_j. The delegatee transaction T_j^l will take the responsibility to control the shared data.

In Section 5.2, we have illustrated the motivating mobile-IT scenario in which an IT-officer will try to solve a mobile task. In order for the mobile task to be performed, the artifacts related to the mobile task must be available to the IT officer. If the artifacts are not accessible, the IT-officer will not be able to carry out the mobile task. A mobile task can be considered a local transaction that is carried out in the local workspace at the mobile computer of the IT-officer. The shared artifacts are equivalent to the shared data items. In order for the local transaction to be carried out at the mobile host, the needed data must be available in the local workspace. In Figure 5.22, transaction T_1^i at mobile host MH_1 is in need of shared data item X, which is not cached in the local workspace. The data item is currently being cached and manipulated by the transaction T_2^j in the local workspace at mobile host MH_2. Transaction T_2^j, which holds the write lock X_W on X in the local workspace at mobile host MH_2, can delegate the access right of data item X, i.e., its write lock on X, to the transaction T_1^i.

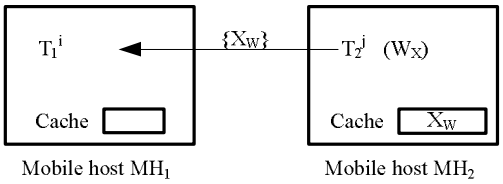

Figure 5.22: Sharing data status

Table 5.8 summaries the sharing of locks.

Table 5.8: Lock sharing

		Delegator transaction T_i^k shares	
	Lock type	Read	Write
Delegatee transaction T_j^l requests	Read	Allowed	N/A
	Write	N/A	Allowed

If the delegator T_i^k at the mobile host MH_i holds a read lock on the data item X, the export transaction $T_i^{k.E}$ will transfer the read lock into the export-import sharing workspace. A delegatee transaction T_j^l at the mobile host MH_j is allowed to obtain this delegated read lock.

The sharing of write locks can be further categorised into two sub-cases: (1) a delegator transaction delegates a write lock on a shared data item to a delegatee transaction; (2) a delegator transaction relinquishes only its write access right to a delegatee transaction but retains the read access right, i.e., downgrading the lock. A delegatee transaction can obtain this shared write lock as a new write lock in the local workspace. If this shared data is already cached read-only in the local workspace, the delegatee transaction can obtain this shared write lock to upgrade the access right of the shared data from read-only to updating, i.e., upgrading the lock.

If delegator transaction T_i^k at mobile host MH_i holds a write lock on data item X and wants to delegate this write lock, an export transaction $T_i^{k.E}$ will transfer the write lock on the data item on behalf of transaction T_i^k. A delegatee transaction T_j^l at mobile host MH_j can acquire the write permission on the shared data item by executing an import transaction $T_j^{l.I}$. There can be more than one delegatee transactions that compete for this write access right; however, only one delegatee transaction can successfully obtain the shared write lock on X. This condition ensures that the shared data item is only modifiable at one mobile host at any time. Note that the sharing data status among transactions occurs while the mobile hosts are disconnected from the database servers. This means that at the database servers the anchor transactions do not know about this sharing data status, i.e., the lock sets held by the anchor transactions and at the mobile host are inconsistent. When the mobile hosts reconnect to the database servers, the inconsistent lock sets will be reconciled (see Section 6.6.2). Because the delegator transaction T_i^k does not hold a write lock on the shared data item, the delegatee transaction T_j^l, which takes control over the shared data item, must take responsibility to finally integrate this shared data item into the database servers.

In Figure 5.23, delegator transaction T_1^1 at mobile host MH_1 shares the write permission on data item X to the export-import repository, and allows a delegatee transaction at another mobile host to continue updating this data item. In this case, export transaction $T_1^{1.E}$ releases the ownership on behalf of transaction T_1^1 on data item X. After this, delegatee transaction T_2^1 at mobile host MH_2 successfully obtains data item X with write lock via import transaction $T_2^{1.I}$, updates it to the new value $V_{X'}$, and finally integrates this value $V_{X'}$ into the database servers. Note that at this time at the database server, anchor transaction T_1^A of the mobile host MH_1 still holds the write lock on X, and anchor transaction T_2^A of the mobile host MH_2 does not hold this write lock on X. In other words, both anchor transactions T_1^A and T_2^A do not know about the sharing of write lock on X until the mobile hosts reconnect to the database servers. If mobile host MH_2 reconnects to the database server before mobile host MH_1, and the transaction T_2^1 is integrated, there will be a conflict. The reason is that both the anchor transaction T_1^A of mobile host MH_1 and the transaction T_2^1 at mobile host MH_2 hold write locks on data item X (see Section 6.6.2 for more detail of handling the conflicts).

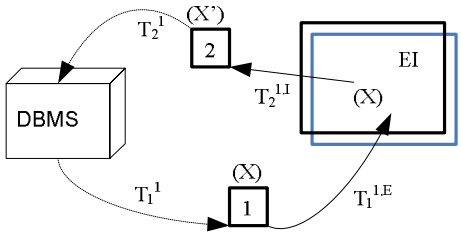

Figure 5.23: Sharing locks between standard transactions

If shared data item X is cached with write lock at the mobile host MH_1, but local transactions at this mobile host do not perform any updating operations (i.e., not

following execution plans), the write lock on shared data item *X* should be released so that a transaction at another mobile host can be carried out.

A delegator transaction carries out a *downgrading* lock procedure to diminish its control over the shared data item from a write to a read-only level. This means that the delegator transaction will relinquish its write permission on *X* but retains a read permission on *X*. This downgrading lock procedure allows another transaction to gain write access to the shared data item, i.e., reducing blocking time. Similarly, if delegatee transaction T_j^l already holds a read permission on shared data item *X*, it can upgrade its access right by obtaining a write lock on *X* from the delegator transaction (see Figure 5.24). Again, the anchor transactions are not aware of these upgrade or downgrade lock procedures at the disconnected mobile hosts. Therefore, in both cases, the corresponding lock conflicts must be taken care of (in Section 6.6.2 we will address how to handle these conflicting situations).

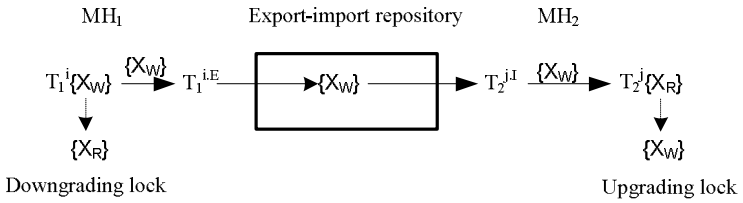

Figure 5.24: Downgrading and upgrading locks

5.5.6 Recursive sharing

A delegatee transaction T_j^l, which has successfully obtained a lock on a shared data item *X* from a delegator transaction T_i^k, can share data state V_X or a corresponding lock again with other transactions T_n^m. This sharing scenario is called *recursive sharing*. Moreover, such recursive sharing can happen in different export-import repositories, i.e., when the mobile host has participated in more than one mobile affiliation workgroup. Figure 5.25 illustrates a recursive sharing scenario. After standard transaction T_2^1 obtains a write lock on data item *X* from delegator transaction T_1^1 through the export-import repository EI_1, it updates the data item and shares the modified data item *X* (with the updated value $V_{X'}$) either back to the original repository EI_1 or to a new repository EI_2. In this case, standard transaction T_2^1 plays roles as both delegator and delegatee transaction.

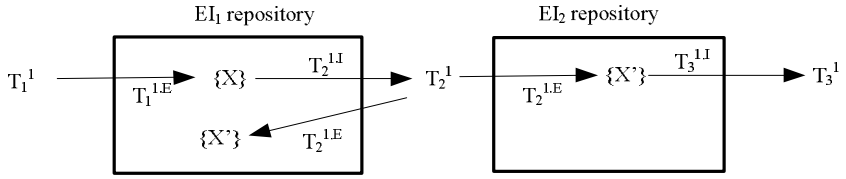

Figure 5.25: Recursive sharing

5.6 Management of mobile sharing workspaces

Sharing of data among mobile hosts in a mobile affiliation workgroup is carried out through the export-import repository. Shared data items are stored in a mobile sharing workspace that is distributed among mobile hosts (see Section 5.3.3). The management of the export-import sharing workspace consists of two parts as illustrated in Figure 5.26: (1) management of the physical export-import repository, and (2) management of the shared data in the mobile sharing workspace.

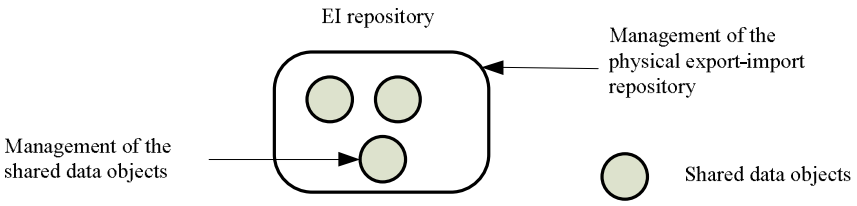

Figure 5.26: Management of a mobile sharing workspace

5.6.1 Managing the physical distribution of the export-import repository

An export-import repository is a mobile sharing workspace that supports data sharing among transactions at different mobile hosts that belong to a mobile affiliation workgroup. As we described in Section 5.3, this mobile sharing workspace is dynamically created, reconfigurable, and physically distributed among the involved mobile hosts. The management of the export-import repository structure includes the following functions as summarized in Table 5.9.

Table 5.9: Management of a mobile sharing workspace

Functions	Descriptions
Creating	Initiating a new mobile sharing workspace
Disposing	Destroying the current mobile sharing workspace
Expanding	Adding more storage capacity into the existing sharing workspace
Shrinking	Reducing the storage capacity of the sharing workspace
Merging	Joining export-import sharing workspaces into a larger one
Partitioning	Dividing a sharing workspace into several sub-workspaces

The above functions are elaborated as follows:

- *Creating* a new mobile sharing workspace. When a group of mobile hosts that belong to a mobile affiliation workgroup is in need of sharing data, a mobile sharing workspace is created. After this, an export-import repository is created, and standard transactions (i.e., delegator and delegatee transactions) at different mobile hosts can join the mobile sharing workspace and start sharing information.

- *Disposing* an existing export-import repository. When the collaborative work or the data sharing process among standard transactions is completed, the export-import repository of the mobile affiliation workgroup will be destroyed.

- *Expanding* the storage capacity of the existing export-import repository. As described in Section 5.3.4, the physical mobile sharing workspace is distributed among mobile hosts of the mobile affiliation workgroup. Therefore, the storage capacity of the export-import repository depends on the contribution of the involved mobile hosts. When a mobile host decides to contribute more storage space to the workgroup, this new storage space will be added to the current capacity of the export-import repository. The mobile sharing workspace can now accommodate more shared data items.

- *Shrinking* the current capacity of the existing export-import repository. A mobile host can withdraw its contributory sharing workspace from the mobile affiliation workgroup when it is leaving the mobile workgroup or it needs to scale down its operations due to the constraints of mobile resources. Thus, the sharing workspace capacity of the mobile affiliation workgroup is reduced, i.e., decreasing its storage capacity. This can have impact on the execution of current database operations that are accessing shared data items stored in this partition because these shared data items need to be re-allocated from the mobile sharing workspace (see Section 5.6.2).

- *Merging* several export-import repositories into a larger one. This procedure is performed when several collaborative mobile affiliation workgroups join together to form a larger mobile affiliation workgroup. The individual mobile sharing workspaces of each mobile affiliation workgroup will be combined together to benefit the mobile collaborative work, for example by allowing more shared data items in a larger export-import repository.

- *Partitioning* an existing export-import repository into smaller mobile sharing workspaces. This procedure is the inverse of the merging procedure described above. If a sub-group of mobile hosts that belong to a mobile affiliation workgroup is going to be temporarily disconnected from the original workgroup (and these mobile hosts will continue to collaborate), the existing mobile sharing workspace will be partitioned into several smaller sharing workspaces for the new sub-workgroups.

5.6.2 Data management in the export-import repository

Due to the changes in capacity (i.e., expanding and shrinking) and in organization (i.e., merging and partitioning) of an export-import repository, the management of shared data that resides in the mobile sharing workspace consists of following functions: adding, removing and moving (see Table 5.10 for a summarization).

Table 5.10: Management of shared data items in a mobile sharing workspace

Functions	Descriptions
Adding	Placing new shared data items into the sharing workspace
Removing	Withdrawing shared data items from the sharing workspace
Moving	Changing the storage location of shared data items

The following discussion explains the management functions of the shared data items in a mobile sharing workspace:

- *Adding* new shared data items into the mobile sharing workspace. The adding function provides an interface to an export transaction to place a new shared data item into the mobile sharing workspace. The adding function can also replicate shared data items in an export-import repository to increase the level of data availability. For example, the shared data items can be duplicated when more storage workspace is available (i.e., when the capacity of the export-import repository is expanded) or when the export-import repository is split into sub-workspaces (i.e., when the mobile affiliation workgroup is partitioned into sub-workgroups).

- *Removing* shared data items from the mobile sharing workspace. A shared data item that is currently stored in an export-import repository will be removed in several circumstances. First, the shared data item is removed when it is no longer needed (i.e., the mobile data sharing is completed). Second, when a delegator transaction wants to withdraw its shared data, the shared data item will be removed from the mobile sharing workspace. Third, the shared data item may be removed when the export-import repository does not have storage capacity to accommodate all the shared data items. Removing may also be carried out when the capacity of the export-import repository is decreased, i.e., shrinking.

- *Moving* the physical storage location of shared data items to a new location. When the capacity or the organization of the export-import repository is changed or reconfigured, some of the shared data items in the mobile workspace will be re-allocated among mobile hosts. For example, when a mobile host is about to disconnect from the mobile affiliation workgroup, the shared data items that are currently stored in its sharing workspace partition will be moved to other available locations (at other mobile hosts) in the mobile affiliation workgroup. This will avoid interrupting the execution of transactions that are accessing these shared data items. Moving includes two sequential steps: (1) adding the shared data item to a new storage location, and (2) removing the shared data item from the old storage location.

5.7 Management of transaction execution behavior

Our mobile transaction processing system includes two types of transaction: standard transaction and shared transaction. Standard transactions (i.e., delegator and delegatee transactions) are executed in the local workspaces at the mobile hosts and integrated in

the global workspace. Shared transactions (i.e., export and import transactions) are initiated by standard transactions to support mobile data sharing among these standard transactions. Both standard and shared transactions can be either planned in advance or generated at runtime (see the discussion of the mobile task characteristics in Section 5.2.1). The behavior of shared transactions determines the successfulness of the mobile data sharing among standard transactions. For example, if either an export or import transaction fails, the mobile data sharing process between the delegator and the delegatee will not be carried out. Furthermore, due to the movement of the mobile host from one mobile cell to another, the mobile transactions which are executed at the mobile hosts are also moved. To support and manage the execution of transactions in mobile environments, the management of the transaction execution behavior in our mobile transaction processing system contains three parts: *execution dependency*, *structural dependency* and *mobility manager*.

- ***Execution dependency.*** Control and manage the effects of the termination of transactions on other transactions, for example the abortion or commitment effect of the delegator transactions upon the delegatee transactions.

- ***Structural dependency.*** Control and manage the init, commit and abort operations of transactions; and support transaction restructuring operations like split, join and adopt.

- ***Mobility manager.*** Control and manage the mobility of transactions when mobile hosts are moving across mobile cells or participating in different mobile affiliation workgroups.

The following sub-sections discuss the management of the transaction execution behavior.

5.7.1 Managing the execution dependency

The execution dependency among transactions consists of two types of dependencies: static dependency and dynamic dependency. The static dependencies support the mobile transaction processing system to enforce the strict relationships among transactions. The dynamic transaction dependencies allow the mobile transaction processing system to dynamically determine the dependencies between transactions in accordance with their interactions and execution progress.

Static dependency

A static dependency can be either planned beforehand or initiated at runtime, and cannot be changed. There are two categories of static transaction dependencies: (1) abort dependency, and (2) commit dependency. An abort dependency identifies what transactions must be aborted when a related transaction is aborted. For example, if a delegator transaction T_i^k that shares an intermediate data value aborts, those delegatee transactions T_j^l that have read the shared data values must be aborted. On the other hand,

when a transaction commits, a commit dependency determines the commitment order that the involved transactions must follow to assure consistency of the data.

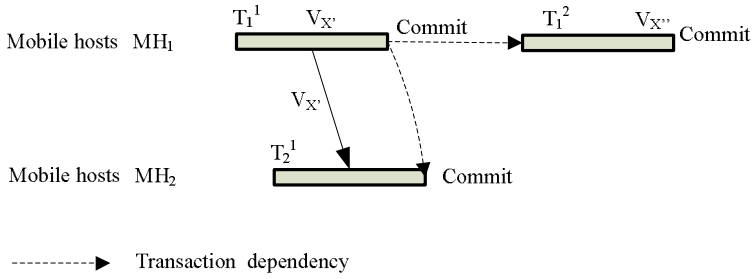

Figure 5.27: Static transaction dependencies

In Figure 5.27, delegator transaction T_1^1 at mobile host MH_1 shares the updated data value $V_{X'}$ to delegatee transaction T_2^1 at mobile host MH_2. At a later time, transaction T_1^2 at mobile host MH_1 continues to update this shared data item X to new value $V_{X''}$. In this scenario, transaction T_2^1 must be scheduled after T_1^1 and before T_1^2. Such strict schedules can only be guaranteed by the support of an explicit static dependency between the transactions T_1^1 and T_2^1. Note that transaction T_2^1 at mobile host MH_2 does not know about transaction T_1^2 at mobile host MH_1 until both the mobile hosts synchronize their local transactions at the database servers.

Dynamic dependency

A dynamic dependency is modifiable at runtime. Dynamic dependencies are essential to transactions in mobile environments in order to cope with long disconnections and unexpected termination of related transactions. The dynamic dependencies among transactions are also used when it is necessary to change a transaction execution schedule.

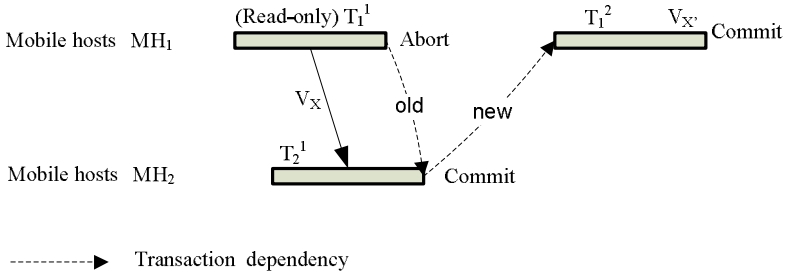

Figure 5.28: Dynamic transaction dependencies

In Figure 5.28, read-only delegator transaction T_1^1 at mobile host MH_1 shares the original data value V_X to delegatee transaction T_2^1 at mobile host MH_2. At a later time, transaction T_1^2 at mobile host MH_1 modifies this shared data item X. In this case, transaction T_2^1 must be scheduled after T_1^1 and before T_1^2. When transaction T_1^1 aborts (and due to the disconnection between the two mobile hosts, transaction T_2^1 does not know about this abortion until the transaction integration stage – see Section 6.7), the commit dependency

between the transactions T_1^1 and T_2^1 is no longer valid. However, the transaction T_2^1 should not be aborted because it has not read an inconsistent data value from the transaction T_1^1, i.e., V_X is consistent. As a result, the mobile transaction processing system must provide mechanisms to deal with an unexpected abortion of transaction T_1^1. In this case, a new dynamic transaction dependency between transactions T_2^1 and T_1^2 will be defined so that transaction T_2^1 can commit and be scheduled before transaction T_1^2.

5.7.2 Managing the structural dependency

Shared transactions are initiated on demand of the data sharing among standard transactions. The execution behavior of shared transactions depends on the structure (i.e., flat or nested) of the standard transactions (as discussed in Section 5.4.2). Furthermore, participation of a mobile host in many mobile affiliation workgroups leads to involvement of transactions across several mobile sharing workspaces. Consequently, the execution of shared transactions will be affected when their corresponding standard transactions move from one mobile sharing workspace to another. Therefore, the mobile transaction processing system must handle both primitive transaction operations such as initiate, commit or abort [CR94], and transaction re-structuring operations like split, join, or adopt (see Table 5.11).

Table 5.11: Management of transaction behavior

Operations	Descriptions
Initiate	Setting up a new shared transaction
Commit/Abort	Triggering the execution or termination of related transactions
Split	Breaking up shared transactions into sub-transactions
Join	Merging a shared transaction into another shared or standard transaction
Adopt	Integrating a shared transaction as a sub-transaction in a nested standard transaction

As discussed in Section 5.4.3, the structure of a standard transaction has a strong impact on the creation of shared transactions. If a standard transaction is a flat transaction, it can initiate a new export transaction. If a standard transaction is a sub-transaction of a nested transaction, it can ask the parent transaction to initiate an export transaction (see Section 6.4.3 for more detail).

An export transaction supports a delegator transaction to share data via an export-import repository. When the export transaction commits, related import transactions, which are waiting for the shared data, will be triggered and start executing (the export transaction fulfills the isolation property of transactions as we have addressed in Section 5.4.2). If the shared data is withdrawn and the export transaction is compensated, these related import transactions will be aborted (if they have not committed) or compensated (if they have committed). Due to the relaxation of the isolation property of import transactions, the shared data, which is obtained by an import transaction from the export-import repository, will be made available to all local transactions at the mobile host before the

import transaction commits in the local workspace. However, the original delegatee transaction that initiated this import transaction may impose restrictions on these shared data so that the collected data will be accessed by the delegatee transaction before by any other local transaction. For example, if the delegatee transaction has a flat structure, the import transaction must be joined into the delegatee transaction; and the shared data is available to local transactions after the delegatee transaction commits. If the delegatee transaction has a nested structure, the import transaction will be adopted as a sub-transaction of the delegatee transaction. In this case, the collected shared data is available to other local transactions after the top-level transaction of the hierarchical structure commits. When standard transactions are integrated to the database servers, the commitment or abortion of a delegator transaction can trigger the commitment or abortion of related delegatee transactions (see Section 6.6 for further detail).

Due to the availability of shared data items, the structure of a shared transaction can be dynamically changed. For example, if a delegatee transaction wants to obtain a set of shared data, it can issue an import transaction to carry out this job. However, all the needed information might not be available at that time or not be accessible in one mobile sharing workspace. Instead of waiting for these shared data items to be available, the import transaction can be *split* into several (sub)-import transactions that can collect the different shared data items in the mobile sharing workspaces.

Furthermore, during the execution of shared transactions, a mobile host can change from one mobile affiliation workgroup to another. This results in changes of the mobile sharing workspaces that the mobile host is participating in. Consequently, a shared transaction changes its operating environment, i.e., from one export-import repository to another. For example, if a delegatee transaction moves to a new mobile sharing workspace, the current active import transaction in the old mobile sharing workspace will be *split* into two sub-import transactions. The first sub-import transaction can either (1) continue executing in the old mobile sharing workspace if it has not completed its assigned operations, or (2) commit in the local workspace and make the already collected shared data visible to local transactions. The second sub-import transaction will start operating in the new mobile sharing workspace. If the delegatee transaction later re-joins back to the previous export-import repository, the split sub-import transactions will be *joined* together.

5.7.3 Managing the mobility of transactions

In this section, we discuss how our mobile transaction processing system supports the mobility of transactions. We differentiate two mobility patterns in relation to the movement of mobile hosts: (1) the mobile hosts are moving across different mobile cells; and (2) the mobile hosts are moving across different mobile affiliation workgroups. The main distinguishing characteristic between these two mobility patterns is: the standard hand-off or hand-over processes [SRA04] do not happen when the mobile host is moving across mobile affiliation workgroups. The movement of the anchor transaction supports the mobility of local transactions across different mobile cells; while the shared transactions assist the mobility of standard transactions across different mobile sharing

workspaces when the mobile host is moving across different mobile affiliation workgroups (see Table 5.12).

Table 5.12: Management of transaction mobility

Mobility patterns of the mobile host	Handling the mobility of transactions
Across mobile cells	By the movement of anchor transactions
Across mobile affiliation workgroups	By the dynamic re-structuring of shared transactions

Mobility of transactions across mobile cells

The location of the mobile host is identified by the identity of the mobile cell the mobile host stays within. In the new mobile cell, the mobile host must be able to contact the mobile support station MSS_{ID} of the mobile cell in order to determine its new location and to communicate with other hosts (see the architecture of the mobile transaction environment in Section 3.4). In our mobile transaction processing system, the anchor transaction of each mobile host will support the movement of the mobile hosts. The anchor transaction resides at the wired network, i.e., at the mobile support stations or at the database servers. These mobile support stations or database servers are the anchor points of the anchor transactions. When the mobile host moves into a new mobile cell, a hand-over process will be performed so that the anchor transaction will be moved from the previous anchor point to the new one. In Figure 5.29, when mobile host MH_i moves from the mobile cell MC_n to the new mobile cell MC_m, the hand-over process will move the anchor transaction T_i^A from the mobile support station MSS_n to MSS_m. The anchor transaction T_i^A will keep track of the mobile support stations that it is moving across, i.e., MSS_n and MSS_m, and therefore, support the mobility of transactions across different mobile cells.

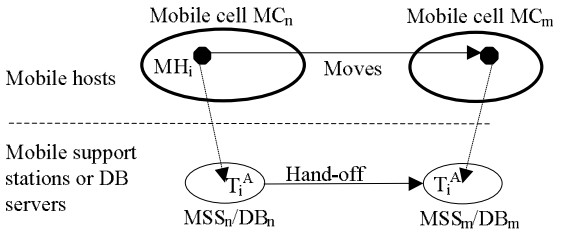

Figure 5.29: Mobility of transactions across mobile cells

Compared to other hand-over mechanisms [DHB87, KK00, MB01], our hand-over mechanism has two main advantages. First, the hand-over process is actively initiated by the mobile host. As we have discussed in Chapter 3, the hand-over process is not necessary if the transactions are local and processed entirely at the mobile host. In other words, in our mobile transaction processing system, the hand-over process is only performed when it is needed. Second, a mobile host can be aware of the movement of the neighbouring mobile hosts. The residence of anchor transactions at an anchor point represents the mobile hosts that are currently staying in the same mobile cell. When a

mobile host moves to a new mobile cell, it can inform other mobile hosts about its new location.

Mobility of transactions across mobile affiliation workgroups

When being disconnected from the database servers, a mobile host can participate in several mobile affiliation workgroups MA_i. Consequently, standard transactions at the mobile host share data through several export-import repositories EI_i. When a standard transaction is leaving an old export-import repository and joining a new export-import repository, the associated shared transactions of this standard transaction will be transferred to the new export-import repository.

By keeping track of the mobile affiliation workgroups MA_{ID} and the export-import repositories EI_{ID}, the mobile transaction processing system can handle the movement of standard transactions across different export-import repositories. The transfer of shared transactions across different export-import repositories is achieved by applying the split and join operations described in Section 5.7.2. In Figure 5.30, when mobile host MH_i moves from mobile affiliation workgroup MA_k to MA_l, import transaction $T_i^{k.I}$ of standard transaction T_i^k will be moved from the mobile sharing workspace EI_k to EI_l. The import transaction $T_i^{k.I}$ will be split into two sub-import transactions $T_i^{k.I1}$ and $T_i^{k.I2}$. The sub-import transaction $T_i^{k.I1}$ will continue executing in the export-import repository EI_k, while the sub-import transaction $T_i^{k.I2}$ will start executing in the new export-import repository EI_l. When mobile host MH_i is re-joining the mobile affiliation workgroup MA_k, the two sub-import transactions $T_i^{k.I1}$ and $T_i^{k.I2}$ will be joined together.

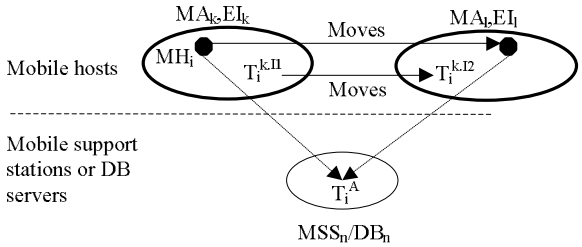

Figure 5.30: Mobility of transactions across mobile affiliation workgroups

5.8 Conclusions

In this chapter, we have presented our approach to develop a mobile transaction processing system. The main contribution is the new horizontal collaboration model to support collaborative work in mobile environments. The fundamental idea is to support disconnected mobile hosts to form dynamic mobile affiliation workgroups by taking advantage of wireless communication technologies. This way the mobile hosts can continue carrying out their cooperative work while being on the move and without any support from non-mobile database servers. Our data sharing mechanism enhances the data sharing in mobile environments by supporting different types of data sharing: sharing data states and sharing data status. The mobility of transactions is handled via the movement of anchor transactions and the dynamic restructuring of shared transactions.

Moreover, the anchor transactions also support the mobile transaction processing system in handling conflict awareness among transactions at different mobile hosts.

Our mobile transaction processing system is appropriate for mobile environments because it takes into account the mobility of computing hosts (via mobile affiliation workgroups), the low bandwidth and disconnections of wireless networks (by separating shared transactions from standard transactions), and the limitation of mobile computing resources (via the distribution of export-import repositories).

Chapter 6
Formalizing the Mobile Transaction Processing System

In this chapter, we formalize the mobile transaction processing system that has been presented in Chapter 5. We formally describe in detail the operations of the mobile transaction processing system that includes four different stages: (1) the data hoarding stage, (2) the mobile data sharing stage, (3) the disconnected transaction processing stage, and (4) the transaction integration stage. We also formalize operations that manage the mobility and the dependency of transactions in mobile environments.

6.1 Introduction

Chapter 5 has presented and discussed the mobile transaction processing system that focuses on supporting mobile data sharing among transactions at different mobile hosts. This chapter formally addresses in detail the operations of the mobile transaction processing system.

The lifespan of a mobile transaction process can be divided into four main stages: (1) the data hoarding, the mobile data sharing, the disconnected transaction processing, and the transaction integration (see Figure 6.1). These four different stages of the mobile transaction processes are not necessarily to be carried out in that sequential order. When the mobile host is disconnected from the database servers, transactions are locally executed in the local workspaces at the mobile hosts. The mobile host can also join mobile affiliation workgroups and share data with other mobile hosts. When the mobile hosts connect to the database servers, the mobile hosts can perform either the data hoarding or the transaction integration or both. The data hoarding and the mobile data sharing stages support the disconnected processing stage. The transaction integration stage assures the data consistency in global workspace after the disconnected transaction processing stage.

Data hoarding stage. In order to support the disconnected transaction processing, before the mobile host is disconnected from the database servers, necessary data must be cached in the local workspace at the mobile host. During the data hoarding phase, consistent shared data that is stored at the database servers is downloaded into the local storage of

the mobile host with the support of the anchor transaction (to recap, the anchor transaction plays a role as a proxy transaction to all local transactions that are disconnectedly processing in the local workspace of the mobile host). The amount of information that can be stocked in the local storage at the mobile hosts depends on several factors. First, the storage capacity of a mobile host determines the upper bound of the amount of information that could be locally stored at the mobile host. Second, the actual amount of information that can be downloaded is also affected by the bandwidth of the wireless networks and the connection period of the data hoarding phase. If the data hoarding interval is short, the mobile host may not be able to fully cache the needed data (because the amount of transferred data from the database servers to the mobile host is proportional to the network bandwidth and the connection time). Third, the most interesting issue of this data hoarding stage is which shared data items are allowed to be cached at the mobile host without causing any data inconsistency with other mobile hosts. In other words, we have to answer the question: how to avoid or be aware of conflicts among transactions at different disconnected mobile hosts.

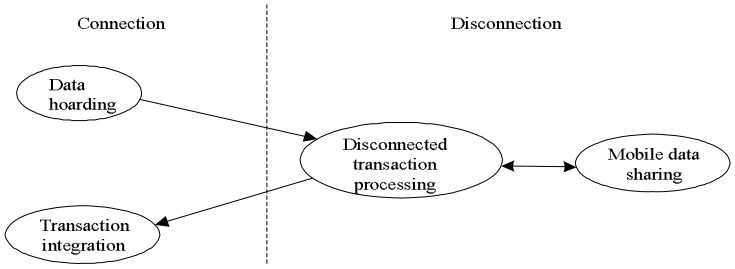

Figure 6.1: Stages of mobile transaction processes

Mobile data sharing stage. While being disconnected from the database servers, a mobile host can join mobile affiliation workgroups and directly share information with other mobile hosts. This means that the database servers are not aware of these mobile data sharing processes. The mobile data sharing operations are carried out through the export-import repositories with the support of the export and import transactions. The sharing of mobile information includes both sharing data states (i.e., data values) and data status (i.e., locks). Shared data can be either consistent cached data or partial results of locally committed transactions.

Disconnected transaction processing stage. When the mobile host is disconnected from the database servers, local transactions at the mobile host are carried out based on the cached data. The locally cached data can be either the original consistent data that is hoarded at the data hoarding stage, or the exchanged data that is obtained in the mobile data sharing stage. Therefore, the cached data can be either fully consistent or temporarily inconsistent. Local transactions are allowed to locally commit in the local workspaces at the mobile hosts, and the locally committed results will be made available to other local transactions.

Transaction integration stage. When the mobile hosts reconnect to the database servers, integration processes are performed to ensure that the global data consistency is fulfilled. In this stage, the locally committed transactions will be evaluated against other transactions to determine the global transaction execution schedule (that can be serializable schedule or user defined schedule). If there is a conflict that cannot be resolved, one or more locally committed transactions will be aborted; otherwise the locally committed transactions will be allowed to finally commit at the database servers.

The rest of this chapter is organized as follows. Section 6.2 formalizes the concept of mobile transactions and the management of mobile transaction dependencies. The operations of the mobile transaction processing system that includes the data hoarding stage, the mobile data sharing stage, the disconnected transaction processing stage, and the transaction integration stage will be formalized in Section 6.3, 6.4, 6.5 and 6.6, respectively. In Section 6.7, we formalize operations that manage the mobility and the dependency of transactions in mobile environments. Section 6.8 concludes the chapter.

6.2 Management of mobile transaction dependencies

In this section, we present the concepts of mobile transactions and formalise the management of transaction dependencies among mobile transactions. To recap, we distinguish two types of mobile transactions: (1) the standard transaction, and (2) the shared transaction. The standard transactions, i.e., delegator and delegate transactions, are transactions that are locally executed in the local workspaces at the disconnected mobile hosts. The shared transactions, i.e., export and import transactions, are transactions that support the standard transactions to share information. To ease the following discussion, in this section, let T^{Dor}, T^{Dee}, T^{E}, and T^{I} denote the delegator, delegatee, export and import transactions, respectively.

Definition (*transaction*). *A transaction T^i is a partially ordered set with a partial order relation $<_i$ where:*

- $T^i \subseteq \{R_X, W_X \mid X \text{ is a shared data item}\} \cup \{c,a\}$
- $\forall R_X, W_X \in T^i$, *either* $R_X <_i W_X$ *or* $W_X <_i R_X$
- $c \in T^i$ *iff* $a \notin T^i$
- $\forall Op \in T^i$, $Op \notin \{c,a\}$, *either* $Op <_i a$ *or* $Op <_i c$

Definition (*mobile transaction*). *A mobile transaction is a tuple of $(\Im_E, T^M, \Im_I)$ where:*

- T^M *is the transaction that is being locally performed at the mobile host.*
- $\Im_E$ *is the set of export transactions T^E associated with the standard transaction T^M.*
- $\Im_I$ *is the set of import transactions T^I associated with the standard transaction T^M.*

A *delegator transaction* T^{Dor} is a mobile transaction that only exports its shared data to other transactions, i.e., $\Im_E \neq \varnothing \wedge \Im_I = \varnothing$. A *delegatee transaction* T^{Dee} is a mobile transaction that only obtains data from other transactions, i.e., $\Im_E = \varnothing \wedge \Im_I \neq \varnothing$.

The export and import transactions are initiated by the delegator and delegatee transactions, respectively. The shared transactions can be specified in advance or created during the execution of the standard transactions. Figure 6.2 illustrates the possible interactions among these shared and standard transactions. To recap, the export transaction fully meets the standard ACID transaction properties; hence, the associated import transaction is triggered when the export transaction commits in the mobile sharing workspace. The isolation property of the import transaction can be relaxed, i.e., the delegatee transaction can view the intermediate results of the import transaction.

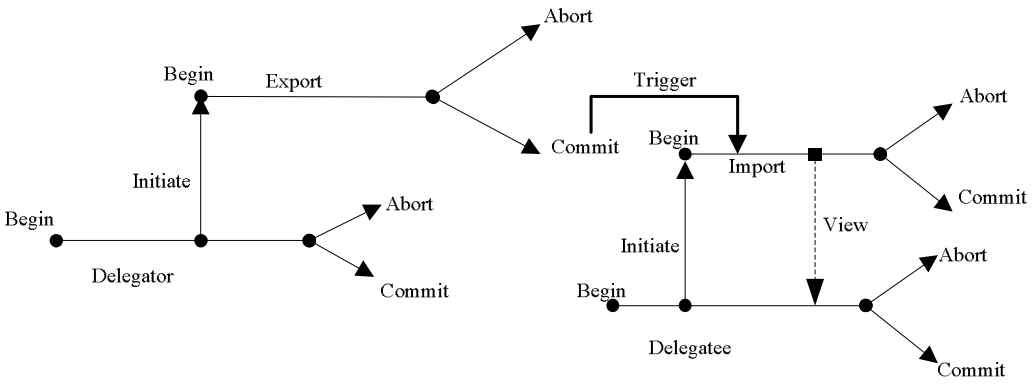

Figure 6.2: Interactions of standard and shared mobile transactions

We differentiate two types of transaction dependency: (1) structural transaction dependency, and (2) execution constraint dependency. The structural transaction dependency focuses on the effect of the abortion of one transaction on others; while the execution constraint dependency focuses on the execution order of committed transactions. Figure 6.3 illustrates the possible dependencies among transactions.

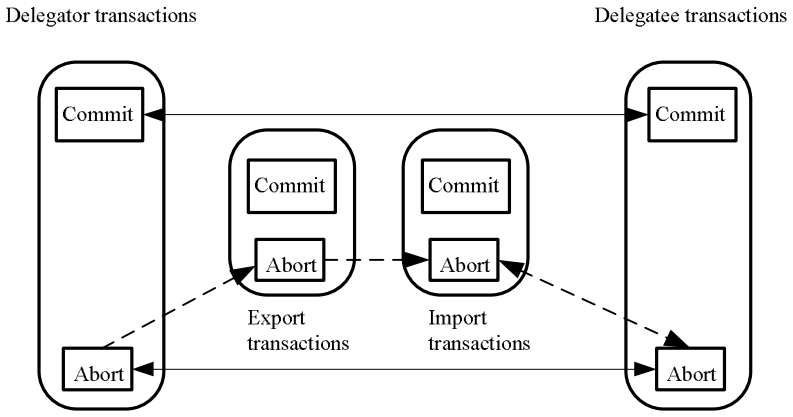

Figure 6.3: Transaction dependencies

[CR94] defined the ACTA transactional framework for reasoning about and synthesising the dependencies among transaction. In our mobile transaction processing system, we will reuse the *commit-dependency* and the *abort-dependency* rules from the ACTA transactional framework. In addition, we define a new structural transaction dependency

rule, called *multiple-abort-dependency*, which provides a flexible way to characterize the structural transaction dependency among mobile transactions. The following sub-sections discuss these two types of transaction dependency and the operations for managing the dependencies among mobile transactions.

6.2.1 The transaction dependencies

There are two types of abort dependency among mobile transactions: the abort-dependency and the multiple-abort-dependency. The following discussion will address the usage of these transaction abort dependencies:

- **Abort-Dependency (T^i *AD* T^j)**: *if transaction T^i aborts and transaction T^j has not committed, then T^j aborts. If transaction T^j has committed then it is compensated.*

  The usages of the abort-dependency rule are summarized in Table 6.1. The transaction abort dependencies can be categorised into three parts: (1) the dependency between delegator and delegatee transactions in the global workspace (rule AD1), (2) the dependency between the standard transaction and the associated shared transactions in the local workspace (rules AD2 and AD3), and (3) the dependency between shared transactions in the mobile sharing workspace (rule AD4). Depending on the actual interactions between standard and shared transactions (see discussion in Section 5.4.3), the abort-dependency between each pair of interactive transactions must be explicitly defined.

Table 6.1: Transaction abort-dependencies

Rules	Relation of T^i and T^j	Descriptions
AD1	T^{Dor} *AD* T^{Dee}	Abort dependency between delegator and delegatee transactions in the global workspace
AD2	T^{Dor} *AD* T^{E}	Abort dependency between the delegator and its export transactions
AD3	T^{Dee} *AD* T^{I}	Abort dependency between the delegatee and its import transactions in the local workspace
	T^{I} *AD* T^{Dee}	
AD4	T^{E} *AD* T^{I}	Abort dependency between shared transactions in mobile sharing workspaces

The above four abort-dependency rules represent the abort dependency among transactions in the horizontal collaboration dimension (see Section 5.4.1). The first rule *AD1* specifies the correlation between a delegator transaction and a delegatee transaction in the global workspace. If the delegator transaction aborts, the delegatee transaction that has read shared data from this delegator transaction must also abort. However, the abortion of the delegatee transaction could be delayed until the transaction integration stage due to the disconnections of the mobile hosts (see Section 6.6). Therefore, when a delegator transaction aborts, the mobile host will

have to keep the records of the aborted delegator transaction so that this information can be propagated to the associated delegatee transactions at later time (see Section 6.5.4).

The rule *AD2* specifies the correlation between the delegator transaction and its export transactions. If the delegator transaction aborts, and the data shared by this delegator transaction can become invalid, hence, the associated export transactions must be aborted. If these export transactions had committed in the mobile sharing workspace, they will be compensated to ensure that no invalid information is shared. It is not necessary that all the correlated export transactions must be aborted because the delegator transaction could have shared consistent data, for example consistent read-only data. Therefore, the abort-dependency between the delegator and each of its export transactions must be explicitly defined.

The rule *AD3* specifies the relationship among the delegatee transaction and its import transactions. And there are two applicable instances of this rule: $(T^{Dee}\ AD\ T^{I})$ and $(T^{I}\ AD\ T^{Dee})$. For the first instance, if the delegatee transaction aborts, its import transactions will abort because the shared data is no longer needed. For the second instance, if the import transaction aborts, the delegatee transaction will abort because the obtained data is invalid.

The rule *AD4* defines the association between the export transaction and the import transactions that have read the shared data written by the committed export transaction in the mobile sharing workspace. If the export transaction is compensated due to the invalidation of the shared data (see rule *AD2*), these import transactions must be aborted. If these import transactions had committed, they are compensated.

- **Multiple-Abort-Dependency ($\mathcal{T}_i$ *MD* T^j):** *if a set of transactions* $\mathcal{T}_i = \{T^i, i>1\}$ *aborts, then transaction* T^j *aborts.*

The usages of the multiple-abort-dependency rule are summarized in Table 6.2.

Table 6.2: Transaction multiple-abort-dependencies

Rules	Relation of $\mathcal{T}_i$ and T^j	Descriptions
MD1	$\{T^{Dor}\}$ *MD* T^{Dee}	Abort dependency between a set of delegator transactions and a delegatee transaction in the global workspace
MD2	$\{T^{I}\}$ *MD* T^{Dee}	Abort dependency between a set of import transactions and a delegatee transaction in the local workspace

The two multiple-abort-dependency rules support the mobile transaction processing system to avoid the problem of unnecessary aborts of delegatee transactions. For example, a delegatee transaction can initiate many import transactions to obtain shared data items in many export-import repositories. The delegatee transaction can

develop abort dependencies with many delegator transactions. However, an abortion of a delegator transaction or an import transaction must not cause the entire delegatee transaction to abort. In Figure 6.4, the delegatee transaction T^3 is only aborted if both delegator transactions T^1 and T^2 are aborted. The main difference between these rules is that: (1) the multiple-abort-dependency between the standard transactions, i.e., rule *MD1*, is applied in the global workspace and is evaluated at the transaction integration stage (see Section 6.6); and (2) the multiple-abort-dependency between a delegatee transaction and its import transactions, i.e., rule *MD2*, is applied in the local workspace at the disconnected mobile host.

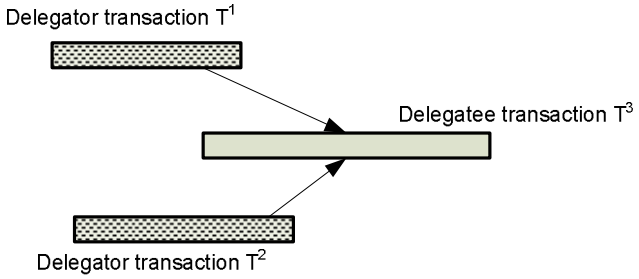

Figure 6.4: Multiple abort dependency

The abort-dependency and multiple-abort dependency allow the mobile transaction processing system to specify the correlation among the standard and shared transactions in accordance with their interactions. In the transaction integration stage, the abort-dependency will be checked before the multiple-abort-dependency (see Section 6.6.2).

6.2.2 The execution constraint

The transaction execution constraint dependency is applied when the mobile transactions are preparing to commit in the global workspace. To ensure that the states of the database are fully consistent and recoverable, the mobile transaction processing system must enforce the order of transaction commitments:

- **Commit-Dependency (T^i *CD* T^j)**: *if both transactions T^i and T^j commit, then T^i must commit before T^j.*

  The usage of the commit-dependency rule is summarized in Table 6.3.

Table 6.3: Transaction commit-dependencies

Rules	Relation of T^i and T^j	Descriptions
CD1	T^{Dor} *CD* T^{Dee}	Commit dependency between the delegator and delegatee transactions in the global workspace

The rule *CD1* specifies the order of commitment between the delegator and delegatee transactions. When a delegator transaction shares an updated data state to a delegatee transaction, the delegator transaction must commit before the delegatee transaction in order to achieve recoverability.

6.2.3 Managing transaction dependencies and execution constraints

The usage of the mobile transaction dependencies depends on the progress of the execution processes and the interactions among mobile transactions. Therefore, the mobile transaction processing system must provide the following operations to support the management of the transaction dependencies. When a transaction dependency or an execution constraint is defined, an appropriate operation will be executed to register the specified rule in the mobile transaction processing system. These operations are described as follows:

- CreateDependency(T^i, T^j, dependency_rule, dependency_type): This method initiates a new transaction *dependency_rule* between two transactions T^i and T^j. This newly created transaction dependency rule can be either an abort-dependency or a commit-dependency. The *dependency_type* is either static or dynamic dependency (see Section 5.7.1).

- RemoveDependency(T^i, T^j, dependency_rule): This method removes an existing transaction *dependency_rule* between two transactions T^i and T^j. This allows the mobile transaction processing system to dynamically define the correlations among mobile transactions that are being executed at the mobile hosts. If the *dependency_rule* is a static rule, it cannot be removed unless the involved transactions are aborted.

- TemporaryDisableDependency(T^i, T^j, dependency_rule): This method deactivates an active transaction *dependency_rule* between two transactions T^i and T^j. This operation is used in a mobile data sharing scenario in which a mobile transaction does have many options to interact with other mobile transactions (see illustration in Figure 5.8).

- ReEnableDependency(T^i, T^j, dependency_rule): This method re-enables a previously temporary disabled transaction *dependency_rule* between two transactions T^i and T^j. This operation is used when a transaction T^i finally determines its relationship with a transaction T^j.

- CreateMultipleAbortDependency($\Im_i$, T^j): This method initiates a new multiple-abort-dependency between the set of transactions $\Im_i$ and the transaction T^j. $\Im_i$ is either a set of delegator transactions or a set of import transactions; and T^j is the associated delegatee transaction.

6.3 Data hoarding stage

In this section, we formalize the data hoarding phase that will support the disconnected transaction processing stage by caching necessary data into the local workspaces at the mobile hosts. First, we present three different caching modes of mobile data. Second, we describe the data hoarding algorithm, and finally we show how our mobile transaction processing system supports the conflict awareness among transactions at different mobile hosts via the conflict awareness property of shared data.

6.3.1 Data caching modes

As described in Section 5.5.2, for each mobile host MH_i, there is an anchor transaction T_i^A that plays a role as a proxy transaction for all (offline) local transactions T_i^k at the mobile host MH_i. During the data hoarding stage, the anchor transaction (on behalf of local transactions) will try to acquire all the needed data items from the database servers. When an anchor transaction sends its lock action requests to the database servers, these lock requests have to compete with other lock requests that are coming from other online transactions or anchor transactions. For online transactions, the standard write and read locks [GR93] are applied. However, for offline transactions, these standard locks seem too strict to be applied in the mobile environments, i.e., only allowing non-conflict data caching mode (addressed below). Consequently, the mobile transaction processing system provides two additional data caching modes, called *read-write conflict* and *write-read conflict*. These conflict data sharing modes allow offline transactions to obtain conflict locks on shared data items. First, we present the basic definitions that will lead to our discussion on the conflict sharing modes:

> **Definition (*conflicting operations* [GUW01]).** *Two database operations Op_i and Op_j of two transactions T^i and T^j are in conflict if they are: (1) accessing the same data item, (2) one of them is a write operation. The conflict of database operations is denoted by Conflict(Op_i,Op_j).*
>
> **Definition (*directly conflicting transactions*).** *Two transactions T^i and T^j are in direct conflict, denoted by $T^i C_d T^j$, if there is an operation Op_i of transaction T^i that conflicts with an operation Op_j of transaction T^j.*
>
> **Definition (*indirectly conflicting transactions*).** *Two transactions T^i and T^j are in indirect conflict, denoted by $T^i C_{id} T^j$, if there is a transaction T^k that T^i either develops a direct conflict or an indirect conflict with, and T^k develops either a direct conflict or an indirect conflict with T^j, i.e.,*
>
> $$T^i C_{id} T^j \text{ if } \exists T^k, (T^i C_d T^k \vee T^i C_{id} T^k) \wedge (T^k C_d T^j \vee T^k C_{id} T^j)$$

In our mobile transaction processing system, there are three different data caching modes: non-conflict, read-write conflict and write-read conflict. These mobile data caching modes are discussed below.

Non-conflict data caching mode

For non-conflict data sharing mode, the database servers make sure that no conflict lock request is allowed during data caching phase. The standard exclusive (i.e., write) and inclusive (i.e., read) locking matrix is applied (see Table 6.4). The database servers grant only non-conflict locks to the lock requests from the anchor transaction T_i^A, and the shared data that is cached at the local mobile host is fully consistent.

Table 6.4: Non-conflict sharing mode

		Online transaction T^k or anchor transaction T_i^A holds	
	Lock type	Read	Write
Online transaction T^p or anchor transaction T_j^A requests	Read	No conflict	Conflict
	Write	Conflict	Conflict

Note that in non-conflict data caching mode, a mobile host starts with no conflicts in shared data before disconnection from the database servers. However, the mobile host may end up with conflicts on locks on shared data if the mobile host carries out mobile data sharing with other mobile hosts while being disconnected from the database servers (see Section 6.4.2).

Read-write conflict data caching mode

In mobile environments, the non-conflict data sharing mode above seems to be too restricted to be useful. Figure 6.5 illustrates the scenario. Suppose that an online transaction T_1^i at connected mobile host MH_1 is holding a read lock X_R on a shared data item X, and an offline transaction T_2^j at mobile host MH_2 requests a write lock X_W on this shared data. The write lock request can be granted to the offline transaction T_2^j because the write operation W_X by transaction T_2^j is not immediately carried out at the database servers, even after the online transaction T_1^i has committed. And the transaction T_1^i is scheduled to execute before transaction T_2^j, i.e., $T_1^i \rightarrow T_2^j$ (the execution constraints are discussed in Section 6.5).

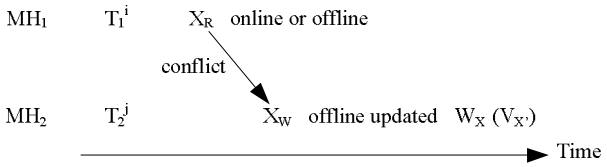

Figure 6.5: Read-write conflict mode

To handle this limitation, the mobile transaction processing system will allow these conflict lock requests to be compatible:

Definition (*read-write conflict*). *If an online transaction T^k or an anchor transaction T_i^A holds a read lock on data item X, and an anchor transaction T_j^A requests a write lock on data item X, the database server grants the write lock to T_j^A. We call this conflict mode a read-write (RW) conflict and denote it $X_{RW}(T^k, T_j^A)$ or $X_{RW}(T_i^A, T_j^A)$.*

The lock table for the read-write conflict is presented in Table 6.5.

Table 6.5: Read-write conflict mode

		Online transaction T^k or anchor transaction T_i^A holds	
	Lock type	Read	Write
Anchor transaction T_j^A requests	Read	No conflict	Conflict
	Write	*Allowed rw-conflict*	Conflict

Our read-write conflict mode focuses on supporting offline transactions at the disconnected mobile hosts. The read-write conflict provides the mobile transaction processing system the ability to avoid blocking of the execution of an offline updating transaction, i.e., if the shared data item X is read locked by an online transaction T^k or an anchor transaction T_i^A, the write lock request from anchor transaction T_j^A will be granted. In Figure 6.5, when the mobile host MH_2 reconnects to the database servers, the write (offline) transaction T_2^j will be converted to an online transaction (i.e., with online write lock on the shared data item X) so that the updated data value $V_{X'}$ will be integrated into the database servers. At this time, any on-going online transaction T^p that currently holds read lock on the shared data item X is either allowed to commit (given that the final commitment of the transaction T_2^j will be delayed) or aborted (see Section 6.6 for further detail).

Write-read conflict data caching mode

In read-write conflict data caching mode, a write lock request on the shared data item of an offline transaction is granted even if the shared data is currently being read lock by other transactions. On the other hand, an online transaction or an offline transaction can be allowed to read a shared data item while another offline transaction holds a write lock on the same shared data item, as long as these transactions can be serialized with the offline updating transaction.

Figure 6.6 illustrates the write-read conflict scenario. The offline transaction T_2^j at disconnected mobile host MH_2 holds a write lock X_W on the shared data item X. However, this data item is not being immediately modified at the database servers because the mobile host MH_2 that executes transaction T_2^j is currently being disconnected. When an (online or offline) transaction T_1^i at mobile host MH_1 requests a read lock X_R on the data item X, this read lock will conflict with the write lock on X held by transaction T_2^j. In this

case, the database server can grant a read lock on X (and consequently allow the read operation to be executed) for transaction T_1^i, given the original value V_X of the data item X is returned (this value might be inconsistent with the value of X that is stored and being modified at the disconnected mobile host MH_2). In fact, at the database servers, the original data value V_X is the most up-to-date and consistent data. Consequently, to ensure that the involved transactions are serializable, transaction T_1^i must be scheduled before transaction T_2^j, i.e., $T_1^i \rightarrow T_2^j$. Note that the offline transaction T_2^j may not know about this conflict that is happening at the database servers.

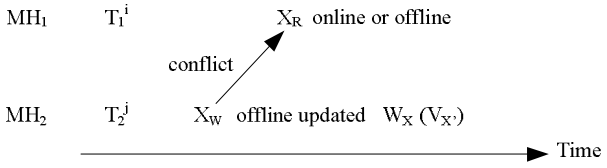

Figure 6.6: Write-read conflict mode

To handle this limitation, the mobile transaction processing system will allow these conflict lock requests to be compatible:

> **Definition (*write-read conflict*).** *If an anchor transaction T_i^A holds a write lock on data item X, and an online transaction T^k or an anchor transaction T_j^A requests a read lock on data item X, the database server grants the read lock request and the un-modified value of X is returned. We call this conflict mode a write-read (WR) conflict and denote it $X_{WR}(T_i^A, T^k)$ or $X_{WR}(T_i^A, T_j^A)$.*

The lock table for the write-read conflict is presented in Table 6.6.

Table 6.6: Write-read conflict mode

		Transaction T_i^A holds	
	Lock type	Read	Write
Online transaction T^k or anchor transaction T_j^A requests	Read	No conflict	*Allowed wr-conflict*
	Write	Conflict	Conflict

The write-read conflict mode allows read operations to be executed when there is a write operation that is being executed at the disconnected mobile host, i.e., avoids blocking of the execution of the read operations on the shared data item. In Figure 6.6, when the mobile host MH_2 reconnects to the database servers, the write (offline) transaction T_2^j will be converted to an online updating transaction with an online write lock on the shared data item X so that the updated data value $V_{X'}$ will be integrated into the database servers. At this time, any on-going online transaction T^p that currently holds a read lock on the shared data item X is either allowed to commit (given that the final commitment of the transaction T_2^j will be delayed) or aborted (see Section 6.6 for further detail).

6.3.2 Shared data in a mobile environment

The properties of a shared data item are: *value*, *conflict awareness* and *dependency awareness* (see Figure 6.7).

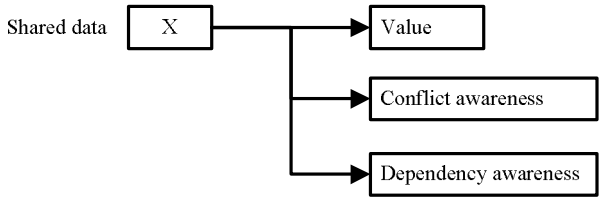

Figure 6.7: Properties of shared data in a mobile environment

The properties of a shared data item X, which is cached in the local workspace at a mobile host MH_i, are explained as follows:

- The value V_X is the actual value of the shared data item X in the local workspace.

- The conflict awareness X_{CA} is a set of conflict records whose structure is $X_{conflict\_mode}\ (T_i^A, T^c)^{conflict\_type}$ or $X_{lock\_type}(T^c, shared\_mode)$.

  The record $X_{conflict\_mode}\ (T_i^A, T^c)^{conflict\_type}$ is explained as follows:
  - The *conflict_mode* denotes the conflict data caching mode between the anchor transaction T_i^A and the transaction T^c on the shared data item X. Therefore, the *conflict_mode* is either a read-write conflict (*RW*) or a write-read conflict (*WR*).
  - T_i^A is the anchor transaction of the mobile host MH_i.
  - The transaction T^c can be either:
    - An anchor transaction T_j^A of the mobile host MH_j. The conflict record implies that there is a local transaction T_i^k at the mobile host MH_i that is conflict with a local transaction T_j^l at the mobile host MH_j. This conflict awareness occurs in the data hoarding stage, and the actual identifications of the local transactions T_i^k and T_j^l are not to be known until the transaction integration processes are carried out.
    - A local transaction T_j^l or a set of local transactions $\Im_j$ at mobile host MH_j. The conflict record means that there is a conflict between a local transaction T_i^k at the mobile host MH_i with one or many local transactions T_j^l at the mobile host MH_j. This conflict awareness occurs in the transaction integration stage where the identification of the local transaction(s) T_j^l is known (see Section 6.6).
  - The *conflict_type* is either an *Active conflict* or a *Passive conflict.* The active conflict is a conflict that occurs in the data hoarding stage and before the mobile host is disconnected. This means that both the anchor transaction T_i^A and local transactions T_i^k at the mobile host MH_i are aware of these conflicts. The passive conflict is a conflict that occurs after the mobile host is

disconnected from the database servers. Therefore, only the anchor transaction T_i^A is aware of the conflict, and the local transaction T_i^k at the disconnected mobile host MH_i is not aware of the conflict. The active and passive conflicts are denoted by the superscripts ^{A} and ^{P}, respectively.

The record $X_{lock\_type}(T^c, shared\_mode)$ is explained as follows:

- The *lock_type* can be either a read lock (*R*), or a write lock (*W*) or a pseudo-read lock (*Rp*). The pseudo-read lock is used when a delegator transaction shares a data state to a delegatee transaction.
- T^c is the delegator transaction that shares data item *X*.
- The *shared_mode* can be either *Original*, *Updated* or *Status.* The original or updated mode is applied with sharing data states and corresponds with the pseudo-read lock, while the status mode is used with sharing data status.

- The dependency awareness X_{DA} is a set of dependency rules whose structure is: $X(T_i^k, dependency\_rule)$, where:
  - T_i^k is the transaction that manipulated the shared data item *X*.
  - The *dependency_rule* can be either an *Abort-dependency* or a *Commit-dependency* (see Section 6.2). For example, the dependency awareness $X(T_i^k, AD)$ indicates that any transaction T_j^l that accesses the shared data item *X* will develop an abort-dependency with the transaction T_i^k.

The properties of a shared data item can be dynamically modified by local transactions at a mobile host. The usages of these properties will be presented in the following sub-sections.

6.3.3 Caching algorithm for the anchor transaction

In this section, we present the data caching algorithm that allows consistent data to be granted to a mobile host for supporting disconnected transaction processing. Before going into detail of the algorithm, we need to define several notations:

- T_i^j denotes a local transaction T^j at the mobile host MH_i that will be carried out when the mobile host is disconnected.

- $D_i^j = D_i^{jR} \cup D_i^{jW}$ denotes the accessed data set associated with the local transaction T_i^j, where D_i^{jR} and D_i^{jW} are the read and write data sets respectively required by the transaction T_i^j when the mobile host is disconnected. The data set $D_i^j = D_i^{jR} \cup D_i^{jW}$ that is needed for the local transaction T_i^j will be cached in the local workspace at the mobile host. A shared data item exclusively belongs either to a read data set or a write data set, i.e., $D_i^{jR} \cap D_i^{jW} = \varnothing$.

- $\mathfrak{I}_i$ denotes the set of local transactions T_i^j at the mobile host MH_i, i.e.,
  $\mathfrak{I}_i = \{ T_i^j, j > 0 \}$).

- $D_i = D_i^R \cup D_i^W$ denotes the accessed data set, which is associated with the local transaction set $\mathfrak{T}_i$, that need to be cached at a mobile host MH_i for disconnected transaction processing. D_i^R and D_i^W denote the read data set and write data set respectively of all the transactions belonging to the mobile host MH_i. Thus,

$$D_i^W = \bigcup_{j=1}^{n} D_i^{jW} \ \wedge \ D_i^R = \bigcup_{j=1}^{n} D_i^{jR}$$

- X_R and X_W denote the read and write lock associated with the data item X, respectively. Let L_i be the set of locks associated with the data set D_i, i.e., L_i contains all the read and write locks of cached data at the mobile host MH_i.

$L_i = L_i^R \cup L_i^W$ where L_i^R is the read lock set of the read data set D_i^R, and L_i^W is the write lock set of the write data set D_i^W.

The read lock set L_i^R and the write lock set L_i^W might be intersecting with each other, i.e., $L_i^R \cap L_i^W \neq \varnothing$. This is due to the overlap of accessed data sets of local transactions at the mobile hosts, i.e., $D_i^W \cap D_i^R \neq \varnothing$. Consequently, this may cause redundant lock requests from the anchor transaction. For example, the anchor transaction may request both read lock and write lock for a modifiable data item. Hence, we define the actual needed caching data and lock sets:

$$D_i^A = D_i^{RA} \cup D_i^{WA} \ \wedge \ D_i^{RA} \cap D_i^{WA} = \varnothing$$
where $D_i^{WA} = D_i^W \ \wedge \ D_i^{RA} = D_i^R \setminus D_i^W$

$$L_i^A = L_i^{RA} \cup L_i^{WA} \ \wedge \ L_i^{RA} \cap L_i^{WA} = \varnothing$$
where L_i^{RA} is the read lock set of the actually needed read data set D_i^{RA}, L_i^{WA} is the write lock set of the actually needed write data set D_i^{WA}.

For example, if a transaction T_i^1 requests a read data set $D_i^{1R} = \{a,b,c\}$ and a write data set $D_i^{1W} = \{d,e,f\}$, and transaction T_i^2 requests a read data set $D_i^{2R} = \{a,d,e\}$ and a write data set $D_i^{2W} = \{b,c,f\}$, the actual read data set D_i^{RA} and write data set D_i^{WA}, which will be requested to be cached at the mobile host MH_i, and the associated read lock set L_i^{RA} and write lock set L_i^{WA} will be:

$$D_i^{WA} = D_i^W = D_i^{1W} \cup D_i^{2W} = \{b,c,d,e,f\}$$
$$D_i^R = D_i^{1R} \cup D_i^{2R} = \{a,b,c,d,e\}$$
$$D_i^{RA} = D_i^R \setminus D_i^W = \{a,b,c,d,e\} \setminus \{b,c,d,e,f\} = \{a\}$$
$$L_i^{RA} = \{a_R\} \wedge L_i^{WA} = \{b_W, c_W, d_W, e_W, f_W\}$$

The anchor transaction T_i^A is considered as a root transaction that will request all the locks of the lock set L_i^A associated with the actually needed data set D_i^A for a set of local

transactions $\mathfrak{I}_i$ at the mobile host MH_i. The procedure of granting locks on shared data items for anchor transactions depends on the caching modes which are deployed by a mobile transaction processing system. The default caching mode in our mobile transaction processing system is to allow both read-write and write-read conflicts.

Note that during the data hoarding stage, the anchor transaction might not successfully obtain all shared data items in the actually needed data set D_i^A due to conflicts with other online or anchor transactions. For example, the database server will not grant any lock request on a shared data item that is being modified by an online transaction. Therefore, the granted access data set $D_i^G = D_i^{RG} \cup D_i^{WG}$ and the granted lock set $L_i^G = L_i^{RG} \cup L_i^{WG}$ can be different from the actually needed data set D_i^A and the associated lock set L_i^A, respectively. When the data hoarding stage is completed, the anchor transaction will hold the granted access data set $D_i^G = D_i^{RG} \cup D_i^{WG}$ and the granted lock set $L_i^G = L_i^{RG} \cup L_i^{WG}$. Figure 6.8 presents the data caching algorithm of the anchor transaction T_i^A of the mobile host MH_i.

(1) Initially: $L_i^G = L_i^{RG} \cup L_i^{WG} = \varnothing$
Initially: $D_i^G = D_i^{RG} \cup D_i^{WG} = \varnothing$
(2) For each lock request X_W in the lock set L_i^{WA}
Request the write lock X_W with the default caching mode
If the write lock X_W is granted
Add X_W to the granted write lock set, i.e., $L_i^{WG} = L_i^{WG} \cup \{X_W\}$
Add X to the granted write data set, i.e., $D_i^{WG} = D_i^{WG} \cup \{X\}$
If there are read-write conflicts
Add these read-write conflicts to the conflict awareness X_{CA}
(3) Else If X_R is in the read lock set L_i^R
Request the read lock X_R with the default caching mode
If the read lock X_R is granted
Add X_R to the granted read lock set, i.e., $L_i^{RG} = L_i^{RG} \cup \{X_R\}$
Add X to the granted read data set, i.e., $D_i^{RG} = D_i^{RG} \cup \{X\}$
If there are write-read conflicts
Add these write-read conflicts to the conflict awareness X_{CA}
(4) Replicate a copy of the granted lock set and the granted data set to the local workspace at the mobile host

Figure 6.8: Algorithm for data caching stage

The above data caching algorithm of the anchor transaction T_i^A of the mobile host MH_i is explained as follows:

(1) The granted access data sets and lock sets are initially empty.

(2) The anchor transaction T_i^A will first try to obtain the needed write locks, i.e., those are in the actually needed write lock set L_i^{WA}, by submitting write lock requests to the database servers. If these write lock requests are granted by the database servers, the locks will be added to the granted write lock set L_i^{WG}. The data items are downloaded into the local cache of the mobile host, and the local transactions T_i^j at the mobile host have the right to modify these shared data items. The anchor transaction T_i^A will hold these write locks and any read-write conflict associated on these shared data items (see Sections 6.3.2 and 6.3.4 for conflict awareness).

(3) If a write lock request of a shared data item X is rejected, the anchor transaction will check if there is any other local transaction T_i^j that wants to read this shared data item X, i.e., $X \in D_i^R$ and $X_R \in L_i^R$. If it is true, then the anchor transaction will try to request the read lock of the shared data item X. If the read lock request is granted by the database servers, the read lock X_R will be added to the granted read lock set L_i^{RG}. The data items are downloaded into the local cache of the mobile host as read-only, i.e., the local transactions T_i^j at the mobile host MH_i can only read these shared data items. The anchor transaction T_i^A will hold the read lock and any write-read conflict associated on the shared data item (see Sections 6.3.2 and 6.3.4 for conflict awareness).

(4) The granted lock set $L_i^G = L_i^{RG} \cup L_i^{WG}$ and the granted access data set $D_i^G = D_i^{RG} \cup D_i^{WG}$ will be locally replicated on the mobile host, denoted by $L_i^{GR} = L_i^{RGR} \cup L_i^{WGR}$ and $D_i^{GR} = D_i^{RGR} \cup D_i^{WGR}$. This replica of the granted lock set L_i^{GR} will be used by the transaction manager at the mobile host to support the concurrency control of local transactions.

At the end of the data hoarding stage, the anchor transaction T_i^A will hold the granted access data set $D_i^G = D_i^{RG} \cup D_i^{WG}$ and the granted lock set $L_i^G = L_i^{RG} \cup L_i^{WG}$. If the actually needed data set is not fully cached in the local workspace, i.e., $D_i^G \subset D_i^A$ and $L_i^G \subset L_i^A$, the mobile host will try to obtain more shared data from other mobile host (see Section 6.4 for mobile data sharing stage) while being disconnected from the database servers. Therefore the local replicated lock set L_i^{GR} at the mobile host can be modified and temporarily inconsistent with the originally granted lock set L_i^G held by the anchor transaction T_i^A. These inconsistencies will be reconciled at the transaction integration stage.

When mobile host MH_i reconnects to the database servers, the original lock set L_i^G held by anchor transaction T_i^A will be synchronised with the replicated local lock set L_i^{GR}. The lock synchronization process is performed at the database servers and can cause the anchor transaction T_i^A to have to synchronize conflict locks with other anchor transactions (discussed in Section 6.6.2). If the conflicts are resolved, the locally committed transactions T_i^j will be finally committed at the database servers. Otherwise these local transactions will be aborted. Finally, the anchor transaction T_i^A releases all the locks and commits.

6.3.4 Supporting conflict awareness

When conflict data caching modes are allowed, local transactions that are planned for disconnected processing at the mobile hosts must be aware of conflicts of their database operations. These conflicts can either happen in the data hoarding stage or after the mobile hosts are disconnected from the database servers. The anchor transactions that reside at the fixed database servers will support the local transactions at the mobile hosts to be aware of these conflict operations. For each cached data item, the conflict awareness identifies the potential conflicts between transactions at different mobile hosts.

Figure 6.9 illustrates the awareness support of anchor transactions during the data hoarding stage. Time proceeds from left to right. At the time t_1, the anchor transaction T_1^A of the mobile host MH_1 holds a read lock and a write lock on shared data items X and Y, respectively. At this time, there is no conflict on the system and all local transactions at the mobile host MH_1 are not aware of any database conflict. At the time t_2, when the mobile host MH_1 has been disconnected from the database servers, the anchor transaction T_2^A of the mobile host MH_2 requests a write offline lock on the shared data item X. The database servers grant this lock request. Both anchor transactions T_1^A and T_2^A are aware of, and will modify the conflict awareness X_{CA} of the shared data item X with read-write conflict $X_{RW}(T_1^A, T_2^A)$. For the mobile host MH_1, this is a passive conflict awareness, denoted by the $X_{RW}(T_1^A, T_2^A)^P$. This means that the local transactions at the disconnected mobile host MH_1 do not know about this conflict. For the mobile host MH_2, this is an active conflict awareness, denoted by the $X_{RW}(T_1^A, T_2^A)^A$. At the time t_3, the anchor transaction T_3^A requests both read locks on the shared data items X and Y, and the database servers grant these conflicting locks. Anchor transactions T_1^A, T_2^A and T_3^A are aware of these new conflicts. The anchor transactions T_1^A modifies the conflict awareness Y_{CA} of the shared data item Y with a passive write-read conflict $Y_{WR}(T_1^A, T_3^A)^P$, the anchor transaction T_2^A modifies the conflict awareness X_{CA} of the shared data item X with a passive write-read conflict $X_{WR}(T_2^A, T_3^A)^P$, and the anchor transaction T_3^A will modify the conflict awareness of both Y and X as active write-read conflict $Y_{WR}(T_1^A, T_3^A)^A$ and $X_{WR}(T_2^A, T_3^A)^A$, respectively.

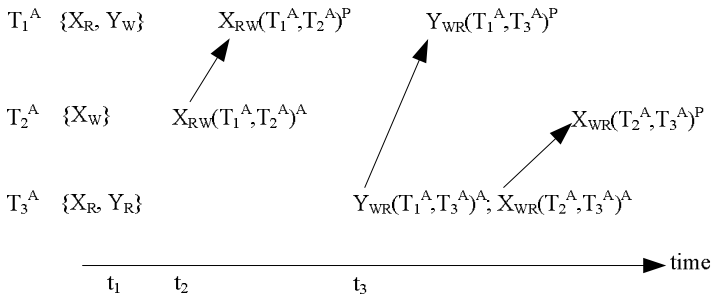

Figure 6.9: Conflict awareness of transactions

Table 6.7 indicates the locks and conflict awareness records held by the anchor transactions and in the local workspace at the disconnected mobile hosts. For the mobile

hosts MH_1 and MH_2, the conflict awareness records held by the anchor transactions are inconsistent with the ones in the local workspace at the disconnected mobile hosts. These conflict awareness records will be used in the transaction integration stage to determine the final execution schedule of transactions.

Table 6.7: Locks and conflict awareness among mobile hosts

		MH_1	**MH_2**	**MH_3**
Locks	Anchor transaction	$X_R;Y_W$	X_W	$X_R;Y_R$
	Local workspace	$X_R;Y_W$	X_W	$X_R;Y_R$
Conflict awareness	Anchor transaction	$X_{RW}(T_1^A,T_2^A)^P$; $Y_{WR}(T_1^A,T_3^A)^P$	$X_{RW}(T_1^A,T_2^A)^A$; $X_{WR}(T_2^A,T_3^A)^P$	$X_{WR}(T_2^A,T_3^A)^A$; $Y_{WR}(T_1^A,T_3^A)^A$
	Local workspace	None	$X_{RW}(T_1^A,T_2^A)^A$	$X_{WR}(T_2^A,T_3^A)^A$; $Y_{WR}(T_1^A,T_3^A)^A$

The operations for managing conflict awareness

The conflict lock requests (at the database servers) can happen any time during the data hoarding stage or when the mobile hosts are being disconnected. The mobile transaction processing system, thus, provides the following operations to support the anchor transaction T_i^A to manage the conflict awareness:

- AddConflict(*shared_data, conflict_transaction, conflict_mode, conflict_type*). This operation adds a new conflict awareness record on a *shared_data X* to the conflict awareness record set X_{CA} that is held by the anchor transaction T_i^A. The *conflict_transaction* can be either an anchor transaction or a standard transaction. The *conflict_mode* is either a read-write conflict or write-read conflict between the anchor transaction T_i^A and the *conflict_transaction*. If the mobile host is still connected to the database servers at the time that the conflict lock occurs, the *conflict_type* is an active conflict; otherwise, it is a passive conflict.

- RemoveConflict(*shared_data*, *conflict_transaction*). This operation removes the conflict awareness record between the anchor transaction T_i^A and the *conflict_transaction* from the conflict awareness record set X_{CA} of *shared_data X*. This operation is invoked when the *conflict_transaction* is no longer involved in the shared data item.

- ModifyConflict(*shared_data, anchor_transaction*, *new_conflict_transaction*). This operation allows an anchor transaction T_i^A to modify a conflict awareness record when a mobile transaction finally commits at the database servers. The conflict awareness record on the shared data between the transaction pair (T_i^A, T_j^A) will be replaced by the conflict transaction pair (T_i^A,T_j^l) where T_j^l is the identification of the standard conflicting mobile transaction (see Section 6.6 for further detail).

6.4 Mobile data sharing stage

In this section, we formalize the mobile data sharing process among transactions at different disconnected mobile hosts. To recap, we distinguish two main mobile data sharing types: sharing data states and sharing data status. The mobile data sharing operations between the standard delegator and delegatee transactions are carried out with the support of the export and import transactions (from now, we will assume that the delegator and delegatee transactions belong to different mobile hosts). Table 6.8 summaries the management of mobile data sharing between the delegator and delegatee transactions.

Table 6.8: Management of mobile data sharing

	Delegator transaction	**Delegatee transaction**
Sharing data states	Exports original or updated data states	Imports data states as new data states Upgrades data states
Sharing data status	Delegates read or write locks Downgrades write locks	Imports read or write locks as new locks Upgrades write locks

6.4.1 Management of sharing data states

In this section, we will formalize the sharing of mobile data states (i.e., data values) among standard transactions at different mobile hosts. For sharing values, only the values of shared data items that are being cached at a mobile host are revealed to other transactions at different mobile hosts.

The delegator transaction T_i^{Dor} will export shared data values to the export-import repository together with any conflict awareness or dependency awareness related to these shared data values. The delegator transaction T_i^{Dor} still holds the responsibility (i.e., locks) of the shared data items. As discussed in Section 5.5.4, depending on status of the shared data (i.e., read or write lock) that is cached in the local workspace, the delegator transaction T_i^{Dor} can share either the original data value or the updated data value (see Figure 6.10). Furthermore, the delegator transaction does not need to be aware of the states of the associated delegatee transactions. In other words, it is not necessary for the delegator transaction to know about what delegatee transactions that will obtain its shared data states. The delegatee transaction T_j^{Dee} can either obtain the shared data state as a new data item or upgrade its local cached data (see Figure 6.10). If the shared data item is not cached in the local workspace, the delegatee transaction will import it as a newly cached data. On the other hand, if the shared data item is already being cached in the local workspace, the delegatee transaction can use this opportunity to upgrade the value of the shared data item to the most up-to-date value.

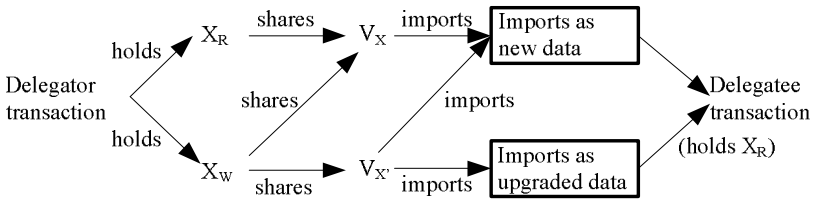

Figure 6.10: Sharing data states

Figure 6.11 illustrates an example for mobile data sharing states among mobile transactions at two mobile hosts MH_1 and MH_2. The example will be used to illustrate our analysis in the rest of this section. The anchor transaction T_1^A of the mobile host MH_1 holds a non-conflict read lock on the shared data item X, and an active read-write conflict $Y_{RW}(T_2^A, T_1^A)^A$ on shared data item Y (i.e., with a write lock on Y) with the anchor transaction T_2^A of the mobile host MH_2. At the same time, the anchor transaction T_2^A of the mobile host MH_2 holds a passive read-write conflict $Y_{RW}(T_2^A, T_1^A)^P$ on shared data item Y (i.e., with a read lock on Y) and a write lock on the shared data item Z. During the mobile data sharing stage, delegator transactions at the mobile host MH_1 share both the original value of X, i.e., V_X, and the modified value of Y, i.e., $V_{Y'}$, into the mobile sharing workspace. Delegator transactions at the mobile host MH_2 share both the original and updated value of data item Z, i.e., V_Z and $V_{Z'}$. A delegatee transaction at the mobile host MH_1 will sequentially obtain both the shared data values of the shared data item Z. And, a delegatee transaction at the mobile host MH_2 will import the shared data value of X as a new cached data, and upgrade its local cache on the shared data item Y to the most up-to-date value $V_{Y'}$.

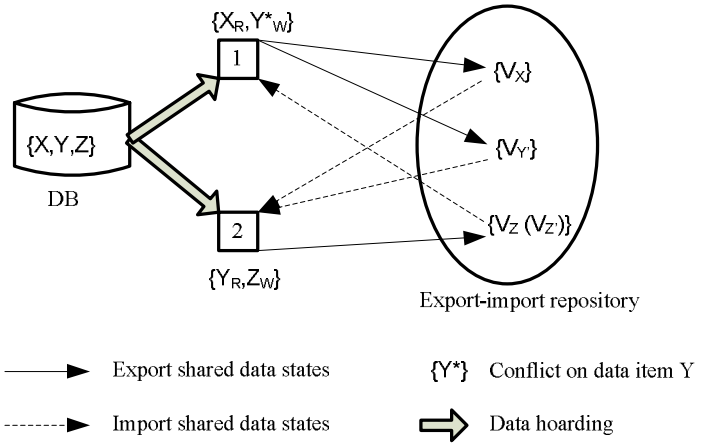

Figure 6.11: Shared data states in the export-import sharing space

The locks and conflict awareness records held by the anchor transactions at the database servers and in the local workspaces at the disconnected mobile hosts, as well as the mobile data sharing states are summarized in Table 6.9.

Table 6.9: Locks and data conflict awareness of sharing data state scenarios

		MH_1	MH_2
Locks	Anchor transaction	$X_R;Y_W$	$Y_R;Z_W$
	Local workspace	$X_R;Y_W$	$Y_R;Z_W$
Conflict awareness	Anchor transaction	$Y_{RW}(T_2^A,T_1^A)^A$	$Y_{RW}(T_2^A,T_1^A)^P$
	Local workspace	$Y_{RW}(T_2^A,T_1^A)^A$	None
Mobile data sharing	Exported data states	V_X, $V_{Y'}$	V_Z, $V_{Z'}$
	Imported data states	V_Z, $V_{Z'}$	V_X, $V_{Y'}$

Conditions of sharing data states

As we have discussed in Section 5.5.4, in order to be able to share the data state of the shared data item *X*, which is either an original state or an updated state, a delegator transaction T_i^{Dor} at mobile host MH_i must hold the appropriate lock on the shared data item *X*. This means that the following conditions must be met:

(1) For sharing of an original data state of the data item *X*: the data item *X* is cached (with read lock or write lock) in the local workspace at the mobile host MH_i, and the data item *X* is in the accessed data set of the delegator transaction T_i^{Dor}, i.e.,

$$(l_X \in L_i^{GR}) \wedge (X \in D_i^{Dor})$$

Note that the delegator transaction T_i^{Dor} can be either a read-only transaction or an updating transaction. If the delegator transaction is a read-only transaction, then the data item *X* is in the read data set. Otherwise, the data item *X* is in the write data set of the updating delegator transaction T_i^{Dor} (i.e., $X_W \in L_i^{WGR}$), however, the value of *X* is not modified by T_i^{Dor} yet.

(2) For sharing of an updated data state of the shared data item *X*: the data item *X* is cached with write lock in the local workspace at the mobile host MH_i, and the data item *X* is in the write data set of the updating delegator transaction T_i^{Dor}, i.e.,

$$(X_W \in L_i^{WGR}) \wedge (X \in D_i^{Dor})$$

Operations of sharing data states

When the delegator transaction T_i^{Dor} at the mobile host MH_i shares the value of the data item *X*, the procedure of exporting shared data states is implemented as follows:

(1) The delegator transaction T_i^{Dor} initiates an export transaction $T_i^{Dor.E}$ that will export the shared data state into the export-import repository.

(2) For each shared data item *X*, attach all associated information to the export transaction $T_i^{Dor.E}$ (see Table 6.10). The associated information of the export transaction is also logged in the local workspace at the mobile host MH_i.

(3) The export transaction $T_i^{Dor.E}$ is dispatched to be executed in the export-import repository.

Table 6.10: Data structure for exporting shared data states

Attribute	Description
ItemID	The identification of the shared data item
ItemValue	The shared value of the shared data item
TypeOfState	The type of sharing data state is either *original* or *updated* data state
DelegatorID	The identification of the delegator transaction
TypeOfShare	The type of data sharing is *share_state* (i.e., *read-only* here)
ItemDepend	The dependency awareness related to the shared data item
ItemConflict	The conflict awareness related to the shared data item

The data structure of the shared data state that is exported by the delegator transaction contains all the necessary information that describes the correlation between the delegator transaction and the shared data item. When a delegatee transaction imports this shared data state, the information will be used as a means to set the relationship between the delegator and delegatee transactions. Furthermore, the attached information is associated with individual shared data items, and therefore, supports different versions of a data item to be shared in the mobile sharing workspaces. These shared data items are independent of each other. Consequently, the delegatee transactions can select which shared data items to be obtained.

When a delegatee transaction T_i^{Dee} at the mobile host MH_i wants to obtain shared data, the delegatee transaction T_i^{Dee} will initiate an import transaction $T_i^{Dee.I}$ that will try to collect the shared data from the export-import repository. The delegatee transaction must clearly specify what type of shared data it wants to import, i.e., read-only or modifiable. To recap, the imported data states are read-only; therefore, if the delegatee transaction wants to obtain modifiable shared data, it must try to import the data status (see Sections 5.5.4 and 6.4.2). Moreover, the delegatee transaction does not know what shared data is available or how the shared data is shared in the export-import repository (i.e., share states or share status). The actual result of the import transaction indicates whether the collected data is a shared state or a shared status. In this section, we focus on obtaining the shared data state, i.e., read-only shared data. When the wanted data item is obtained, the delegatee transaction will also be aware of and handle any conflict related to the shared data.

When the delegatee transaction T_i^{Dee} at the mobile host MH_i imports the value of the shared data item X, the procedure of importing shared data state is implemented as follows:

(1) The delegatee transaction T_i^{Dee} initiates an import transaction $T_i^{Dee.I}$ that will import the needed shared data from the export-import repository to the local workspace.

(2) All necessary information related to the needed shared data (see Table 6.11) is attached to the import transaction $T_i^{Dee.I}$. This information is also written to a log in the local workspace.

(3) The import transaction $T_i^{Dee.I}$ is dispatched to the export-import repository.

Table 6.11: Data structure for importing shared data states

Attribute	Description
ItemID	The identification of the shared data item
TypeOfShare	The type of data sharing is *share_state* (i.e., *read-only* here)
TransDepend	The transaction dependency between the delegatee and the import transaction(s) (i.e., *abort-dependency* or *multiple-abort-dependency*)

The import transaction will select and read from the export-import repository the most equivalent shared data item (if there are many different versions of the data item in the export-import repository). After that, the import transaction writes the obtained data into the local workspace at the mobile host and commits. For sharing data states, the obtained data values are read only to local transactions.

Before the collected shared data state V_X of the shared data item X is made available to other local transactions, the following procedure is carried out:

(1) The newly obtained data value is added to the local cache as a new read-only shared data. If the shared data is already being read-only cached, its value will be updated to the most up-to-date value.

(2) A *pseudo-read lock* X_{Rp} of shared data item X will be added to the replicated read lock set L_i^{RGR} . All database operations at the mobile host that read this new obtained data value are marked as *pseudo-read* operations. This is to distinguish between the actual read operations that are protected by a read lock at the anchor transaction, and the pseudo-read operations that read the imported shared data not being read locked by the anchor transaction. In other words, the pseudo-read operation allows transactions to read a shared data item without connecting to the database servers to obtain the appropriate read lock.

(3) The conflict awareness X_{CA} and dependency awareness X_{DA} are modified in accordance with the properties of the shared data value obtained, explained as follows:

- If a delegator transaction T_i^{Dor} shares an original data state, a conflict awareness $X_{Rp}(T_i^{Dor}, original)$ is added to X_{CA}.
- If a delegator transaction T_i^{Dor} shares an updated data state, the following conflict awareness and dependency awareness records will be added to X_{CA} and X_{DA}:

- A conflict awareness $X_{Rp}(T_i^{Dor}, updated)$ is added to X_{CA}.
- An abort-dependency $X(T_i^{Dor}, AD)$ is added to X_{DA}: this indicates that if the delegator transaction T_i^{Dor} aborts, transactions T_j^l that have read X will be aborted, i.e., $T_i^{Dor} AD\ T_j^l$.
- A commit-dependency $X(T_i^{Dor}, CD)$ is added to X_{DA}: this indicates that if a transaction T_j^l has reads X, it will commit after transaction T_i^{Dor} has committed, i.e., $T_i^{Dor}\ CD\ T_j^l$.

o If there are other conflict awareness or dependency awareness records associated with X (indicated via *ItemConflict* and *ItemDepend* records – see Table 6.10), these records will be added to X_{CA} and X_{DA} respectively.

In the following illustrations, we address in detail what actually happens when sharing of data states takes place. There are four different sharing data state scenarios that are grouped into three different cases (see Table 6.12). These examples build on those in Figure 6.11.

Table 6.12: Sharing data state scenarios

Case	Delegator transaction	Delegatee transaction
1	Holds read lock and exports original data value	Imports the shared data value as a new shared data
	Holds write lock and exports original data value	Imports the shared data value as a new shared data
2	Holds write lock and exports updated data value	Imports the shared data value as a new shared data
3	Holds write lock and exports updated data value	Imports the shared data value as an updated shared data

Case 1: The delegator transaction shares an original data state and the delegatee transaction imports the shared data state as a new shared data.

Figure 6.12 illustrates examples of sharing the original data states between transactions at mobile hosts MH_1 and MH_2. The delegator transaction T_1^i at the mobile host MH_1 holds a read lock on the shared data item X and shares the original data value V_X to the delegatee transaction T_2^j at the mobile host MH_2. And the delegator transaction T_2^k at the mobile host MH_2 that holds a write lock on the shared data item Z shares the original data value V_Z to the delegatee transaction T_1^l at the mobile host MH_1.

The conditions for sharing of data states of two delegator transactions T_1^i and T_2^k are:

- For the delegator transaction T_1^i: $(X_R \in L_1^{R_{GR}}) \wedge (X \in D_1^i)$
- For the delegator transaction T_2^k: $(Z_W \in L_2^{W_{GR}}) \wedge (Z \in D_2^k)$

As described above, these conditions are fulfilled. Note that the delegator transaction T_2^k has not modified the value of data item Z yet.

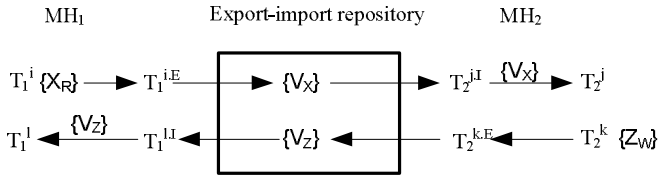

Figure 6.12: Share original data states

The following information is attached to the export transactions $T_1^{i.E}$ and $T_2^{k.E}$ and logged in the local workspaces before these export transactions are dispatched to the export-import repository:

- For the export transaction $T_1^{i.E}$: (X,V_X,original, T_1^i, share_state, none, none)
- For the export transaction $T_2^{k.E}$: (Z,V_Z,original, T_2^k, share_state, none, none)

Note that there is no conflict awareness or dependency awareness related to these shared data states. The shared data states of the data items *X* and *Z* are consistent with the ones in the database server. Therefore, if the delegator transaction aborts, the export transaction can still commit.

The delegatee transactions T_1^l and T_2^j will obtain these shared data states via the import transaction $T_1^{l.I}$ and $T_2^{j.I}$. The following information is attached to the import transactions $T_1^{l.I}$ and $T_2^{j.I}$ and logged in the local workspaces before these import transactions are dispatched to the export-import repository:

- For the import transaction $T_1^{l.I}$: (Z,read_only, none)
- For the import transaction $T_2^{j.I}$: (X,read_only, none)

There is no transaction dependency between the delegatee and import transactions. This means that the import transactions can commit in the local workspaces regardless of the state of their delegatee transactions.

When these import transactions commit in the local workspaces, the following procedures are carried out:

- At mobile host MH_1:
  - A pseudo-read lock Z_{Rp} is added to the granted read lock set, i.e.,
    $L_1^{RGR} := L_1^{RGR} \cup \{Z_{Rp}\}$
  - The shared data value V_Z is added as a new data item, i.e.,
    $D_1^{RGR} := D_1^{RGR} \cup \{Z\}$
  - A conflict awareness record $Z_{Rp}(T_2^k,original)$ will be added to the conflict awareness set Z_{CA} of data item *Z*, i.e.,
    $Z_{CA} := Z_{CA} \cup \{Z_{Rp}(T_2^k,original)\}$
- At mobile host MH_2:
  - A pseudo-read lock X_{Rp} is added to the granted read lock set, i.e.,
    $L_2^{RGR} := L_2^{RGR} \cup \{X_{Rp}\}$
  - The shared data value V_X is added as a new data item, i.e.,
    $D_2^{RGR} := D_2^{RGR} \cup \{X\}$

- A conflict awareness record $X_{Rp}(T_1^i,original)$ will be added to the conflict awareness set X_{CA} of data item X, i.e.,
  $X_{CA} := X_{CA} \cup \{X_{Rp}(T_1^i,original)\}$

The conflict awareness records will be used to determine the execution schedule between the delegator and delegate transactions. In Section 6.5 we will further formalize this execution schedule.

After these operations are completed, the collected shared data states are made accessible to the delegatee and other local transactions as if they are cached data. All the local read operations related to these shared data items will be marked as pseudo-read operations R_P. Table 6.13 summaries the states of cached data in the local workspaces and at the anchor transactions after this mobile data sharing.

Table 6.13: Locks and awareness of sharing original data states

		MH_1	**MH_2**
Locks	Anchor transaction	$X_R;Y_W$	$Y_R;Z_W$
	Local workspace	$X_R;Y_W;Z_{Rp}$	$X_{Rp};Y_R;Z_W$
Conflict awareness	Anchor transaction	$Y_{RW}(T_2^A,T_1^A)^A$	$Y_{RW}(T_2^A,T_1^A)^P$
	Local workspace	$Y_{RW}(T_2^A,T_1^A)^A$ $Z_{Rp}(T_2^k,original)$	$X_{Rp}(T_1^i,original)$

Case 2: The delegator transaction shares an updated data state and the delegatee transaction imports the updated data state as a new shared data.

In Figure 6.13, the delegator transaction T_2^j at the mobile host MH_2 updates the data item Z in the local workspace. After this, the delegator transaction T_2^j shares this modified data state $V_{Z'}$ of the shared data item Z to the delegatee transaction T_1^i at the mobile host MH_1. Because this shared data item Z is not cached in the local workspace at the mobile host MH_1, the delegatee transaction T_1^i imports this updated value $V_{Z'}$ of the data item Z as a new shared data item.

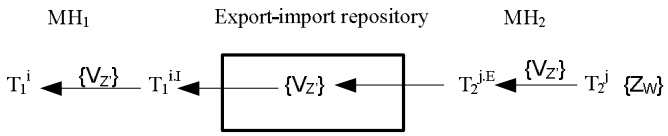

Figure 6.13: Share modified data state

The conditions for sharing of data state of the delegator transaction T_2^j at the mobile host MH_2 are:

$(Z_W \in L_2^{W_{GR}}) \wedge (Z \in D_2^j)$

Before the export transactions $T_2^{j.E}$ is dispatched to the export-import repository, the following information is attached to it and logged in the local workspace:

(Z,$V_{Z'}$,modified, T_2^j, share_state, Z(T_2^j,AD), none)

Note that there is an abort-dependency between the delegator transaction T_2^j and the delegatee transactions that will read the modified data state of data item *Z*. In other words, if the delegator transaction T_2^j aborts, the shared data value $V_{Z'}$ will become invalid. Therefore, if the delegator transaction T_2^j aborts, the export transaction $T_2^{j.E}$ must abort or be compensated, consequently the delegatee transactions that have read $V_{Z'}$ will be aborted. The abort-dependency is transferred via the dependency awareness record $Z(T_2^j,AD)$ of data item *Z*.

The delegatee transaction T_1^i will initiate an import transaction $T_1^{i.I}$ to obtain the shared data state $V_{Z'}$ from the export-import repository. The following information is attached to the import transactions $T_1^{i.I}$ and logged in the local workspace before the import transaction is dispatched to the export-import repository:

(Z,read_only, none)

When the import transaction $T_1^{i.I}$ commits in the local workspace, the following procedure is carried out at the mobile host MH_1:

- A pseudo-read lock Z_{Rp} is added to the granted read lock set, i.e.,
  $L_1^{RGR} := L_1^{RGR} \cup \{Z_{Rp}\}$
- The shared data value $V_{Z'}$ is added as a new data item, i.e.,
  $D_1^{RGR} := D_1^{RGR} \cup \{Z\}$
- A conflict awareness record $Z_{Rp}(T_2^j,updated)$ is added to the conflict awareness set Z_{CA} of data item *Z*, i.e.,
  $Z_{CA} := Z_{CA} \cup \{Z_{Rp}(T_2^j,updated)\}$
- A dependency awareness record $Z(T_2^j,AD)$ is added to the dependency awareness set Z_{DA} of data item *Z*, i.e.,
  $Z_{DA} := Z_{DA} \cup \{Z(T_2^j,AD)\}$
  This dependency awareness record indicates that local transactions T_1^p at the mobile host MH_1 that read data item *Z* will have an abort-dependency with transaction T_2^j, i.e., T_2^j *AD* T_1^p.
- A dependency awareness record $Z(T_2^j,CD)$ is further added to the dependency awareness set Z_{DA} of data item *Z*, i.e.,
  $Z_{DA} := Z_{DA} \cup \{Z(T_2^j,CD)\}$
  This dependency awareness record indicates that local transactions T_1^p at the mobile host MH_1 that read data item *Z* will have a commit-dependency with transaction T_2^j, i.e., T_2^j *CD* T_1^p.

After these operations are completed, the collected shared data state $V_{Z'}$ is made accessible to other local transactions as if it is cached data. All the local read operations related to these shared data items will be marked as pseudo-read operations R_p. Any local transactions T_1^i at the mobile host MH_1 that read this shared data item *Z* will develop: (1) an abort-dependency T_2^j *AD* T_1^i with the delegator transaction T_2^j at the mobile host MH_2, i.e., if the delegator transaction T_2^j aborts, the local transactions T_1^i must also abort

because these transactions have read an invalid data value $V_{Z'}$; (2) a commit-dependency T_2^j *CD* T_1^i with the delegator transaction T_2^j, i.e., the delegator transaction T_2^j must commit before transactions T_1^i. Table 6.14 summaries the states of cached data in the local workspaces of the mobile hosts and at the anchor transactions after this mobile data sharing.

Table 6.14: Locks and awareness of sharing modified data states

		MH_1	**MH_2**
Locks	Anchor transaction	$X_R;Y_W$	$Y_R;Z_W$
	Local workspace	$X_R;Y_W;Z_{Rp}$	$Y_R;Z_W$
Conflict awareness	Anchor transaction	$Y_{RW}(T_2^A,T_1^A)^A$	$Y_{RW}(T_2^A,T_1^A)^P$
	Local workspace	$Y_{RW}(T_2^A,T_1^A)^A$ $Z_{Rp}(T_2^j,\text{updated})$	None
Dependency awareness	Anchor transaction	None	None
	Local workspace	$Z(T_2^j,AD)$ $Z(T_2^j,CD)$	None

Case 3: The delegator transaction shares an updated data state and the delegatee transaction upgrades its local cache to the most up-to-date value.

In Figure 6.14, the delegator transaction T_1^i at the mobile host MH_1 updates the data item Y in the local workspace. After this, the delegator transaction T_1^i shares this modified data state $V_{Y'}$ of the shared data item Y to the delegatee transaction T_2^j at the mobile host MH_2. Because this shared data item Y is already cached in the local workspace at the mobile host MH_2, the delegatee transaction T_2^j imports this updated value $V_{Y'}$ of the data item Y to upgrade its local cache to the most up-to-date value.

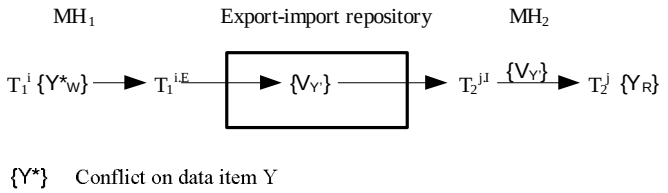

Figure 6.14: Upgrade data state in the local workspace

The conditions for sharing of data state of the delegator transaction T_1^i at the mobile host MH_1 are:

$$(Y_W \in L_1^{WGR}) \wedge (Y \in D_1^i)$$

The following information is attached to the export transaction $T_1^{i.E}$ and logged in the local workspace at the mobile host MH_1 before the export transaction is dispatched to the export-import repository:

$(Y, V_{Y'}, \text{modified}, T_1^i, \text{share\_state}, Y(T_1^i, AD), Y_{RW}(T_2^A, T_1^A)^A)$

As in case 2, if the delegator transaction T_1^i aborts, the shared data value $V_{Y'}$ will become invalid. Therefore, there is an abort-dependency between the delegator transaction T_1^i and the delegatee transactions that will read the modified data state of data item *Y*. The abort-dependency is transferred by the export transaction $T_1^{i.E}$ via the dependency awareness record $Y(T_1^i,AD)$ of data item *Y*. Furthermore, there is an active read-write conflict $Y_{RW}(T_2^A, T_1^A)^A$ on the shared data item *Y* at the mobile host MH_1. This conflict information must also be passed to the delegatee transaction T_2^j that will read the updated value $V_{Y'}$ of the shared data item *Y*. Note that the delegator transaction T_1^i does not know about the delegatee transaction T_2^j at the mobile host MH_2.

The delegatee transaction T_2^j at the mobile host MH_2 will initiate an import transaction $T_2^{j.I}$ to obtain the shared data state $V_{Y'}$ from the export-import repository. The following information is attached to the import transaction $T_2^{j.I}$ and logged in the local workspace before it is dispatched to the export-import repository:

(Y,read_only, none)

When the import transaction $T_2^{j.I}$ commits in the local workspace, the following procedure is carried out at the mobile host MH_2:

- As the mobile host MH_2 is already holding a read lock on the data item *Y*, no pseudo-read lock will be added to the granted read lock set L_2^{RGR} .
- The data value V_Y of the shared data item *Y* in the read data set D_2^{RGR} is updated with the new value $V_{Y'}$.
- A conflict awareness record $Y_R(T_1^i,updated)$ is added to the conflict awareness set Y_{CA} of data item *Y*, i.e.,

  $Y_{CA} := Y_{CA} \cup \{Y_R(T_1^i,updated)\}$
- A conflict awareness record $Y_{RW}(T_2^A, T_1^A)^A$ will also be added to the conflict awareness set Y_{CA} of data item *Y*, i.e.,

  $Y_{CA} := Y_{CA} \cup \{Y_{RW}(T_2^A, T_1^A)^A\}$
- A dependency awareness record $Y(T_1^i,AD)$ is added to the dependency awareness set Y_{DA} of data item *Y*, i.e.,

  $Y_{DA} := Y_{DA} \cup \{Y(T_1^i,AD)\}$

  This dependency awareness record indicates that local transactions T_2^p at the mobile host MH_2 that read data item *Y* will have an abort-dependency with transaction T_1^i, i.e., T_1^i *AD* T_2^p. Note that the locally committed transactions T_2^k at the mobile host MH_2 that have read the original value V_Y will not be affected by this abort-dependency.
- A dependency awareness record $Y(T_1^i,CD)$ is further added to the dependency awareness set Y_{DA} of data item *Y*, i.e.,

  $Y_{DA} := Y_{DA} \cup \{Y(T_1^i,CD)\}$

  This dependency awareness record indicates that local transactions T_2^p at the mobile host MH_2 that read data item *Y* will have a commit-dependency with transaction T_1^i, i.e., T_1^i *CD* T_2^p. Note that the locally committed transactions T_2^k at the mobile host MH_2 that have read the original value V_Y will not be affected by this commit-dependency.

The new conflict awareness and dependency awareness records have the following meanings: (1) any local transaction T_2^p at the mobile host MH_2 that reads the upgraded shared data item Y will develop an abort-dependency (T_1^i AD T_2^p) with the delegator transaction T_1^i at the mobile host MH_1; (2) the local transactions T_2^p will also develop a commit-dependency (T_1^i AD T_2^p) with the delegator transaction T_1^i; and (3) the local transactions T_2^p must be aware that it can conflict with other local transactions T_l^l at the mobile host MH_1 (for example, the local transaction T_l^l at the mobile host MH_1 subsequently modifies the shared data item Y after the delegator transaction T_1^i). These transaction dependencies and execution constraints (explained in Section 6.5) will be reconciled at the transaction integration stage (see Section 6.6).

Table 6.15 summaries the states of cached data in the local workspaces of the mobile hosts and at the anchor transactions after this mobile data sharing. Note that the conflict awareness on the shared data item Y is an active conflict at the disconnected mobile host MH_2, while the anchor transaction T_2^A at the database servers is still holding a passive conflict awareness.

Table 6.15: Locks and awareness of upgrading data states

		MH_1	**MH_2**
Locks	Anchor transaction	$X_R;Y_W$	$Y_R; Z_W$
	Local workspace	$X_R;Y_W$	$Y_R; Z_W$
Conflict awareness	Anchor transaction	$Y_{RW}(T_2^A,T_1^A)^A$	$Y_{RW}(T_2^A,T_1^A)^P$
	Local workspace	$Y_{RW}(T_2^A,T_1^A)^A$	$Y_R(T_1^i,\text{updated})$ $Y_{RW}(T_2^A,T_1^A)^A$
Dependency awareness	Anchor transaction	None	None
	Local workspace	None	$Y(T_1^i,AD)$ $Y(T_1^i,CD)$

6.4.2 Management of sharing data status

In this section, we will formalize the sharing of mobile data status (i.e., locks) among standard transactions at different mobile hosts. For mobile sharing status, a delegator transaction T_i^{Dor} shares its locks to a delegatee transaction T_j^{Dee}. The sharing of data status means that the delegator transaction T_i^{Dor} no longer holds the responsibility of the shared data items. The delegator transaction T_i^{Dor} at the mobile host MH_i carries out a mobile data sharing status procedure when it wants to delegate the locks on shared data items and allows the delegatee transactions T_j^{Dee} at the mobile host MH_j to take over the control of the delegated locks.

Figure 6.15 summaries the mobile data sharing status between a delegator and a delegatee transaction. As discussed in Section 5.5.5, depending on status of the shared data (i.e., read or write lock) that is cached in the local workspace, the delegator transaction T_i^{Dor} can delegate either the read or the write lock on the shared data to the delegatee transaction T_j^{Dee}. Furthermore, if a shared data item is originally write locked in the local

workspace, the delegator transaction can delegate this write lock but keep the read lock on the shared data item, i.e., the delegator transaction performs the downgrading lock operations. For the delegatee transaction T_j^{Dee}, it can obtain the delegated lock as a new lock in the local workspace. If a shared data item is already cached with read lock at the mobile host, and the delegator transaction T_j^{Dor} delegates the write lock on this shared data item, the delegatee transaction T_i^{Dee} can upgrade the control of the shared data item from read lock to write lock.

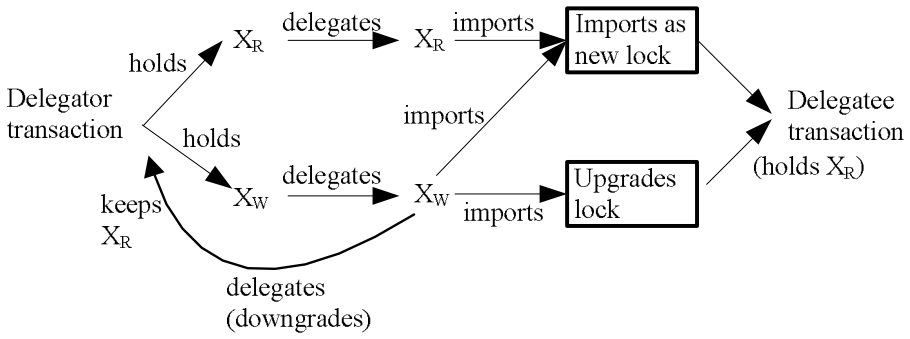

Figure 6.15: Sharing data status

Figure 6.16 illustrates an example for mobile data sharing operations among mobile transactions at two mobile hosts MH_1 and MH_2. The example will be used to illustrate our analysis of the mobile data sharing status in this section.

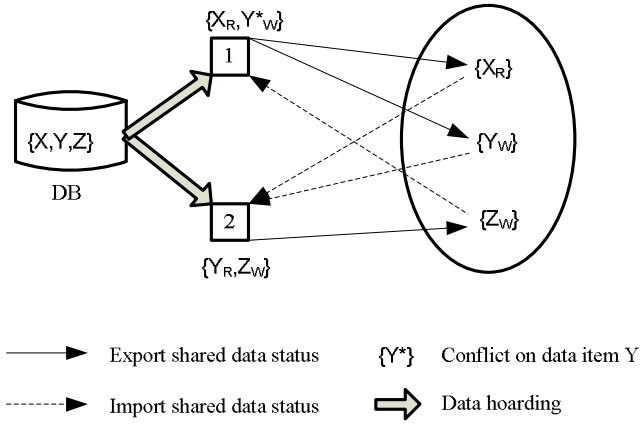

Figure 6.16: Sharing data status between mobile hosts

In the example, the anchor transaction T_1^A of the mobile host MH_1 holds a non-conflict read lock X_R on the shared data item X, and an active read-write conflict $Y_{RW}(T_2^A, T_1^A)^A$ on the shared data item Y (i.e., with a write lock Y_W on Y) with the anchor transaction T_2^A of the mobile host MH_2. The anchor transaction T_2^A of the mobile host MH_2 holds a passive read-write conflict $Y_{RW}(T_2^A, T_1^A)^P$ on data item Y (i.e., with a read lock Y_R on Y) and a write lock Z_W on data item Z. During the mobile data sharing status, a delegator transaction T_1^i at the mobile host MH_1 will delegate the read lock X_R and the write lock Y_W on the shared data items X and Y, respectively. A delegator transaction T_2^j at the

mobile host MH_2 will delegate the write lock Z_W on the shared data item Z, but the read lock Z_R on this data item will be retained at this mobile host. At the same time, a delegatee transaction T_l^l at the mobile host MH_1 will obtain the delegated write lock Z_W on the shared data item Z as a new lock. At the mobile host MH_2, a delegatee transaction T_2^k imports the read lock X_R on the shared data item X as a new lock and the write lock Y_W on the shared data item Y as an upgraded lock.

The locks and conflict awareness records held by the anchor transactions at the database servers and in the local workspaces at the disconnected mobile hosts, as well as the mobile data sharing states are summarized in Table 6.16.

Table 6.16: Locks and data conflict awareness of sharing data status scenarios

		MH_1	**MH_2**
Locks	Anchor transaction	$X_R;Y_W$	$Y_R;Z_W$
	Local workspace	$X_R;Y_W$	$Y_R;Z_W$
Conflict awareness	Anchor transaction	$Y_{RW}(T_2^A,T_1^A)^A$	$Y_{RW}(T_2^A,T_1^A)^P$
	Local workspace	$Y_{RW}(T_2^A,T_1^A)^A$	None
Mobile data sharing	Exported data status	X_R, Y_W	Z_W (delegate and downgrade)
	Imported data status	Z_W	X_R, Y_W (upgrade)

Conditions of sharing data status

In order to be able to delegate the data status of the shared data item X, a delegator transaction T_i^{Dor} at the mobile host MH_i must fulfill the following conditions:

(1) For sharing the read lock X_R
- The shared data item X must be cached at the mobile host with a read lock X_R (the pseudo-read lock X_{Rp} can not be shared), i.e., $X_R \in L_i^{RGR}$.
- There is no other local transaction T_i^k that accesses the data item X when the delegator transaction T_i^{Dor} shares this read lock, i.e.,
  $\forall T_i^k,\ T_i^{Dor} \neq T_i^k,\ (X \in D_i^{Dor}) \wedge (X \notin D_i^k)$
  If there is another transaction T_i^k that holds the read lock X_R on the shared data item X, the exporting read lock process will be delayed or redirected (see Section 6.4.3).

(2) For sharing the write lock X_W
- The shared data item X must be cached at the mobile host with a write lock, i.e., $X_W \in L_i^{W_{GR}}$.
- Data item X belongs to the write data set of the delegator transaction T_i^{Dor}, i.e., $X \in D_i^{Dor}$. This means that there is no other transaction T_i^k that is concurrently accessing this data item X.
- All local transactions T_i^k that have updated data item X must be aborted. These aborts can lead to the abortion of local transactions that have accessed the

updated data item X. However, in case of downgrading locks, the local transactions T_i^k, which have read the original data value V_X of the data item X, will not be aborted. These transactions will develop a read-write conflict with a delegatee transaction T_j^{Dee} at the mobile host MH_j that (later) imports the shared write lock X_W.

Operations of sharing data status

When the delegator transaction T_i^{Dor} at the mobile host MH_i relinquishes the lock of the data item X, the procedure of exporting shared data status is implemented as follows:

(1) The delegator transaction T_i^{Dor} initiates an export transaction $T_i^{Dor.E}$ that will export the shared data status into the export-import repository.

(2) The cached data set and the replicated granted lock set at the mobile host MH_i will be updated. If this sharing status operation is a downgrading lock operation, the delegator transaction T_i^{Dor} will modify the lock status of the shared data item X in the local workspace from X_W to X_R.

(3) For each shared data item X, attach all associated information to the export transaction $T_i^{Dor.E}$ (see Table 6.17). The associated information of the export transaction $T_i^{Dor.E}$ is also logged in the local workspace at the mobile host MH_i.

(4) The export transaction $T_i^{Dor.E}$ is dispatched to the export-import repository.

Table 6.17: Data structure for exporting data status

Attribute	Description
ItemID	The identification of the shared data item
ItemValue	The shared value of the shared data item
TypeOfStatus	The type of sharing data status is either *read* or *write* lock
DelegatorID	The identification of the delegator transaction
ItemDepend	The dependency awareness related to the shared data item
ItemConflict	The conflict awareness related to the shared data item

The data structure for the shared data status contains all the correlated information. Again, the attached information is associated with individual shared data items. Therefore, the mobile data sharing status mechanism allows different status of the shared data item to be shared in the mobile sharing workspaces. As a result, the delegatee transactions can select which shared data status to be obtained, i.e., read or write status.

When a delegatee transaction T_i^{Dee} at the mobile host MH_i wants to take the control of a shared data item, the delegatee transaction T_i^{Dee} will initiate an import transaction $T_i^{Dee.I}$ that will obtain the status of the shared data from the export-import repository. The delegatee transaction must specify what type of status of a shared data item that it wants

to import, i.e., read or write lock. When the wanted data status is obtained, the delegatee transaction will also be aware of and handle any data conflicts related to the shared data.

When the delegatee transaction T_i^{Dee} at the mobile host MH_i imports the status of the shared data item X, the procedure of importing shared data status is implemented as follows:

(1) The delegatee transaction T_i^{Dee} initiates an import transaction $T_i^{Dee.I}$ that will import the control of the needed shared data from the export-import repository to the local workspace.

(2) All necessary information related to the wanted shared data (see Table 6.18) is attached to the import transaction $T_i^{Dee.I}$. This information is also written to a log in the local workspace.

(3) The import transaction $T_i^{Dee.I}$ is dispatched to the export-import repository.

Table 6.18: Data structure for importing data status

Attribute	Description
ItemID	The identification of the shared data item
TypeOfShare	The type of data sharing is either *read* or *write* lock
TransDepend	The transaction dependency between the delegatee and the import transaction(s) (i.e., *abort-dependency* or *multiple-abort-dependency*)
StructDepend	The structural dependency between the delegatee and the import transaction(s) (*merge* or *adopt*)

The import transaction $T_i^{Dee.I}$ will retrieve from the export-import repository the wanted data item. When the needed data is completely obtained, depending on the structural dependency between the delegatee and the import transactions, the import transaction can either commit or *merge* with or be *adopted* into the delegatee transaction (see Section 5.4.2).

Before the collected shared data item X is made available to other local transactions, the following procedure is carried out:

(1) If the obtained shared data item is not cached in the local workspace, this shared data item is added to the local cache as a new data.

(2) If the status of the shared data item is read lock, a read lock X_R will be added to the replicated granted read lock set L_i^{RGR}.

(3) If the status of the shared data item is write lock, and the shared data item is a newly cached data, a write lock X_W will be added to the replicated granted write

lock set L_i^{WGR} . If this shared data is already cached with a read lock, i.e., $X_R \in L_i^{RGR}$ at the mobile host, the read lock will be upgraded to the write lock.

(4) If there is any conflict awareness or dependency awareness related to the obtained data status, the conflict awareness or dependency awareness records will be added to the conflict awareness set X_{CA} and the dependency awareness set X_{DA}, respectively.

Depending on how the delegator transaction delegates locks to the delegatee transaction (relinquishing locks or downgrade locks), and how the delegatee transaction imports these shared locks (as new locks or upgraded locks), there are four different sharing data status scenarios that are grouped into three different cases (see Table 6.19). These examples build on those in Figure 6.16. Note that for sharing data status, dependency awareness does not occur.

Table 6.19: Sharing data status scenarios

Case	Delegator transaction	Delegatee transaction
4	Holds and delegates read lock	Imports the shared read lock as a new lock
	Holds and delegates write lock	Imports the shared write lock as a new lock
5	Holds and delegates write lock	Imports the shared write lock as an upgraded lock
6	Holds write lock and downgrades to read lock	Imports the shared write lock as a new or an upgraded lock

Case 4: The delegator transaction shares a read lock or a write lock, and the delegatee transaction imports the shared lock as a new lock.

Figure 6.17 illustrates examples of sharing data status between local transactions at mobile hosts MH_1 and MH_2. The delegator transaction T_1^i at the mobile host MH_1 holds a read lock X_R on the shared data item X and shares this read lock to the delegatee transaction T_2^j at the mobile host MH_2. The delegator transaction T_2^k at the mobile host MH_2 holds a write lock Z_W on the shared data item Z and shares this write lock to the delegatee transaction T_1^l at the mobile host MH_1. Both the delegatee transactions T_2^j and T_1^l import the shared locks as new locks.

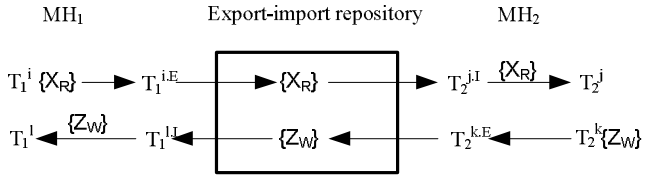

Figure 6.17: Delegating locks

The conditions for sharing of data status of two delegator transactions T_1^i and T_2^k are:

- For the delegator transaction T_1^i: $X_R \in L_1^{RGR} \wedge \forall T_1^n,\ T_1^i \neq T_i^n,\ (X \in D_1^i) \wedge (X \notin D_1^n)$
- For the delegator transaction T_2^k: $Z_W \in L_2^{WGR} \wedge \forall T_2^m,\ T_2^k \neq T_2^m,\ (Z \in D_2^k) \wedge (Z \notin D_2^m)$

The delegator transactions T_1^i and T_2^k will update the states of the local workspaces at the mobile host MH_1 and MH_2 before the shared data status operations are carried out. The following procedures are performed:

- At mobile host MH_1: $D_1^{RGR} := D_1^{RGR} \setminus \{X\} \wedge L_1^{RGR} := L_1^{RGR} \setminus \{X_R\}$
- At mobile host MH_2: $D_2^{WGR} := D_2^{WGR} \setminus \{Z\} \wedge L_2^{WGR} := L_2^{WGR} \setminus \{Z_W\}$

After these operations, the shared data items X and Z are not accessible in the mobile hosts MH_1 and MH_2, respectively.

The following information is attached to the export transactions $T_1^{i.E}$ and $T_2^{k.E}$ and logged in the local workspaces at the mobile hosts before these export transactions are dispatched to the export-import repository:

- For the export transaction $T_1^{i.E}$: (X, V_X, read, T_1^i, none, none)
- For the export transaction $T_2^{k.E}$: (Z, V_Z, write, T_2^k, none, none)

Note that there is no transaction dependency between the delegator transactions and the export transactions. The responsibility of the shared data items X and Z are completely transferred from the delegator transaction to the delegatee transaction via shared transactions.

The delegatee transactions T_1^l and T_2^j will obtain these shared data status via the import transactions $T_1^{l.I}$ and $T_2^{j.I}$. The import transaction $T_1^{l.I}$ will merge with the delegatee transaction T_1^l (which is a flat transaction – if the delegatee transaction has a nested structure, the import transaction will be adopted as a sub-transaction) to ensure that the shared data item Z (with a write lock) will be accessed first by this delegatee transaction. The import transaction $T_2^{j.I}$ can commit in the local workspace at the mobile host MH_2 regardless of the state of the delegatee transaction T_2^j because the imported data item X is read only.

The following information is attached to the import transactions $T_1^{l.I}$ and $T_2^{j.I}$ and logged in the local workspaces before these import transactions are dispatched to the export-import repository:

- For the import transaction $T_1^{l.I}$: (X,read, none, none)
- For the import transaction $T_2^{j.I}$: (Z,write, none, merge)

When these import transactions commit in the local workspaces, the following procedures are carried out:

- At mobile host MH_1:
    - A write lock Z_W is added to the granted write lock set, i.e.,
      $L_1^{WGR} := L_1^{WGR} \cup \{Z_W\}$

- The shared data item Z is added as a new modifiable data item, i.e.,
  $D_1^{WGR} := D_1^{WGR} \cup \{Z\}$
- A conflict awareness record $Z_W(T_2^k,status)$ is added to the conflict awareness set Z_{CA} of data item Z, i.e.,
  $Z_{CA} := Z_{CA} \cup \{Z_W(T_2^k,status)\}$

- At mobile host MH_2:
  - A real read lock X_R is added to the granted read lock set, i.e.,
    $L_2^{RGR} := L_2^{RGR} \cup \{X_R\}$
  - The shared data item X is added as a new read only data item, i.e.,
    $D_2^{RGR} := D_2^{RGR} \cup \{X\}$
  - A conflict awareness record $X_R(T_1^i,status)$ is added to the conflict awareness set X_{CA} of data item X, i.e.,
    $X_{CA} := X_{CA} \cup \{X_R(T_1^i,status)\}$

These conflict awareness records will be used in the transaction integration stage for synchronizing conflicting locks between anchor transactions (see Section 6.6). After these operations are completed, the obtained data items are accessible to the delegatee and other local transactions as if they are cached data. Table 6.20 summaries the states of cached data in the local workspaces and at the anchor transactions after this mobile data sharing.

Table 6.20: Locks and awareness of delegating locks

		MH_1	**MH_2**
Locks	Anchor transaction	X_R; Y_W	Y_R; Z_W
	Local workspace	Y_W; Z_W	X_R; Y_R
Conflict awareness	Anchor transaction	$Y_{RW}(T_2^A,T_1^A)^A$	$Y_{RW}(T_2^A,T_1^A)^P$
	Local workspace	$Y_{RW}(T_2^A,T_1^A)^A$ $Z_W(T_2^k,status)$	$X_R(T_1^i,status)$

Case 5: The delegator transaction shares a write lock and the delegatee transaction imports the shared write lock to upgrade from read lock to write lock.

Figure 6.18 illustrates an example of upgrading the status of a shared data item from a read lock to a write lock. The delegator transaction T_1^i at the mobile host MH_1 delegates the write lock Y_W on the shared data item Y to the delegatee transaction T_2^j at the mobile host MH_2. However, at the mobile host MH_2, the shared data item Y is already cached as a read-only data, i.e., with a read lock Y_R. Therefore, the delegatee transaction T_2^j will upgrade the status of the shared data item Y from read lock to write lock.

The conditions for sharing of data status of the delegator transaction T_1^i are:

$$Y_W \in L_1^{WGR} \wedge \forall T_1^n,\ T_1^i \neq T_1^n,\ (Y \in D_1^i) \wedge (Y \notin D_1^n)$$

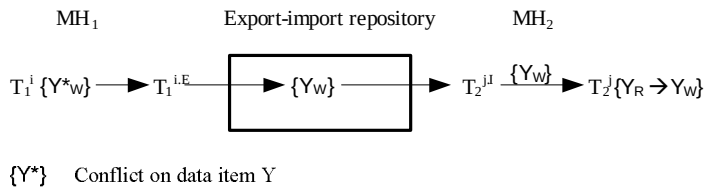

Figure 6.18: Upgrading locks

Before the shared data status operations are carried out, the delegator transaction T_1^i will modify the write data set and lock set at mobile host MH_1 as follows:

$$D_1^{WGR} = D_1^{WGR} \setminus \{Y\} \wedge L_1^{WGR} = L_1^{WGR} \setminus \{Y_W\}$$

After these operations, the shared data item Y is no longer accessible in the mobile host MH_1.

The following information is attached to the export transaction $T_1^{i.E}$ and logged in the local workspace before this export transaction is dispatched to the export-import repository:

$(Y, V_Y, \text{write}, T_1^i, \text{none}, Y_{RW}(T_2^A, T_1^A)^A)$

Note that in the local workspace at mobile host MH_1, there is an active read-write conflict related to the shared data item Y, i.e., $Y_{RW}(T_2^A, T_1^A)^A$. This conflict awareness must also be passed to the delegatee transaction T_2^j at mobile host MH_2.

The delegatee transaction T_2^j will obtain the shared data status via the import transaction $T_2^{j.I}$. As in case 4, the import transaction $T_2^{j.I}$ will merge with the delegatee transaction T_2^j (which is a flat transaction – if the delegatee transaction T_2^j has a nested structure, the import transaction $T_2^{j.I}$ will be adopted as a sub-transaction) to ensure that the shared data item Y (with a write lock) will be accessed first by this delegatee transaction.

The following information is attached to the import transaction $T_2^{j.I}$ and logged in the local workspace before it is dispatched to the export-import repository:

(Y,write, none, merge)

When the import transaction $T_2^{j.I}$ commits in the local workspace, the following operations are carried out at the mobile host MH_2:

- The read lock Y_R on the shared data item Y in the granted read lock set is removed. A new write lock on the data item Y is added to the granted write lock set, i.e.,
  $$L_2^{RGR} := L_2^{RGR} \setminus \{Y_R\} \wedge L_2^{WGR} := L_2^{WGR} \cup \{Y_W\}$$
- The shared data item Y is removed from the read data set and added to the write data set, i.e.,
  $$D_2^{RGR} := D_2^{RGR} \setminus \{Y\} \wedge D_2^{WGR} := D_2^{WGR} \cup \{Y\}$$
- A conflict awareness record $Y_W(T_1^i, status)$ is added to the conflict awareness set Y_{CA} of data item Y, i.e.,
  $$Y_{CA} := Y_{CA} \cup \{Y_W(T_1^i, status)\}$$

A conflict awareness record $Y_{RW}(T_2^A, T_1^A)^A$ associated with data item Y is not added to the conflict awareness set Y_{CA} because the mobile host MH_2 already holds the read lock Y_R on the shared data item Y before the sharing data status. However, if the conflict awareness is related to another anchor transaction T_3^A of mobile host MH_3, a conflict awareness record will be added to the conflict awareness set Y_{CA} so that the local transactions at the mobile host MH_2 will be aware of conflicts with local transactions at mobile host MH_3.

The new conflict awareness record $Y_W(T_1^i, status)$ has the following meaning: the mobile host MH_2 has obtained a write lock Y_W on the shared data item Y from the mobile host MH_1 via the delegator transaction T_1^i. This record will be used at the transaction integration stage to solve conflicts between anchor transactions of the mobile hosts (see Section 6.6).

Table 6.21 summaries the states of cached data in the local workspaces and at the anchor transactions after this mobile data sharing.

Table 6.21: Locks and awareness of upgrading locks

		MH_1	**MH_2**
Locks	Anchor transaction	$X_R; Y_W$	$Y_R; Z_W$
	Local workspace	X_R	$Y_W; Z_W$
Conflict awareness	Anchor transaction	$Y_{RW}(T_2^A, T_1^A)^A$	$Y_{RW}(T_2^A, T_1^A)^P$
	Local workspace	None	$Y_W(T_1^i, status)$

Case 6: The delegator transaction downgrades the status of the shared data item from write lock to read lock.

Figure 6.19 illustrates an example of downgrading the status of a shared data item from a write lock to a read lock. The delegator transaction T_2^j at the mobile host MH_2 delegates the write lock Z_W on the shared data item Z to the delegatee transaction T_1^i at the mobile host MH_1. However, the delegator transaction T_2^j is holding a read permission on the shared data item Z. In other words, the delegator transaction T_2^j will downgrade the write lock status on data item Z to read lock status. This may be due to the fact that the delegator transaction T_2^j does not need to read the data item Z, but there may be other local transactions T_2^k at mobile host MH_2 that need to read this data item Z. The delegatee transaction T_1^i at the mobile host MH_1 can obtain the shared write lock as either a new lock or an upgraded lock (see Cases 4 and 5 above). Here, we will focus on the changes in the local workspace at the mobile host MH_2.

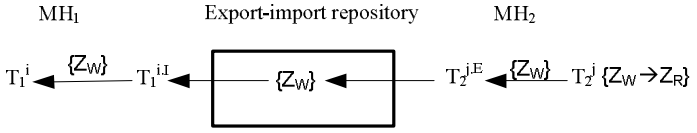

Figure 6.19: Downgrading locks

The conditions for sharing of data status of the delegator transaction T_2^j are:

$$Z_W \in L_2^{WGR} \wedge \forall T_2^m,\ T_2^j \neq T_2^m,\ (Z \in D_2^j) \wedge (Z \notin D_2^m)$$

Before the write lock Z_W is placed into the export transaction $T_2^{j.E}$, the delegator transaction T_2^j will modify the write data set and lock set at the mobile host MH_2 as follows:

- The write lock Z_W on the shared data item Z in the granted write lock set is removed. A new read lock Z_R on the data item Z will be added to the granted read lock set, i.e.,
  $L_2^{WGR} := L_2^{WGR} \setminus \{Z_W\} \wedge L_2^{RGR} := L_2^{RGR} \cup \{Z_R\}$
- The shared data item Z is removed from the write data set and added to the read data set, i.e.,
  $D_2^{WGR} := D_2^{WGR} \setminus \{Z\} \wedge D_2^{RGR} := D_2^{RGR} \cup \{Z\}$
- A conflict awareness $Z_{RW}(T_2^A, T_1^A)^A$ is added to the conflict awareness set Z_{CA} of data item Z so that local transactions at the mobile host MH_2 will be aware of access conflicts on the shared data item Z with other local transactions at the mobile host MH_1, i.e.,
  $Z_{CA} := Z_{CA} \cup \{Z_{RW}(T_2^A, T_1^A)^A\}$

After these operations, the shared data item Z is read-only accessible in the local workspace at the mobile host MH_2. Any local transaction T_2^n at the mobile host MH_2 that has read the original data value V_Z of the shared data item Z will develop a read-write conflict with the transaction T_1^i at the mobile host MH_1. The states of cached data in the local workspace and at the anchor transaction of the mobile hosts MH_1 and MH_2 after this mobile data sharing are summarized in Table 6.22. Note that, in general, the locks and conflict awareness in the local workspace of the mobile host MH_1 will have to depend on whether the delegated write lock is imported as a new lock (Case 4) or as an upgraded lock (Case 5).

Table 6.22: Locks and awareness of downgrading locks

		MH_1	**MH_2**
Locks	Anchor transaction	$X_R; Y_W$	$Y_R; Z_W$
	Local workspace	…	$Y_R; Z_R$
Conflict awareness	Anchor transaction	$Y_{RW}(T_2^A, T_1^A)^A$	$Y_{RW}(T_2^A, T_1^A)^P$
	Local workspace	…	$Z_{RW}(T_2^A, T_1^A)^A$

6.4.3 Redirect sharing operations

In Section 6.4.2, the condition for sharing locks between mobile transactions requires that: for a lock to be shared, there must be no other local transaction that is holding the same lock. When the condition is not met, the delegator transaction will not be able to share the locks to the delegatee transactions. In other words, the sharing data status will be delayed. In Figure 6.20, at mobile host MH_1, both transactions T_1^1 and T_1^2 are accessing the shared data item X. Meanwhile, the delegator transaction T_1^1 is also in need to share the control of the data item X to the delegatee transaction T_2^1 at the mobile host MH_2. Until the local transaction T_1^2 releases its lock on X, the delegator transaction T_1^1 will not be able to share the status of the data item X to the delegatee transaction T_2^1. The question is that: what happens if the delegator transaction T_1^1 commits before it can share the read lock X_R on the data item X to the delegatee transaction T_2^1?

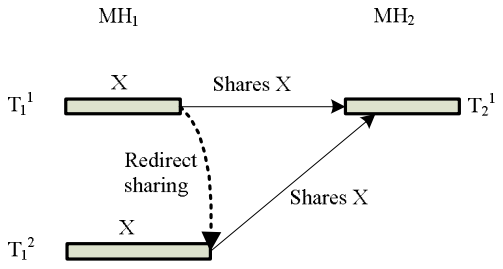

Figure 6.20: Redirect sharing of data

In order to ensure that the sharing status of the shared data item *X* will eventually be carried out, we define a method to redirect the sharing status operations from one transaction to another. The method

`RedirectShare(Ti^k, Tj^l, TransType, ObjSet)`

will transfer the responsibility of sharing data status from the delegator transaction T_i^k to the delegatee transaction T_j^l. The *TransType* is the type of shared transaction that can be either an export or an import transaction. The *ObjSet* is the set of shared data that is needed to be shared.

The same procedure will be applied for the mobile transactions that have nested structure. The sharing of data via export or import transactions must be taken care of because other transactions in the hierarchical structure may be affected. Figure 6.21 illustrates the scenario of redirect sharing of data in a nested transaction. In this example, sub-transaction $T_1^{1.1.1}$ at the mobile host MH_1 needs to share data status with transaction T_2^1 at the mobile host MH_2; and sub-transaction $T_1^{1.2}$ needs to share data status with transaction T_3^1 at mobile host MH_3.

If the sub-transaction $T_1^{1.2}$ carries out the export data status operations on shared data item *X*, it can impact on the execution of other sub-transactions like $T_1^{1.1}$. Therefore, the sharing data status process is redirected to the top-level transaction T_1^1 for that to decide when it will be carried out. If the sub-transaction $T_1^{1.1.1}$ wants its imported data status to be accessible to other sub-transactions like $T_1^{1.1.2}$, it will redirect the sharing data status process to its parent transaction $T_1^{1.1}$.

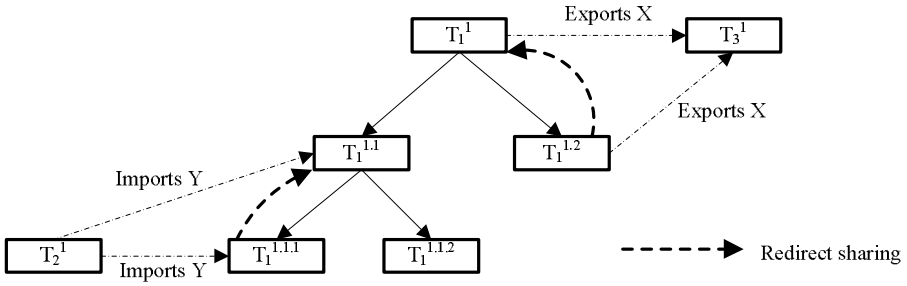

Figure 6.21: Redirect sharing of sub-transactions

6.5 Disconnected transaction processing stage

In this section, we focus our discussion on the transaction processing at the disconnected mobile hosts, i.e., disconnected transaction processing. To recap, the anchor transaction of each mobile host plays a role as top level transaction of an open nested transaction structure. This means that all other local transactions (i.e., standard transactions) are the sub-transactions of this anchor transaction, and these local transactions can commit or abort without any affect in relation to the anchor transaction.

Shared data is cached in the local workspace with all related information - that are: the state, the status, the conflict awareness and the dependency awareness (see Section 6.3.2). Local transactions at the disconnected mobile host are carried out like online transactions are at the database servers. And the transaction manager at the mobile host makes use of the two-phase locking protocol provided by the lock manager to ensure that local transactions are serializable. The local lock manager accepts lock requests from local transactions. If the lock request is legal, the requested lock will be granted to the local transactions. For example, if a local transaction requests a write lock on a data item that is read-only cached in the local workspace, the request is denied and this transaction is aborted.

When a local transaction commits, the locally committed results are visible to all local transactions. When the mobile host reconnects to the database servers, these locally committed transactions will be synchronized with other transactions. Depending on the characteristics of the cached data (explained in Section 6.5.1), the locally committed transactions are either allowed to finally commit at the database servers, or aborted (see Section 6.6 for transaction integration stage). The abortion of one local transaction can lead to abort of other local transactions that have read the results of the aborted transaction.

6.5.1 Constraint and non-constraint cached data

The disconnected transaction processing at the mobile host is carried out based on the actual data sets that have been successfully cached during the data hoarding stage or have been obtained through the mobile data sharing stage. There are two types of cached data at the local mobile host: non-constraint and constraint.

> **Definition (*non-constraint cached data*).** *A cached data item X is non-constraint if it does not represent any conflict awareness nor any dependency awareness, i.e.,*
>
> $X_{CA} = \varnothing \wedge X_{DA} = \varnothing$

> **Definition (*constraint cached data*).** *A cached data item X is constraint if it represents either some conflict awareness or some dependency awareness, i.e.,*
>
> $X_{CA} \neq \varnothing \vee X_{DA} \neq \varnothing$

The *non-constraint cached data* is shared data that is being considered by the local transactions as consistent data, and there is no transaction at other mobile hosts that is performing conflicting operations on this cached data. In other words, the local transactions that access non-constraint cached data will not hold any dependency with other local transactions at other hosts.

The *constraint cached data* is cached data that will cause execution dependencies among transactions that access this shared data. In other words, when local transactions access constraint cached data, they have to be aware that there are other local transactions at other mobile hosts that are currently accessing and potentially performing conflicting operations on these shared data.

In the next sub-sections, we will discuss the disconnected transaction processing of local transactions that operate on non-constraint and constraint shared data that is cached at the disconnected mobile host.

6.5.2 Local transactions operate on non-constraint cached data

For local transactions that operate on non-constraint cached data and hold no structural dependency with other local transactions at other mobile hosts, if these transactions commit, these transactions will eventually be allowed to finally commit at the database servers.

The mobile host MH_i will keep a set *LocalCommitted* (LC_i) of locally committed transactions (this *LocalCommitted* set is initially an empty set, i.e., $LC_i = \varnothing$).

> **Definition (*local committed transaction set*).** *A locally committed transaction set $LC_i = \{T_i^j \mid T_i^j$ is a locally committed transaction$\}$ is a partially ordered set with a partial order relation $<_l$, i.e.,*
>
> $\forall\, T_i^k, T_i^l \in LC_i$, *either* $T_i^k <_l T_i^l$ *or* $T_i^l <_l T_i^k$

When a local transaction T_i^k, which only accesses non-constraint data, requests to commit, if none of the operations of this local transaction involves a local conflict within the scope of the local workspace at the mobile host, the local transaction T_i^k will be allowed to locally commit at the mobile host. The locally committed transaction T_i^k will be added to the locally committed transaction set LC_i, i.e.,

$$LC_i := LC_i \cup \{T_i^k\}$$

In Figure 6.22, initially the local transactions at the mobile host MH_1 do not know about the conflict on the shared data item *Y*, which is cached with a write lock in the local workspace, with other transactions at the mobile host MH_2. This is because that at the time the mobile host MH_1 disconnects from the database servers the anchor transaction T_1^A does not hold any conflict. Therefore, all the local transactions at the mobile host MH_1 will think that they are operating on the non-conflict data item *Y*. When these local transactions commit locally, they will be allowed to finally commit at the database servers when the mobile host MH_1 reconnects to the database servers. The local transaction manager at the mobile host MH_1 will keep track of the order of the locally

committed transactions T_1^1, T_1^2 and T_1^3, i.e., $LC_i = \{T_1^1 < T_1^2 < T_1^3\}$. If the anchor transaction T_1^A holds any passive conflicts with other transactions at the mobile host MH_2, these conflicts are only known when the mobile host MH_1 reconnects to the database servers. On the other hand, a local transaction T_2^1 at the mobile host MH_2 is aware of potential conflicts on shared data item Y. However, the local transaction T_2^1 does not know exactly which transactions in the mobile host MH_2 it is conflicting with. When the local transactions of mobile host MH_1 are finally committed in the database servers, the conflict awareness record Y_{CA} held by the anchor transaction T_2^A will be modified so that local transaction T_2^1 at mobile host MH_2 can be correctly scheduled in the global workspace, for example $T_2^1 < T_1^1 < T_1^2 < T_1^3$ (this will be explained in Section 6.6).

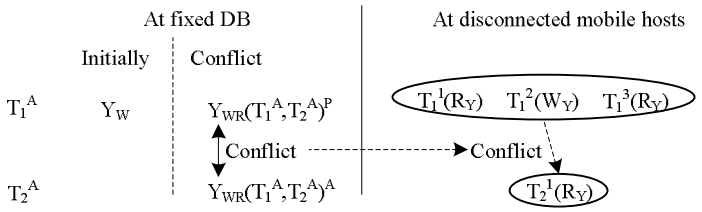

Figure 6.22: Disconnected transaction processing with accessing conflict

6.5.3 Local transactions operate on constraint cached data

When a local transaction at the disconnected mobile host accesses constraint cached data, the conflict awareness on shared data will produce execution constraints (discussed below); while the dependency awareness will produce transaction dependencies (see Figure 6.23).

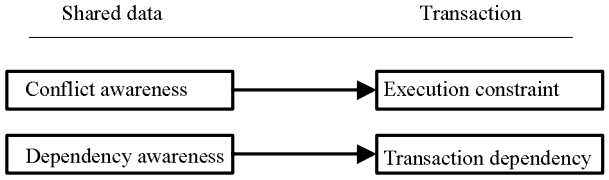

Figure 6.23: Effects of shared data on transactions

Local transactions access cached data with conflict awareness

To recap, for a data item X that is cached in the local workspace at the mobile host, the conflict awareness set X_{CA} keeps track of all the potential conflicts that could occur when a transaction accesses this data item. Among these conflict records, only the conflict records associated with the data hoarding stage and sharing data states (i.e., the read-write conflict, write-read conflict, and the pseudo-read records) will produce execution constraints among transactions. Other conflict records, i.e., conflicts that occur with sharing data status (see Section 6.4.2), do not cause any execution constraints.

We define the execution constraint among transactions that access constraint cached data as follows:

> **Definition (*execution constraint*).** *A transaction T^i is said to be scheduled before a transaction T^j, denoted by $T^i \rightarrow T^j$, if all the conflicting operations Op_i of transaction T^i is executed before the conflicting operations Op_j of transaction T^j, i.e.,*
>
> $$T^i \rightarrow T^j \Leftrightarrow (\forall Op_i \in T^i, Op_j \in T^j, Conflict(Op_i, Op_j) \Rightarrow Op_i \rightarrow Op_j)$$

The execution constraint rules associated with read-write and write-read conflicts are:

> **Rule 1 (*execution constraint of rw-conflict*)**: *If transaction T_i^k develops a read-write conflict with transaction T_j^l on shared data X, i.e., transaction T_j^l will modify the shared data X offline after it is being read by transaction T_i^k, transaction T_i^k will be scheduled before transaction T_j^l, i.e., $T_i^k \rightarrow T_j^l$.*

> **Rule *2* (*execution constraint of wr-conflict*)**: *If transaction T_i^k develops a write-read conflict with transaction T_j^l on shared data X, i.e., transaction T_i^k will read the shared data X after it is being modified offline by transaction T_j^l, transaction T_i^k will be scheduled before transaction T_j^l, i.e., $T_i^k \rightarrow T_j^l$.*

During the mobile data sharing stage, a delegator transaction shares either the original data state or the updated data state to the delegatee transaction. These sharing data states imply an execution constraint between the delegator and the delegatee transactions. The following rules define these kinds of execution constraints between mobile transactions:

> **Rule *3* (*execution constraint of sharing original data state*)**: *If delegator transaction T_i^k shares an original data state to delegatee transaction T_j^l, transaction T_j^l must be scheduled before transaction T_i^k, i.e., $T_j^l \rightarrow T_i^k$.*

This rule describes the mobile sharing data states scenario in which a delegator transaction T_i^k shares an original data state V_X of the data item X to a delegatee transaction T_j^l. The delegator transaction T_i^k can hold a read lock, or a write lock on the shared data item but the shared data state has not been modified. In this scenario, both the delegator T_i^k and delegatee T_j^l transactions read the same value V_X of data item X. If the delegator transaction T_i^k reads a consistent data value of X, then the delegatee transaction T_j^l will be assured to read the same consistent data value as the delegator transaction T_i^k.

If the delegator transaction T_i^k holds a read lock X_R on X and there is another transaction T_x^y (at a different mobile host) with which the delegator transaction T_i^k holds a read-write conflict or a write-read conflict, i.e., $T_i^k \rightarrow T_x^y$, this rule ensures that $T_j^l \rightarrow T_i^k \rightarrow T_x^y$, i.e., both transactions T_i^k and T_j^l read consistent data values in relation to the transaction T_x^y.

If the delegator transaction T_i^k holds a write lock X_W on X, and there is another transaction T_x^y (at a different mobile host) with which the delegator transaction T_i^k holds a read-write conflict or a write-read conflict, i.e., $T_x^y \rightarrow T_i^k$, this rule ensures that either $T_j^l \rightarrow T_x^y \rightarrow T_i^k$ or $T_x^y \rightarrow T_j^l \rightarrow T_i^k$, i.e., both transactions T_j^l and T_x^y read consistent data values in relation to the transaction T_i^k.

In Figure 6.24, an example of sharing original values with a read lock is shown. Time proceeds from left to right. The delegator transaction T_1^1 at the mobile host MH_1 holds a read lock on the shared data item X and shares the value V_X to the delegatee transaction T_2^1 at the mobile host MH_2. If these two transactions T_1^1 and T_2^1 finally commit when the mobile hosts reconnect to the database servers, the transaction T_2^1 must be scheduled before the transaction T_1^1, i.e., $T_2^1 \rightarrow T_1^1$.

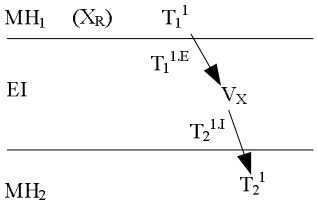

Figure 6.24: Execution constraint of sharing original value with read lock

In Figure 6.25, an example of sharing original values with a write lock is shown. Time proceeds from left to right. The delegator transaction T_1^1 at the mobile host MH_1 holds a write lock on data item Y and shares the original (i.e., non-modified) value V_Y to the delegatee transaction T_2^1 at the mobile host MH_2. In this case, the final transaction schedule will again be $T_2^1 \rightarrow T_1^1$.

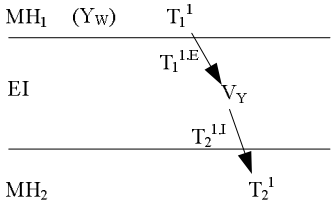

Figure 6.25: Execution constraint of sharing original value with write lock

When a delegator transaction shares an updated data state to a delegatee transaction, the following rule is applied:

> **Rule *4* (*execution constraint of sharing updated data state*)**: *If delegator transaction T_i^k shares an updated data state to delegatee transaction T_j^l, transaction T_j^l must be scheduled after transaction T_i^k and before any transaction T_i^n that is scheduled - due to another update - after transaction T_i^k in the locally committed transaction set LC_i at the same mobile host, i.e.,*
>
> $$\forall T_i^n \in LC_i, T_i^k \in LC_i, T_i^k < T_i^n \Rightarrow T_i^k \rightarrow T_j^l \rightarrow T_i^n$$
>
> *This rule is denoted by $T_i^k \rightarrow\bullet T_j^l$.*

This mobile sharing data states scenario happens when a delegator transaction T_i^k holds a write lock X_W on the shared data item X, and the shared data item has been modified. If the delegatee transaction T_j^l were only to be scheduled after the delegator transaction T_i^k, and the shared data item is later modified again by another transaction T_i^n (the transaction T_i^n is executed at the same mobile host as the delegator transaction and also scheduled

after T_i^k), the execution schedule $T_i^k \rightarrow T_i^n \rightarrow T_j^l$ will not be correct. Instead, the correct execution schedule must be $T_i^k \rightarrow T_j^l \rightarrow T_i^n$, i.e., with the above rule $T_i^k \rightarrow \bullet T_j^l$ being met.

In Figure 6.26, an example of sharing updated values with a write lock is shown (as an extension to the one in Figure 6.25). At some time, the transaction T_1^1 shares the new value $V_{Y'}$ to the delegatee transaction T_2^1. At mobile host MH_1, there is another transaction T_1^2 that later updates it to a new value $V_{Y''}$. The transaction T_1^2 is scheduled after the transaction T_1^1. Rule 4 ensures that the transaction T_2^1 will be scheduled between transactions T_1^1 and T_1^2. This means that the final global transaction schedule is $T_1^1 \rightarrow T_2^1 \rightarrow T_1^2$. When the mobile hosts MH_1 and MH_2 reconnect to the database servers, this transaction execution constraint will be used to support the transaction integration process.

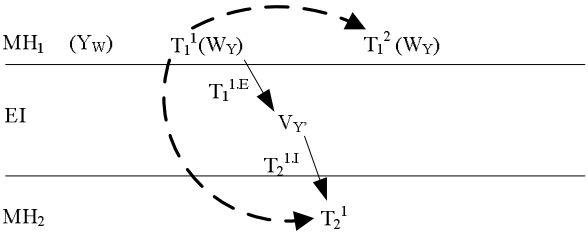

Figure 6.26: Execution constraint of sharing updated value with write lock

Local transactions access cached data with dependency awareness

To recap, for a data item X that is cached in the local workspace at a mobile host, the dependency awareness set X_{DA} keeps track of all the potential dependencies that could occur when a transaction accesses this data item. The dependency awareness set X_{DA} includes abort-dependencies and commit-dependencies.

When a local transaction T_i^k at mobile host MH_i access a data item X, whose dependency awareness set X_{DA} contains an abort-dependency $X(T_j^l,AD)$ and/or a commit-dependency $X(T_j^l,CD)$, it will develop an abort-dependency (T_j^l AD T_i^k) and/or a commit-dependency (T_j^l CD T_i^k) with the transaction T_j^l. Furthermore, the local transaction T_i^k can induce a multiple-abort-dependency with other transactions if it accesses a set of constraint cached data. The dependencies among transactions are created and can be modified via the operations for managing transaction dependencies and execution constraints – addressed in Section 6.2.3.

In Figure 6.27, during the mobile data sharing stage, the delegator transaction T_1^1 at the mobile host MH_1 shares the updated data state $V_{Y'}$ of the data item Y to the delegatee transaction T_2^1 at the mobile host MH_2. There is an abort-dependency $X(T_1^1,AD)$ and a commit-dependency $X(T_1^1,CD)$ related to the shared data item Y (see Case 2 in Section 6.4.1). Later, a local transaction T_2^2 also accesses this shared data item Y. In this case, both the delegatee transaction T_2^1 and the local transaction T_2^2 at the mobile host MH_2 develop abort-dependencies and commit-dependencies with the delegator transaction T_1^1 on the shared data item Y, i.e.,

- (T_1^1 AD T_2^1) via the CreateDependency(T_1^1, T_2^1, AD, static) operation
- (T_1^1 AD T_2^2) via the CreateDependency(T_1^1, T_2^2, AD, static) operation
- (T_1^1 CD T_2^1) via the CreateDependency(T_1^1, T_2^1, CD, static) operation
- (T_1^1 CD T_2^2) via the CreateDependency(T_1^1, T_2^2, CD, static) operation

This means that if transaction T_1^1 aborts, then both transactions T_2^1 and T_2^2 must also abort. Otherwise both transactions T_2^1 and T_2^2 must commit after T_1^1.

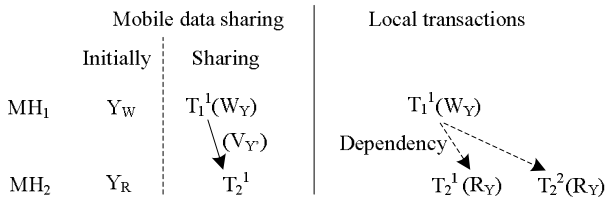

Figure 6.27: Transaction dependencies with constraint cached data

Commit of local transactions that access constraint cached data

When a local transaction T_i^k that operates on a constraint cached data item X commits, the local transaction manager will add this locally committed transaction to the locally committed transaction set LC_i together with its execution constraints and transaction dependencies related to the shared data item X. The log record of the locally committed transaction T_i^k at the mobile host MH_i is as follows:

T_i^k *{(execution_constraint | transaction_dependency)}* where:

- The *execution_constraint* is the execution constraint between the transaction T_i^k and the corresponding transaction T^j that has manipulated data item X.
- The *transaction_dependency* is the transaction dependency between the transaction T_i^k and the corresponding transaction T^j that has manipulated data item X. The transaction dependency can be either an abort-dependency, multiple-abort-dependency or commit-dependency.

When a local transaction T_i^k that operates on constraint cached data item X requests to commit, the following steps are carried out:

(1) The conflict awareness and dependency awareness records associated with the shared data item X are converted to the execution constraints and transaction dependencies, respectively.

(2) The log record of the locally committed transaction T_i^k is added to the locally committed transaction set, i.e.,

$LC_i := LC_i \cup \{ T_i^k$ *{(execution_constraint | transaction_dependency)}}*

In the above example (Figure 6.27), when the local transactions T_2^1 and T_2^2 commit, the following log records are added to LC_2 at the mobile host MH_2:

- For transaction T_2^1: T_2^1{($T_1^1 \rightarrow\bullet T_2^1$), ($T_1^1$ AD T_2^1), (T_1^1 CD T_2^1)}
- For transaction T_2^2: T_2^2{($T_1^1 \rightarrow\bullet T_2^1$), ($T_1^1$ AD T_2^2), (T_1^1 CD T_2^2)}

6.5.4 The aborts of delegator transactions

During the mobile data sharing stage, the interactions between the delegator and delegatee transactions produce dependencies and constraints among these transactions. If there is no abort-dependency between the delegator and delegatee transactions, when the delegator transaction aborts, the delegatee transaction can commit. On the other hand, if there is an abort-dependency between the delegator and delegatee transactions, when the delegator transaction aborts, those delegatee transactions that have read the shared data from this delegator transaction have to abort. In this case, the mobile transaction processing system must keep track of the aborted delegator transactions in order to notify the related delegatee transactions about the abortions.

Figure 6.28 illustrates an abort scenario of the delegator transaction. In the figure, the delegator transaction T_2^1 at the mobile host MH_2 shares a data state $V_{Z'}$ of the data item Z to the delegatee transaction T_1^1 at the mobile host MH_1. At the mobile host MH_1, local transaction T_1^3 also reads this shared data value $V_{Z'}$. Both the transactions T_1^1 and T_1^3 develop abort-dependencies with the delegator transaction T_2^1. If these two mobile hosts are disconnected from each other and the delegator transaction T_2^1 aborts, the transactions T_1^1 and T_1^3 at the mobile host MH_1 will not know about this. Therefore, the mobile transaction processing system must keep track of the abort of the delegator transaction T_2^1 so that the transactions T_1^1 and T_1^3 at the mobile host MH_1 can be notified and aborted at later time.

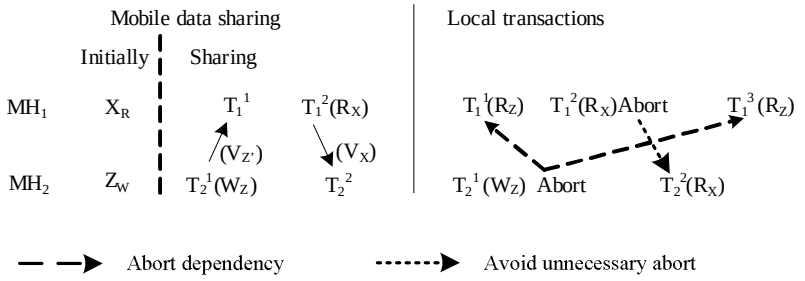

Figure 6.28: Abort of delegator transactions

Also in the Figure 6.28, the delegator transaction T_1^2 at the mobile host MH_1 shares the original data value V_X (which is a consistent with the one in the database server) of the data item X to the delegatee transaction T_2^2 at the mobile host MH_2. At the mobile host MH_2, therefore, there is an execution constraint $T_2^2 \rightarrow T_1^2$ (see Rule 3 in Section 6.5.3) between the delegator and delegatee transactions. There is no abort dependency between these two mobile transactions. This means that if the delegator transaction T_1^2 later aborts, the delegatee transaction T_2^2 can still commit because it has not read an inconsistent data value. The question is: what is the execution schedule position of the delegatee transaction T_2^2 in the global workspace when the delegator transaction T_1^2 aborts?

The transaction manger at the mobile host MH_i will keep a set *LocalAbortedDelegator* (LAD_i) to record the abortions of the delegator transactions. This way, the associated

delegatee transactions will be notified about the abortion of the delegator transaction. Furthermore, in order to support the database servers to find a correct execution schedule for delegatee transactions (which commit even when the corresponding delegator transaction aborts) in the transaction integration stage, the transaction manager at the mobile host will initiate and immediately commit a *pseudo-delegator* transaction T_i^{PD} to the *LocalCommitted* (LC_i) set. This pseudo-delegator transaction T_i^{PD} will mark the position of the actual aborted delegator transaction in the locally committed transaction set LC_i.

When a delegator transaction T_i^k aborts, the following steps will be carried out:

(1) A pseudo-delegator transaction T_i^{PD} is initiated and immediately committed and added to the *LocalCommitted* set in the position of the delegator transaction T_i^k had it committed, i.e.,
$LC_i := LC_i \cup \{T_i^{PD}\}$
(2) The delegator transaction T_i^k is added to the *LocalAbortedDelegator* set, i.e.,
$LAD_i := LAD_i \cup \{T_i^k\}$

Figure 6.29 illustrates how the transaction managers at the mobile hosts handle the abortion of delegator transactions. The delegator transaction T_1^1 at the mobile host MH_1 shares the original data state V_X of data item X to the delegatee transaction T_2^1 at the mobile host MH_2. There is no abort dependency between these two transactions, but there is an execution constraint $T_2^1 \rightarrow T_1^1$ (see Rule 3 in Section 6.5.3). Suppose that if the delegator transaction T_1^1 were committed at the mobile host MH_1, the *LocalCommitted* set LC_i contains: $\{T_1^n < T_1^1 < T_1^m\}$ and hence $T_1^n \rightarrow T_1^1 \rightarrow T_1^m$. When delegator transaction T_1^1 aborts, a pseudo-delegator transaction T_1^{PD} is initiated and committed and inserted in the position of the actual delegator transaction T_1^1, i.e., $\{T_1^n < T_1^{PD} < T_1^m\}$ and hence $T_1^n \rightarrow T_1^{PD} \rightarrow T_1^m$. This way, in the global workspace, the delegatee transaction T_2^1 will be scheduled before the pseudo-delegator transaction T_1^{PD}, i.e., $T_1^n \rightarrow T_2^1 \rightarrow T_1^{PD} \rightarrow T_1^m$.

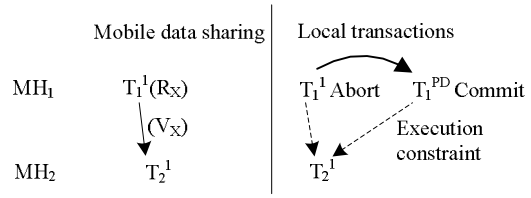

Figure 6.29: The role of the pseudo-delegator transaction

6.6 Transaction integration stage

The transaction integration stage is carried out when the mobile host reconnects to the database servers. In this stage, locally committed transactions, which have been disconnectedly processing at the mobile host, will be validated against other transactions to ensure that the states of the database servers are consistent.

In mobile environments, there is no guarantee that all the mobile hosts will synchronously connect to the database servers to integrate the locally committed transactions at the same time. For example, there is no guarantee that a delegator transaction will be integrated into the database servers before a delegatee transaction or via versa. Furthermore, a local transaction can play roles as both the delegator and delegatee transactions. Consequently, the database servers must keep track of the commit or abort state of both delegator and delegatee transactions in order to determine the effect of one transaction on the others.

Figure 6.30 presents examples of these effects. In Figure 6.30(a), the delegator transaction T_1^i and the delegatee transaction T_2^j, which belong to different mobile hosts MH_1 and MH_2 respectively, develop an abort-dependency (T_1^i *AD* T_2^j) and a commit-dependency (T_1^i *CD* T_2^j). If the delegator transaction T_1^i commits or aborts before the delegatee transaction T_2^j, the final state of the delegatee transaction T_2^j can be determined normally. However, if the delegatee transaction T_2^j requests to finally commit before the delegator transaction T_1^i (as shown in Figure 6.30(b)), the final state of the delegatee transaction T_2^j will not be determined until the state of the delegator transaction T_1^i is known. In this case, the commit of the delegatee transaction T_2^j will be delayed, i.e., resulting in a pending commit.

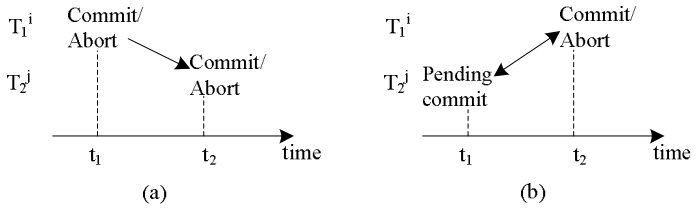

Figure 6.30: The effect of the order of transaction termination requests

Figure 6.31 presents the procedures related to the transaction integration stage. As we have discussed in Section 6.5, at a disconnected mobile host MH_i, the locally committed and locally aborted delegator transactions are kept track of by the transaction manager in two separated set: *LocalCommitted* (LC_i) and *LocalAbortedDelegator* (LAD_i).

For locally aborted delegator transactions in the *LocalAbortedDelegator* (LAD_i) set, these aborted transactions will be transferred to and kept track of in the *GlobalAbortedDelegator* (*GAD*) set at the database servers so that the database servers can inform the associated pending commit delegatee transactions (in the *PendingCommit* (*PC*) set – explained below) about the aborts of delegator transactions.

The locally committed transactions in the *LocalCommitted* (LC_i) set will be validated against other transactions. First, the anchor transaction T_i^A will synchronise its granted lock set L_i^G with the replicated lock set L_i^{GR} at the mobile host. After that, for each of the locally committed transactions in the *LocalCommitted* (LC_i) set, the abort dependencies (that include abort-dependencies and multiple-abort-dependencies) will be verified with the support of the globally aborted delegator transaction *GlobalAbortedDelegator* (*GAD*) set. If the corresponding delegator transactions have not been integrated yet, the locally

committed transactions will be added to the *PendingCommit* (*PC*) set. When the termination states of the corresponding delegator transactions are known, the abort dependencies of the transactions in the *PendingCommit* (*PC*) set will be verified. For those transactions that have passed the transaction dependency check, their execution constraints with other transactions will be checked. If a serializable execution schedule is found, the transactions will be finally committed in the global workspace and added to the *GlobalCommitted* (*GC*) set. But, some of these transactions may be aborted. If an aborted transaction is a delegator transaction, which is locally committed in the local workspace at the mobile host), it will be added to the *GlobalAbortedDelegator* (*GAD*) set.

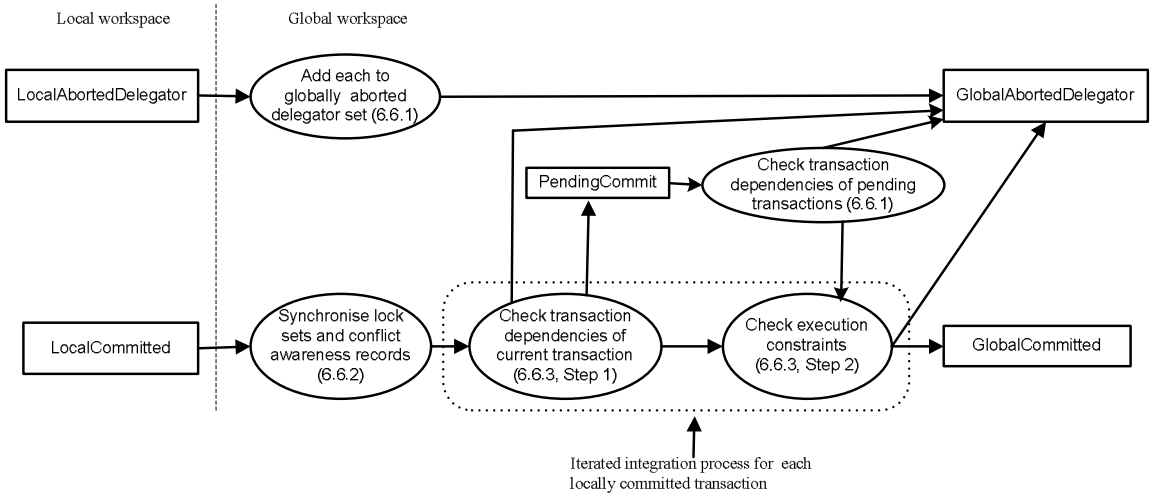

Figure 6.31: Procedures for the transaction integration stage

Section 6.6.1 presents the algorithm that handles the abortion of delegator transactions (i.e., moving transactions from the *LocalAbortedDelegator* (LAD_i) set to the *GlobalAbortedDelegator* (*GAD*) set); and the abort dependencies of transactions (i.e., validating the waiting transactions in the *PendingCommit* (*PC*) set). Section 6.6.2 presents the algorithm that synchronizes the granted lock set L_i^G held by the anchor transaction T_i^A with the replicated lock set L_i^{GR}; and the conflict awareness records. Finally, the checking of transaction dependencies and execution constraints is presented in Section 6.6.3.

6.6.1 Handling the abortion and abort dependencies of transactions

In this section, we present the algorithm that takes care of the final aborts of the locally aborted delegator transactions and verifies the abort dependencies of transactions which are queued in the *PendingCommit* (*PC*) set. The algorithm is illustrated in Figure 6.32 and presented in Figure 6.33.

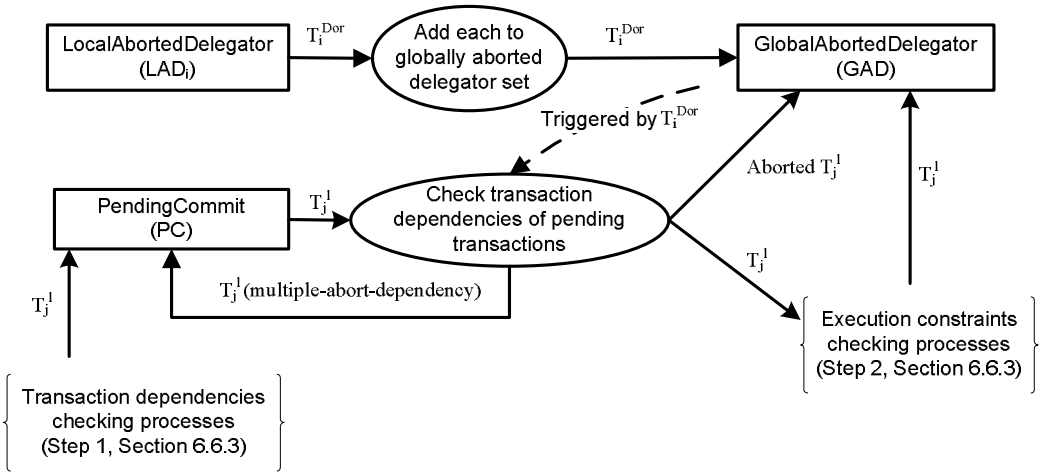

Figure 6.32: Steps of handling the abortion and abort dependencies of transactions

(1) For each T_i^{Dor} in the *LocalAbortedDelegator* set, i.e., $T_i^{Dor} \in LAD_i$
Add T_i^{Dor} to the *GlobalAbortedDelegator* set, i.e., $GAD = GAD \cup \{T_i^{Dor}\}$

(2) For each T_j^l in the *PendingCommit* set that holds an abort-dependency with transaction T_i^{Dor}, i.e., $(T_j^l \in PC) \wedge (T_i^{Dor}\ AD\ T_j^l)$
Abort T_j^l
If T_j^l is a delegator transaction
Add T_j^l to the *GlobalAbortedDelegator* set, i.e., $GAD = GAD \cup \{T_j^l\}$

(3) If T_i^{Dor} belongs to a multiple-abort-dependency with T_j^l in the *PendingCommit* set, *i.e.*, $(T_j^l \in PC) \wedge (T_i^{Dor} \in \Im_i) \wedge (\Im_i\ MA\ T_j^l)$
Mark T_i^{Dor} as an aborted transaction in $\Im_i$
If all transactions in $\Im_l$ have aborted, i.e., $\forall T^l \in \Im_i,\ T^l \in GAD$
Abort T_j^l
If T_j^l is a delegator transaction
Add T_j^l to the *GlobalAbortedDelegator* set, i.e., $GAD = GAD \cup \{T_j^l\}$

Figure 6.33: Handling the abortion and abort dependencies of transactions

The above algorithm is explained as follows:

(1) Each of the locally aborted delegator transaction T_i^{Dor} will be added to the *GAD* set. This will trigger a separate verification of the abort dependencies of the associated transactions.

(2) Any transaction T_j^l in the *PC* set (pending commit transactions are addressed in Section 6.6.3) holding an abort-dependency with the delegator transaction T_i^{Dor} will be aborted. If the aborted transaction T_j^l is a delegator transaction (based on the log of export transactions in the local workspace at the mobile host), the transaction T_j^l will be added to the *GAD* set.

(3) If the aborted delegator transaction T_i^{Dor} belongs to a transaction set $\mathfrak{I}_i$ that holds a multiple-abort-dependency with a pending transaction T_j^l, the transaction T_i^{Dor} in $\mathfrak{I}_i$ will be marked as aborted. If all the transactions in $\mathfrak{I}_i$ are aborted, the transaction T_j^l will abort. Otherwise, the transaction T_j^l remains in the *PC* set. If the aborted transaction T_j^l is a delegator transaction (based on the log of export transactions in the local workspace at the mobile host), the transaction T_j^l will be added to the *GAD* set.

As an example of point (2), in Figure 6.34 the transaction T_2^1 that is pending will be aborted when the corresponding delegator transaction T_1^1 aborts.

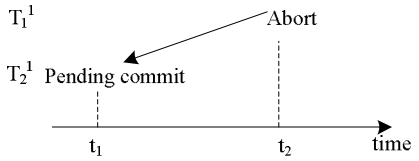

Figure 6.34: Abortion of delegatee transactions

6.6.2 Synchronizing lock sets and conflict awareness records

Before the locally committed transactions at the mobile host MH_i are integrated in the global workspace, the anchor transaction T_i^A synchronizes its locks and the conflict awareness records of the associated cached data items.

The locks in the granted lock set L_i^G held by the anchor transaction must be synchronized with the granted lock set L_i^{GR} that is replicated at the mobile host. Due to the mobile sharing data operations, the L_i^{GR} set may be inconsistent with the L_i^G set. Furthermore, for a cached data item X at the mobile host MH_i, the conflict awareness X_{CA} set may also be modified, therefore, it needs to be synchronized with the one held by the anchor transaction T_i^A.

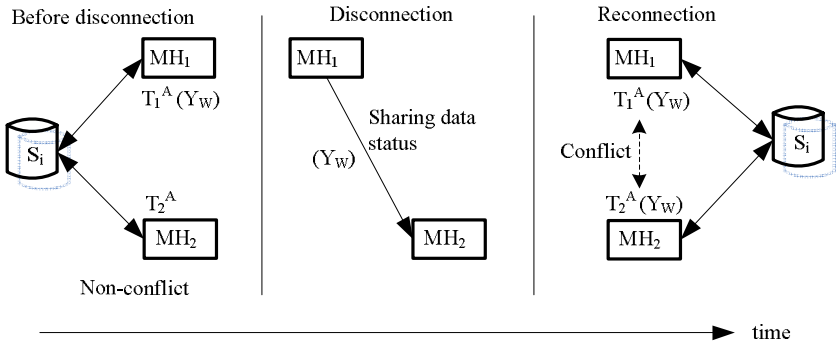

Figure 6.35: Conflicting locks at the anchor transactions

In Figure 6.35, before the disconnection, the anchor transaction T_1^A of the mobile host MH_1 holds a write lock Y_W on data item Y. During the mobile data sharing stage, the write lock Y_W at mobile host MH_1 is delegated to mobile host MH_2. This means that the granted

lock sets L_1^G and L_2^G held by the anchor transactions T_1^A and T_2^A are inconsistent with the lock sets L_1^{GR} and L_2^{GR} at the mobile hosts. This will cause conflicts when the mobile host MH_2 reconnects to the database servers and anchor transaction T_2^A requests an additional write lock on the shared data item *Y*. The database servers cannot grant two write locks on the same data item *Y* to two different mobile hosts (the first write lock was granted to the anchor transaction T_1^A). Furthermore, the conflict awareness sets Y_{CA} can also be inconsistent, and, therefore, must be reconciled.

Before presenting the synchronization done for the anchor transactions, we recap some important results of the previous stages.

At the database servers:

- The anchor transaction T_i^A of mobile host MH_i holds the set of granted locks, i.e., T_i^A holds $L_i^G = L_i^{RG} \cup L_i^{WG} \wedge L_i^{RG} \cap L_i^{WG} = \varnothing$, where:
  L_i^{RG} is the read lock set of the granted read data set D_i^{RG}
  L_i^{WG} is the write lock set of the granted write data set D_i^{WG}
- For each cached data item *X*, there is associated conflict awareness set X_{CA} which records the read-write or write-read conflicts. The conflict awareness records can represent either passive or active conflicts.

At a disconnected mobile host MH_i:

- The granted lock set $L_i^{GR} = L_i^{RGR} \cup L_i^{WGR}$ may be modified due to the mobile sharing data operations, i.e., sharing data states and sharing data status. Therefore, the L_i^{GR} lock set may be inconsistent with the L_i^G lock set held by the anchor transaction T_i^A.
- For each cached data item *X*, the associated conflict awareness set X_{CA} may be modified. Therefore, the conflict awareness records of data item X may be inconsistent with the ones held by the anchor transaction T_i^A.

Based on any differences between the two lock sets $L_i^G = L_i^{RG} \cup L_i^{WG}$ and $L_i^{GR} = L_i^{RGR} \cup L_i^{WGR}$, the anchor transaction will request additional read and/or write locks from the database servers to match the read and/or write locks that are imported by the local transactions at a mobile host during the mobile data sharing stage. The anchor transaction will also release locks that have been delegated during the mobile data sharing stage.

An anchor transaction T_i^A will carry out the following operations:

- Requesting an additional read lock set $L_i^{AR} = L_i^{RGR} \setminus L_i^{RG}$; and an additional write lock set $L_i^{AW} = L_i^{WGR} \setminus L_i^{WG}$.
- Releasing the delegated read lock set $L_i^{DR} = L_i^{RG} \setminus L_i^{RGR}$; and the delegated write lock set $L_i^{DW} = L_i^{WG} \setminus L_i^{WGR}$.

As an example, from the data hoarding stage an anchor transaction T_i^A holds a granted read lock set $L_i^{RG} = \{a_R, b_R\}$ and a granted write lock set $L_i^{WG} = \{c_W, d_W\}$. When a mobile host MH_i is disconnected from the database servers, it imports a read lock e_R on data item *e* and delegates the read lock b_R on data item *b*, i.e., $L_i^{RGR} = \{a_R, e_R\}$. The mobile host MH_i

also imports a write lock f_W on data item f and delegates the write lock d_W on data item d, i.e., $L_i^{WGR} = \{c_W, f_W\}$.

The additional read lock and write lock sets are:

$$L_i^{AR} = L_i^{RGR} \setminus L_i^{RG} = \{a_R, e_R\} \setminus \{a_R, b_R\} = \{e_R\}$$
$$L_i^{AW} = L_i^{WGR} \setminus L_i^{WG} = \{c_W, f_W\} \setminus \{c_W, d_W\} = \{f_W\}$$

The delegated read lock and write lock sets are:

$$L_i^{DR} = L_i^{RG} \setminus L_i^{RGR} = \{a_R, b_R\} \setminus \{a_R, e_R\} = \{b_R\}$$
$$L_i^{DW} = L_i^{WG} \setminus L_i^{WGR} = \{c_W, d_W\} \setminus \{c_W, f_W\} = \{d_W\}$$

The algorithm for synchronization of locks and conflict awareness records held by the anchor transaction T_i^A is presented in Figure 6.36.

(1) For each additionally needed read lock X_R
  Request the read lock X_R with the current caching mode
  If the read lock X_R is granted
    Add X_R to the granted read lock set, i.e., $L_i^{RG} = L_i^{RG} \cup \{X_R\}$
    If there are any new write-read conflicts
      Add these write-read conflicts to the current conflict awareness set X_{CA}

(2) For each additionally needed write lock X_W
  Request the write lock X_W with the current caching mode
  If the write lock X_W is granted
    Add X_W to the granted write lock set, i.e., $L_i^{WG} = L_i^{WG} \cup \{X_W\}$
    If there are any new read-write conflicts
      Add these read-write conflicts to the current conflict awareness set X_{CA}

(3) For each delegated read lock Y_R or write lock Y_W on Y
  Release the lock on Y
  If there is a conflict awareness record $CA_i \in Y_{CA}$
    Notify the corresponding anchor transactions T_j^A about CA_i so that the conflict awareness record CA_i will be disposed of

Figure 6.36: Lock and conflict awareness synchronization

The lock and conflict awareness synchronization algorithm of the anchor transaction T_i^A of the mobile host MH_i is explained as follows:

(1) Additional read locks are the results of (1) importing data values from delegator transactions, i.e., sharing data states; and (2) importing read locks from delegator transactions, i.e., sharing data status. The anchor transaction T_i^A will request the additional read locks from the database servers. If there is any conflict, the conflict awareness records will be used so that the database servers will know about the

delegator transactions that have shared data. For example, a conflict awareness record $Y_W(T_1^1,status)$ indicates that the write lock Y_W on data item Y has been delegated by the delegator transaction T_1^1 at the mobile host MH_1. If there is an anchor transaction T_j^A that holds a conflicting write offline lock, the anchor transaction T_i^A will develop an additional write-read conflict with the anchor transaction T_j^A. A corresponding conflict awareness record is added to the current conflict awareness set Y_{CA} (which is associated with the cached data Y in the local workspace at the mobile host MH_i).

(2) A procedure similar to the one in (1) is carried out for additional write locks on behalf of anchor transaction T_i^A. There may be a write-write locks conflict between two anchor transactions T_i^A and T_j^A (as illustrated in Figure 6.35). In accordance with the conflict awareness records of the cached data item (that includes the identification of the delegator transaction), the database servers will grant the write lock to the anchor transaction T_i^A and send notification to the anchor transaction T_j^A to release its write lock on the shared data item. When the anchor transaction T_j^A receives the release lock message, it will mark the lock as a delegated lock.

(3) For those read and/or write locks that have been delegated to other mobile hosts, the anchor transaction will release those locks. The released locks will make the corresponding data items available to other transactions, i.e., reducing blocking of transactions. If there is any conflict awareness associated with the data items, the anchor transaction T_i^A will notify the corresponding anchor transaction T_j^A about it. The conflict awareness record held by anchor transaction T_j^A will be removed via the method RemoveConflict(*shared_data*,*conflict_transaction*) defined in Section 6.3.4.

After the locks and conflict awareness records held by an anchor transaction have been synchronized, the corresponding locally committed transactions T_i^k in the *LocalCommitted* (LC_i) set will be integrated to the global workspace. From this time on, all the locally committed transactions T_i^k will be considered as online transactions at the database servers.

6.6.3 Checking transaction dependencies and execution constraints

For each transaction in the *LocalCommitted* (LC_i) set of the mobile host MH_i, the integration process includes the following two steps: (1) transaction dependencies are checked; and (2) execution constraints are checked.

The following discussion will address each of these steps in detail.

Step 1: Transaction dependencies of locally committed transactions are checked.

The checking of transaction dependencies is only applied for those transactions that hold abort-dependencies or multiple-abort-dependencies with other transactions. For those transactions that do not hold any abort dependency, this step is not needed in their integration processes. The algorithm for checking the abort dependencies of a locally

committed transaction T_i^k, whose final state depends on the final state of a delegator transaction T_j^{Dor}, is illustrated in Figure 6.37 and presented in Figure 6.38.

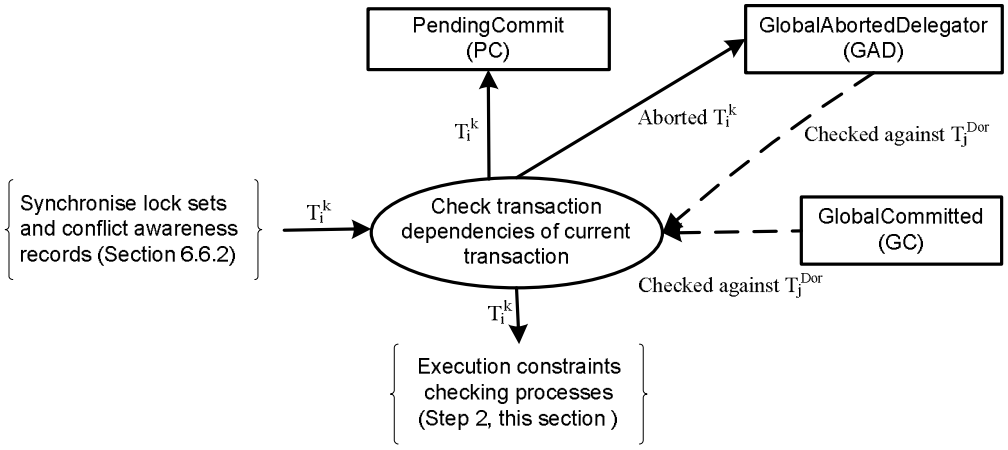

Figure 6.37: Checking trans. dependencies of each locally committed transaction

(1) For each abort-dependency $(T_j^{Dor}\ AD\ T_i^k)$
  If T_j^{Dor} is in the G*lobalAbortedDelegator* set, i.e., $T_j^{Dor} \in GAD$
    Abort T_i^k
    If T_i^k is a delegator transaction
      Add T_i^k to the *GlobalAbortedDelegator* set, i.e., $GAD = GAD \cup \{T_i^k\}$
  Else If T_j^{Dor} is not in the *GlobalCommitted* set, i.e., $T_j^{Dor} \notin GC$
    Add T_i^k to the *PendingCommit* set, i.e., $PC = PC \cup \{T_i^k\}$

(2) For each multiple-abort-dependency $(\mathfrak{I}_i\ MD\ T_i^k)$
  If T_j^{Dor} is in the $\mathfrak{I}_l$ set and T_j^{Dor} is in the *GlobalAbortedDelegator* set,
  i.e., $(T_j^{Dor} \in \mathfrak{I}_l) \wedge (T_j^{Dor} \in GAD)$
    Mark T_j^{Dor} as an aborted transaction in $\mathfrak{I}_i$
    If all transactions in $\mathfrak{I}_i$ have aborted, i.e., $\forall T^m \in \mathfrak{I}_i\ ,\ T^m \in GAD$
      Abort T_i^k
      If T_i^k is a delegator transaction
        Add T_i^k to the *GlobalAbortedDelegator* set, i.e., $GAD = GAD \cup \{T_i^k\}$
  Else If T_j^{Dor} is in the $\mathfrak{I}_i$ set and T_j^{Dor} is not in the *GlobalCommitted* set,
  i.e., $(T_j^{Dor} \in \mathfrak{I}_i) \wedge (T_j^{Dor} \notin GC)$
    Add T_i^k to the *PendingCommit* set, i.e., $PC = PC \cup \{T_i^k\}$

Figure 6.38: Verifying transaction dependencies of a locally committed transaction

The details of the algorithm to verify the transaction dependencies - where the two parts are mutually exclusive, is explained as follows:

(1) For each abort-dependency between the locally committed transaction T_i^k and a delegator transaction T_j^{Dor}, if the delegator transaction T_j^{Dor} has aborted, the transaction T_i^k must abort too. Otherwise, if the delegator transaction T_j^{Dor} has not reached the transaction integration stage, the locally committed transaction T_i^k will be added to the *PC* set. In this case, the abort-dependency will be re-evaluated when the termination state of the delegator transaction T_j^{Dor} is known (see point (2) in Figure 6.33).

(2) For each multiple-abort-dependency, and for each corresponding delegator transaction T_j^{Dor}, if the delegator transaction T_j^{Dor} has aborted, mark T_j^{Dor} as an aborted transaction. If all the corresponding delegator transactions have aborted, the transaction T_i^k aborts too. Otherwise, if a delegator transaction T_j^{Dor} has not reached the transaction integration stage yet, the locally committed transaction T_i^k will be added to the *PC* set. In this case, the multiple-abort-dependency will be re-evaluated when the termination state of the delegator transaction T_j^{Dor} is known (see point (3) in Figure 6.33).

Step 2: Execution constraints of locally committed transactions are checked.

Those locally committed transactions that have passed the transaction dependencies check (i.e., step 1) will enter the final commit process. During this process, the execution constraints among transactions will be evaluated. To recap, a locally committed transaction that operates on non-constraint cached data will be allowed to finally commit at the database servers. However, this transaction must synchronize itself with transactions with which it conflicts passively. On the other hand, a local transaction that operates on constraint cached data, will be validated against other transactions based on the execution constraints (see Section 6.5.3). If finally committing a locally committed transaction causes a non-serializable schedule, the transaction will be aborted. The algorithm for finally committing a locally committed transaction T_i^k that only accesses non-constraint cached data (in the local workspace at the mobile host MH_i) is presented in Figure 6.39.

This final commit process of a locally committed transaction T_i^k that only accesses non-constraint cached data is explained as follows:

(1) If there are passive conflicts - which is the only option in this case - related to a standard transaction T_j^l, which is carried out at the mobile host MH_j and has committed in the global workspace, the execution constraints between transactions T_i^k and T_j^l will be determined based on Rules 1 and 2 in Section 6.5.3 and evaluated. If transactions T_i^k and T_j^l end up being non-serializable, a notification will be sent to transaction manager so that it can be handled separately, e.g., by compensating T_j^l which must be a transaction accessing constraint cached data. After this, transaction T_i^k commits and is added to the *GC* set.

(2) All anchor transactions T_j^A that conflict passively - once more the only option in this case - with T_i^k will be notified about the commit of transaction T_i^k. Each such anchor transaction T_j^A will update its conflict awareness record related to the shared data so that the local transactions T_j^l at mobile host MH_j will know about the conflict with T_i^k when the mobile host MH_j reconnects to the database servers. This is done via the method ModifyConflict (*shared_data, anchor_transaction*, *new_conflict_transaction*) defined in Section 6.3.4.

```
(1)   If there are passive conflicts associated with transaction T_i^k
          For all standard transactions T_j^l that conflict passively with T_i^k
              Check the execution constraints
              If T_j^l and T_i^k are non-serializable
                  Notify the transaction manager for manual handling
      Commit T_i^k
      Add T_i^k to the GlobalCommitted set

(2)   For all anchor transactions T_j^A that conflict passively with T_i^k
          Notify T_j^A about the commit of T_i^k
```

Figure 6.39: Committing transactions accessing non-constraint cached data

The algorithm for finally committing a locally committed transaction T_i^k that accesses constraint cached data (in the local workspace at the mobile host MH_i) is presented in Figure 6.40.

This final commit process of a locally committed transaction T_i^k that accesses constraint cached data - where we may have both active and passive conflicts, is explained as follows:

(1) This concerns the active conflicts - of which there must be at least one. If the checking ends up with a non-serializable result, one of the transactions T_j^l and T_i^k must be aborted. If T_j^l is alive, we have to make a choice between it and T_i^k - which one depends on the policy to be used in a specific system. But if T_j^l has commited, we have no choice but to select T_i^k. Finally, if T_j^l has aborted, the non-serializability check will have ended void. If the aborted transaction T^m is a delegator transaction, it will be added to the *GAD* set so that related pending transactions T^p in the *PC* set may be re-evaluated.

(2) This concerns the situations where there also are passive conflicts - which is not a necessity. Hence the same algorithm as in Figure 6.39 is carried out - except that in this case anchor transactions could conflict both actively and passively with T_i^k.

When all the locally committed transactions in the *LocalCommitted* (LC_i) set have been integrated at the database servers, the anchor transaction T_i^A of the mobile host MH_i will release all the remaining locks and will then commit.

(1) For all transactions T_j^l that represent active conflicts with transaction T_i^k
Check the execution constraints
If T_j^l and T_i^k are non-serializable
If T_j^l is Alive
Choose $T^m = T_j^l \mid T_i^k$
Else
Set $T^m = T_i^k$
Abort T^m
If T^m is a delegator transaction
Add T^m to the *GlobalAbortedDelegator* set
If $T^m = T_i^k$
Exit

(2) If there are passive conflicts associated with transaction T_i^k
For all standard transactions T_j^l that conflict passively with T_i^k
Check the execution constraints
If T_j^l and T_i^k are non-serializable
Notify the transaction manager for manual handling
Commit T_i^k
Add T_i^k to the *GlobalCommitted* set

For all anchor transactions T_j^A that conflict actively or passively with T_i^k
Notify T_j^A about the commit of T_i^k

Figure 6.40: Committing transactions accessing constraint cached data

6.7 Managing dynamic transaction structure and transaction mobility

In this section, we discuss advanced transaction operations that support: (1) dynamic restructuring of transactions, (2) mobility of transactions.

6.7.1 Supporting dynamic restructuring of transactions

The standard transactions will initiate shared transactions when there is a need of mobile data sharing. As discussed in Section 6.4.2, the mobile transaction processing system provides two different methods to generate shared transactions: (1) as a merged transaction, and (2) as a sub-transaction. These two methods are discussed below:

- `MergeImportTrans(`T^{Dee}`,` T^{I}`).` This operation is applied for a flat delegatee transaction. The operation allows a delegatee transaction T^{Dee} to initiate a new import transaction T^I that will be merged into the delegatee transaction when the import transaction has obtained the needed data items.

- `SubImportTrans(`T^{Dee}`,` T^{I}`).` This operation is applied for a nested delegatee transaction. The operation allows a delegatee transaction T^{Dee} to initiate a new import transaction T^I that will be adopted as a sub-transaction of the delegatee transaction

when the import transaction has obtained the needed data items. For example, when a parent delegatee transaction wants to import shared data, it will initiate a new sub-shared transaction that imports shared data for the parent transaction.

6.7.2 Supporting mobility of transactions

The execution of mobile transactions at a mobile host depends on the mobility behavior of the mobile host (see Section 3.5). The mobile host can move to different mobile cells or be involved in many mobile affiliation workgroups during its operation. Therefore the standard transactions will also move from one mobile sharing workspace to another. In Section 5.7.3, we have discussed how the anchor transaction and the shared transactions can support the mobility of the standard transactions as the mobile host moves. To recap, the anchor transaction can support the mobility of transactions across mobile cells, while the shared transactions support the mobility of transactions across mobile sharing workspaces.

The following methods are provided to handle the mobility of transactions:

- `MoveAnchorTrans(`MSS_i`, `MSS_j`)` moves the anchor transaction T_i^A of the mobile host MH_i from the old mobile support station MSS_i to the new mobile support station MSS_j. This means that the mobile host MH_i currently stays in the mobile cell managed by the mobile support station MSS_j and connects to the mobile support station MSS_j. This movement of the anchor transaction is initiated by the mobile host.

- `SplitSharedTrans(`$T_i^{k.S1}$`, `$T_i^{k.S2}$`)` splits the current shared transaction $T_i^{k.S1}$ (which can be either an export or import transaction) of a standard transaction T_i^k into two sub-shared transactions $T_i^{k.S1}$ and $T_i^{k.S2}$. This happens when the mobile host moves from one mobile affiliation workgroup to another. The first sub-shared transaction $T_i^{k.S1}$ can continue in the old mobile sharing workspace while the second sub-shared transaction $T_i^{k.S2}$ will operate in the new mobile sharing workspace.

- `JoinSharedTrans(`$T_i^{k.S1}$`, `$T_i^{k.S2}$`)` joins the shared transaction $T_i^{k.S1}$ with the shared transaction $T_i^{k.S2}$. This happens when the mobile host moves back to a previous mobile affiliation workgroup, i.e., the standard transaction joins the previous mobile sharing workspace. Then the previous split-shared transaction $T_i^{k.S1}$ that is executing in the old mobile sharing workspace, is joined with the on-going sub-shared transaction $T_i^{k.S2}$.

6.8 Conclusions

In this chapter, we have formalized our mobile transaction processing system. The execution of mobile transactions can be divided into four stages: the data hoarding, the mobile data sharing, the disconnected transaction processing, and the transaction integration. In the data hoarding stage, the mobile transaction processing system supports two different conflict modes for dealing with offline transactions: *read-write conflict* and *write-read conflict*. The conflicts among transactions at different mobile hosts are

handled with the support of anchor transactions that play roles as proxy transactions for local transactions at the mobile hosts.

When the mobile hosts are disconnected from the database servers, local transactions at mobile hosts are carried out based on the cached data in the local workspaces. At the same time, the transactions at different mobile hosts can share their cached data with the support of export and import transactions through the export-import repository. This mobile data sharing allows mobile transactions to share data in an asynchronous manner and without any support from the database servers. Therefore, the mobile data sharing increases data availability in mobile environments. When the mobile host reconnects to the database servers, the transaction integration processes are performed. In this stage, the data that has been manipulated during disconnected periods is integrated to ensure global data consistency.

PART III

IMPLEMENTATION and EVALUATION

Chapter 7
Implementation of the Mobile Transaction Processing System

In this chapter, we discuss the abstract architecture of the MOWAHS mobile transaction processing system. Based on this abstract architecture, we have developed the MOWAHS prototype architecture that acts as a proof of concept for our theoretical research. We have chosen two important system components of the MOWAHS prototype architecture, the mobile locking system and the mobile data sharing system, for prototype designing and implementation.

7.1 Introduction

In part two of this thesis, we have presented and formalized the mobile transaction processing system that focuses on supporting mobile data sharing among mobile transactions at different mobile hosts. In this chapter, we shift our focus from theoretical research to empirical work. We will discuss how the mobile transaction processing system is designed, implemented and deployed as a real mobile transaction processing system.

The main strategy of our practical work is that system components of the MOWAHS mobile transaction processing system must be designed as added components. This means that system components of the MOWAHS mobile transaction processing system can be built and deployed besides the existing transaction processing or database systems. To achieve this, we first design an abstract architecture for the MOWAHS mobile transaction processing system. Based on this abstract architecture, we have then developed a prototype architecture that acts as a proof of concept for our theoretical research. Due to the constraints of time and resources of the MOWAHS project, the current MOWAHS mobile transaction processing system is not completely implemented. However, we have successfully designed, implemented and tested two important system components of the mobile transaction processing system: (1) the mobile locking model, which minimizes blocking of mobile transaction processes in mobile environments; and (2) the mobile sharing data system, which supports data sharing among transactions at different mobile hosts.

The organization of this chapter is as follows. Section 7.2 describes the overall abstract architecture of the MOWAHS mobile transaction processing system. Based on this abstract architecture, the MOWAHS prototype architecture is presented in Section 7.3. The design and implementation of the mobile locking system and the mobile data sharing system are presented in Section 7.4 and 7.5 respectively. Section 7.6 summaries the development of the MOWAHS mobile transaction processing system.

7.2 Abstract architecture of the MOWAHS system

This section will discuss the abstract architecture of the MOWAHS mobile transaction processing system. An overview of the MOWAHS system is presented in Figure 7.1.

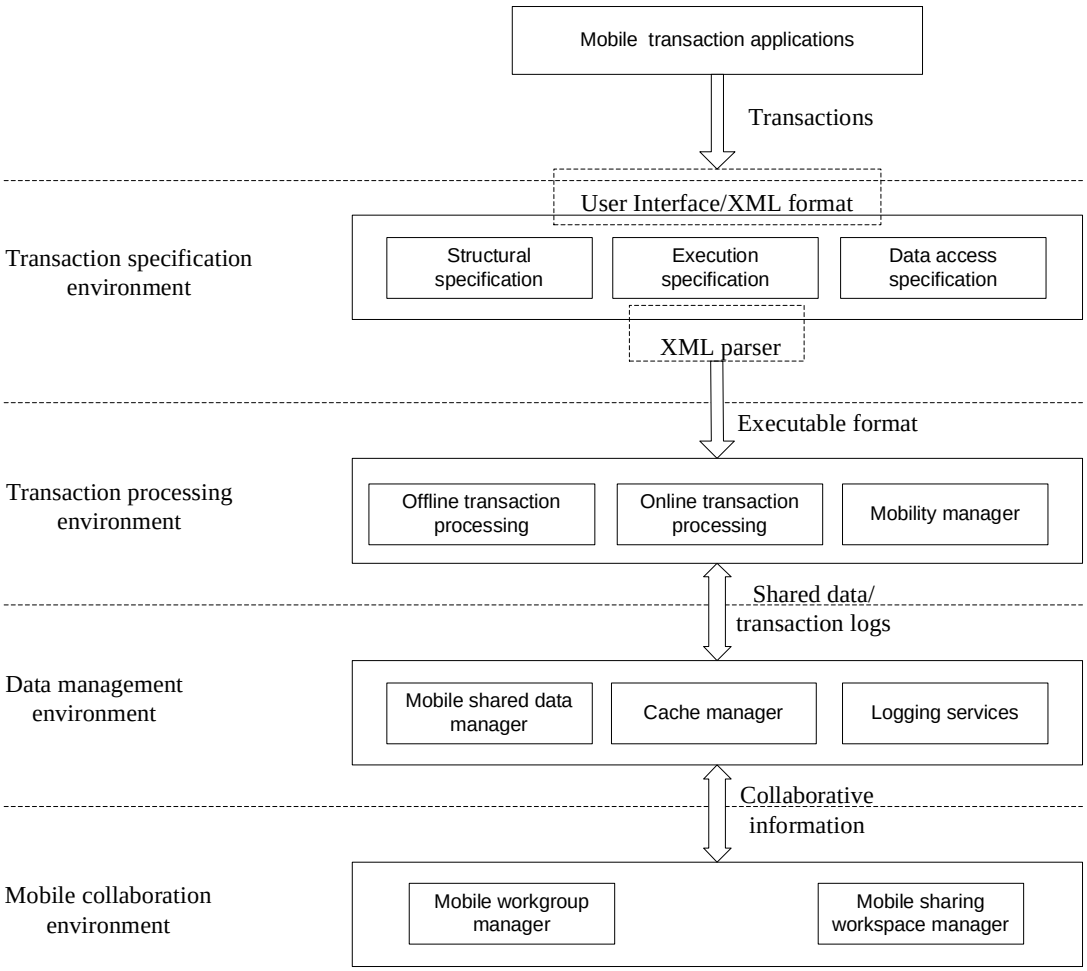

Figure 7.1: MOWAHS system architecture

The MOWAHS system architecture consists of four different layers: the *transaction specification environment*, the *transaction processing environment*, the *data management environment*, and the *mobile collaboration environment*. These four layers realize all the system components of our theoretical research results. For example, the mobile

collaboration environment realizes the mobile affiliation workgroups and the mobile sharing workspaces, while the data management environment enforces the data consistency in local and global workspaces.

The following sections describe the features and functionalities of each of the environment layers.

7.2.1 Transaction specification environment

The transaction specification environment provides an interface for the client applications to submit transactions in mobile environments. The specification information of a mobile transaction is described in an XML document [HM04] and includes the structure, execution and data access characteristics of submitted transactions.

Structural specification. The structural specification provides an interface to describe the structure of transactions. The structure of a transaction specifies (1) if the transaction is a flat or nested transaction, (2) the type of the transaction, i.e., delegator, delegatee, export or import transaction. If a transaction has a nested structure, the type of each sub-transaction must be specified. For example, a submitted transaction T_i^k has a nested structure that includes two sub-transactions $T_i^{k.1}$ and $T_i^{k.2}$, where $T_i^{k.1}$ is a delegator transaction while $T_i^{k.2}$ is a delegatee transaction.

Execution specification. The execution specification provides an interface to describe the execution characteristics of a transaction, i.e., how the transaction is to be carried out. A transaction can be carried out as either an online transaction or an offline transaction (to recap, the *online transactions* are transactions that are executed at the fixed database servers, and the *offline transactions* are those transactions that are carried out and managed by the mobile transaction managers at disconnected mobile hosts). If a transaction is executed as an offline transaction, an anchor transaction will additionally be specified. An execution specification also describes the dependencies among transactions, i.e., abort-dependencies, multiple-abort-dependencies or commit-dependencies. For example, a client application from the mobile host MH_i submits a delegatee transaction T_i^k that will be carried out as an offline transaction and holds an abort-dependency (T_j^l *AD* T_i^k) with delegator transaction T_j^l.

Data access specification. The data access specification provides an interface to describe what shared data will be accessed by a submitted transaction. The accessed data set is exclusively either read-only or updating. Based on the data access specification, the cache manager (in the data management environment - see Section 7.2.3) will try to obtain the needed shared data from the database server (during the data hoarding stage) or from other mobile hosts (through the mobile data sharing stage).

The transaction specification (i.e., in an XML document) will be parsed through an XML parser into executable representations, for example SQL queries, before being transferred to the transaction processing environment.

7.2.2 Transaction processing environment

The transaction processing environment provides the facilities that carry out the execution of the submitted transactions in accordance with the transaction specification.

Offline transaction processing. The responsibility of the offline transaction component includes two parts. First, the offline transaction processing administrates the execution of offline transactions in a local workspace at a mobile host while the mobile host is disconnected from the database server. The transaction manager at the disconnected mobile host will make use of the two phase locking protocol (2PL) to ensure data consistency in the local workspace, i.e., by a serializable execution schedule of local transactions. Second, the offline transaction processing controls the execution of shared transactions, i.e., export and import transactions, which carry out the mobile data sharing among standard transactions through an export-import repository.

Online transaction processing. The online transaction processing component handles the execution of online transactions that include both normal database transactions and anchor transactions. The online transaction processing must control the potential conflicts among transactions due to conflicting cache modes (that are read-write and write-read). The online transaction processing component also supports the integration of local transactions, i.e., when the locally committed transactions at mobile hosts are integrated into the database server.

Mobility manager. The mobility manager provides the facilities to control the movement of transactions in accordance with the movement of mobile hosts. This means that the mobility manager must handle not only the movement of anchor transactions, but also the re-structuring of shared transactions.

7.2.3 Data management environment

The data management environment provides the facilities to support: (1) the management of mobile shared data in a mobile sharing workspace; (2) the cache manager for supporting the data hoarding stage, and (3) the logging service for mobile transactions.

Mobile shared data manager. The mobile shared data manager administrates shared data in the mobile sharing workspaces. While being disconnected from the database servers, the mobile data sharing mechanism supports transactions at the mobile hosts to share data through the mobile sharing workspace (i.e., the export-import repository). Therefore, the mobile shared data manager must provide all the functionalities related to the shared data items that are currently being stored in the mobile sharing workspace (see Table 5.10).

Cache manager. When a mobile host is carrying out data hoarding operations (to support disconnected transaction processing), the data management environment must ensure that cached data in the local workspace is fully consistent. If there is any conflict due to the conflicting cache modes (i.e., read-write conflict and write-read conflict), the cache manager must ensure that the involved transactions are fully aware of that. Moreover, the

cache manger must also manage shared data in the local workspace which can be modified due to the mobile data sharing among standard transactions (i.e., during the mobile data sharing stage – see Section 6.4)

Logging services. The data management environment must also provide a logging service to support the mobile transaction processing system to record the asynchronous interaction and integration of mobile transactions. For example, records of shared data and shared transactions must be kept in order to support the transaction integration stage. The mobile transaction processing system must also be supported to keep track of the abortion and commitment of delegator and delegatee transactions.

7.2.4 Mobile collaboration environment

The mobile collaboration environment provides the facilities that support the management of the mobile affiliation workgroups and mobile sharing workspaces.

Mobile workgroup manager. The mobile workgroup manager provides necessary services that support a mobile host to create, join or leave a mobile affiliation workgroup. The mobile host can create a new mobile affiliation workgroup, and in this case, the mobile workgroup manager must ensure that the identification of the new mobile workgroup does not conflict with other existing mobile workgroups. When a mobile host joins a new workgroup or leaves the current workgroup, the mobile workgroup manager ensures that the collaborative activities of the mobile workgroup continue normally, i.e., without any disruption. The mobile workgroup manager also provides communication functionalities so that each member of the mobile workgroup can notify other members about its membership status. For example, a mobile host may announce to other members the approximate time that it intends to be with the mobile affiliation workgroup.

Mobile sharing workspace manager. The mobile sharing workspace manager provides a directory service to support management of the mobile sharing workspace. The directory service will handle all the management operations related to the physical distribution of the mobile sharing workspace (see Table 5.9), for example to create a new mobile sharing workspace or manage the capacity of the mobile sharing workspace.

7.3 Architecture of the MOWAHS prototype

The MOWAHS prototype architecture consists of two main parts: (1) the mobile transaction support system that is designed for operating at the mobile host, and (2) the non-mobile transaction support system that is designed for supporting transaction processing at the fixed hosts. Figure 7.2 presents the system components of a mobile host and a fixed host.

At a fixed host, the Global transaction manager (Global TM) is responsible for managing the submitted online and offline transactions from the mobile hosts. The lock requests from these online and offline transactions are handled with the support of the Global lock

manager. The Global log manager provides a service to handle the abortion and commitment of the local transactions in the global workspace.

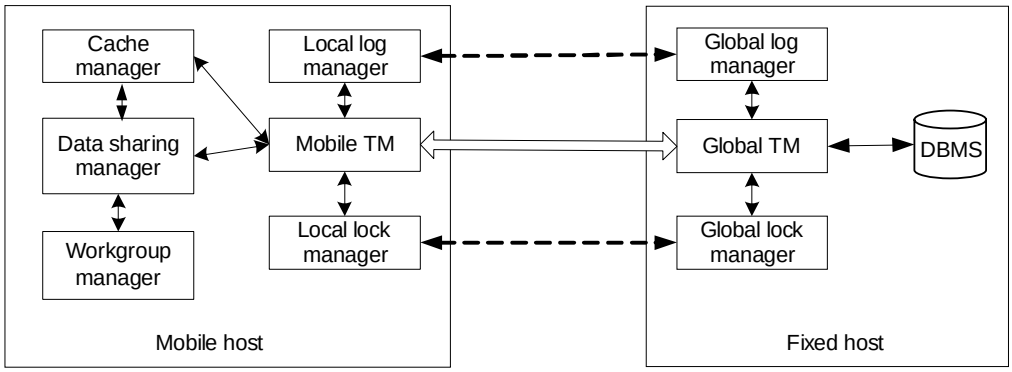

Figure 7.2: Architecture of the MOWAHS prototype

At a mobile host, the Mobile transaction manager (Mobile TM) takes responsibility for managing the local transactions at the mobile host. The Local lock manger at the mobile host manages the local lock requests of local transactions. When a local transaction is locally committed, the Local log manger provides a logging service to ensure that the committed results will not be lost. These commit log records will be used to support the transaction integration processes.

The Cache manager (with the support of the Local lock manager) at the mobile host manages the shared data that is obtained during the data hoarding and mobile data sharing stages. The Workgroup manager and Data sharing manager have responsibility for supporting the mobile data sharing between transactions at different mobile hosts.

Compared to the abstract architecture (see Figure 7.1), the Global and Mobile transaction managers provide interfaces for client applications to specify and submit transactions, i.e., corresponding to the transaction specification environment. The Global transaction manager (with the support of the Global lock and log managers) also takes responsibility to support online transaction processing and transaction mobility; and the Mobile transaction manager (with the support of the Local lock and log managers) supports offline transaction processing. The Local and Global log managers, together with the Cache and Data sharing managers, constitute the data management environments. Finally, the Workgroup manager controls to the features of the mobile collaborative environment.

Due to the constraint of time and resources, the MOWAHS prototype architecture is not fully implemented. Anyway, there are several related sub-system prototypes that have been developed and may be co-deployed with our MOWAHS system prototype. For example, mobile workgroup management in mobile environments has been designed and implemented in several related research works [BCM05, Liu+05].

We have successfully designed, implemented and tested two important components of the MOWAHS mobile transaction processing system. The two selected components are: the mobile locking system and the mobile data sharing system (see Figure 7.3). In the following Sections 7.4 and 7.5, we describe our design and implementation of these two components.

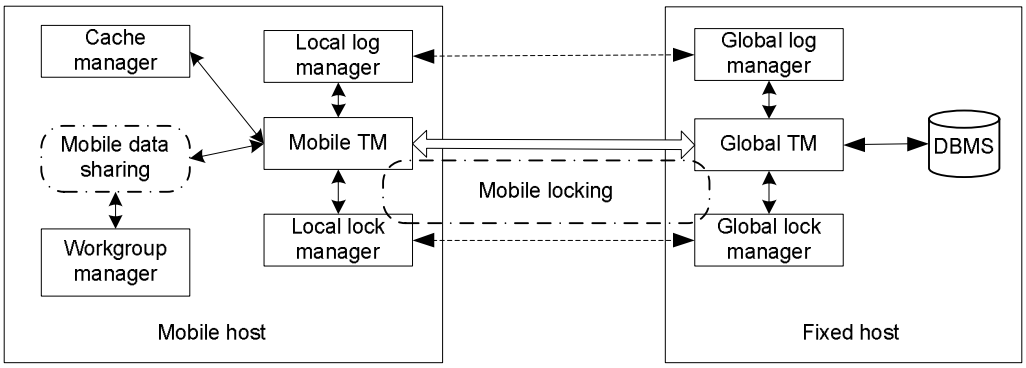

Figure 7.3: The system components selected for implementation

7.4 The mobile locking system

In this section, we describe the design and implementation of the mobile locking model that supports the mobile transaction processing system to cope with disconnections and support online and offline transactions.

7.4.1 The design of the mobile locking system

The mobile locking system consists of two parts: the lock modes and the lock sharing.

Lock modes

One of the challenging issues with mobile databases is that a shared data item could be locked at a disconnected mobile host for long periods. In addition, the execution of mobile transactions can vary due to the constraints of mobile resources, for example inducing longer processing time. This could also delay the execution of other transactions. To deal with this problem, we introduce two different types of lock: offline locks and online locks. Offline locks include read offline and write offline locks that support offline transactions. Online locks are standard read and write locks and are used for online transactions.

The compatibility matrix of all the locks is presented in Table 7.1. In the table, the lock on data item X is denoted $X_{lock\text{-}mode}$, i.e., the four locking modes are X_{roff}, X_{woff}, X_{ron} and X_{won}. A "*Y*" in the table indicates that locks are compatible, i.e., the new lock request can be granted. Otherwise the new lock request is rejected, i.e., "N".

Note that the mobile lock matrix is an asymmetric table due to the fact that a write online lock is not compatible with any other locks. In other words, if an online transaction holds

a write online lock on a shared data item, no other transaction will be allowed to access this shared data item. However, a write online lock request on a shared data item is allowed even when there is an offline transaction that holds a read offline lock on the shared data item. This does not lead to any inconsistency problem because the offline transaction is reading a consistent data.

Table 7.1: Lock matrix of mobile databases

		Transaction T^i holds lock			
		X_{roff}	X_{ron}	X_{woff}	X_{won}
Transaction T^j requests lock	X_{roff}	**Y**	**Y**	**Y**	N
	X_{ron}	**Y**	**Y**	**Y**	N
	X_{woff}	**Y**	**Y**	N	N
	X_{won}	**Y**	N	N	N

Read locks (i.e., read online and read offline locks) are always compatible to each other. A read offline or online lock request on a shared data item can be granted when there is a write offline lock on the same data item. This means that many transactions can request a read lock on a shared data item which is being modified by an offline transaction at a disconnected mobile host. This way, the system throughput may be increased in case a shared data item is write offline locked at a mobile host for a long disconnected period. On the other hand, a write offline lock request on a data item can also be granted even when there are read online and offline locks on the data item. This can be done because the value of data item is not immediately updated at the database servers.

The database servers will keep two lock logs called active and pending lock logs. The active lock log keeps track of the current active online lock on data items. The pending lock log stores the current locks on data items whose values are not be modified immediately at the database servers, i.e., with write offline locks. To support both synchronous and asynchronous database operations, the locking model will uphold the following four rules.

- **Rule 1**: If both X_{ron} and X_{roff} exist, then the X_{ron} is an active lock while the X_{roff} is a pending lock. This means that any write online lock requests on the shared data item X will be rejected. When the online read operation is completed, the X_{roff} lock is changed to the active lock.

- **Rule 2**: If an X_{ron} exists and mobile host MH_j requests an X_{woff}, the X_{woff} is granted as a pending lock. The X_{ron} lock remains active. When the online read operation is completed, the X_{woff} lock is moved to the active lock log at the database servers.

  The reasons for using rule 2 are two fold. First, the value of the shared data is not updated immediately at the database servers. Therefore, on-going operations that read data item X should be allowed to continue executing. Furthermore, offline transactions that read the shared data item X can be scheduled before the updating transaction [HAA02]. Second, an updating transaction is first performed offline in the

local workspace of a disconnected mobile host, and data will remain consistent if no other transaction is allowed to modify the data item.

- **Rule 3**: If an X_{woff} exists and mobile host MH_j requests an X_{ron} or an X_{roff} lock, the X_{ron} is granted as an active lock, while the X_{roff} is granted as a pending lock. The un-modified data value of X is returned for the read operation. If the X_{ron} is granted, then the X_{woff} lock is changed to a pending lock. When the read operation is completed, the X_{woff} lock is changed back to an active lock.

  Rule 3 allows other read operations to be executed immediately. On-going transactions that read the shared data item after the write offline lock will be scheduled before the updating transaction. Moreover, disconnection periods are normally unpredictable and could be long lasting; therefore this rule benefits read only transactions.

- **Rule 4**: If an X_{roff} is an existing active lock and mobile host MH_j requests an X_{won} or an X_{woff}, the X_{won} or X_{woff} lock is granted as an active lock. The X_{roff} lock is changed to a pending lock.

  Rule 4 allows an updating transaction to be carried out immediately. On-going offline transactions that read the shared data item will be scheduled before the updating transaction. The database server will provide a logging service to record the modifications on shared data to ensure that the offline transactions will be notified about such changes when the mobile host reconnects.

The mobile locking model is able to cope with unplanned disconnections. Note that locking modes at mobile hosts and database servers might be different. For example, an offline transaction at a disconnected mobile host can hold a read offline lock on a shared data item, while at the database servers the lock applied on this shared data item can be either an active write offline lock or an active read online lock or a write online lock.

Lock sharing

In this section, we describe the lock sharing operations that allow a transaction to share locks with other transactions. There are three types of lock sharing operations: upgrade, downgrade and delegate. Figure 8.9 illustrates the relationships among locks.

An upgrade lock request is either a take-over or a self-upgrade lock. This can happen when a mobile host changes its network status from disconnected to connected. When a mobile host holds a write offline lock on data item X and its network connectivity state changes from disconnected to connected, a write offline lock will be converted to the normal write online lock on item X. All other transaction that read the data item X might be forced to abort [LNR04]. When a mobile host reconnects to the database server, a read offline lock on item X can be converted to a read online lock if there is not any online transaction that holds a write online lock on item X. If there is an online transaction that is modifying this item X, the conversion will be delayed. If a transaction holds a read offline lock on a shared data item and a write offline lock on the same data item is delegated by a

(delegator) transaction, the transaction can obtain the write offline lock to upgrade its accessing level, i.e., from read offline to write offline.

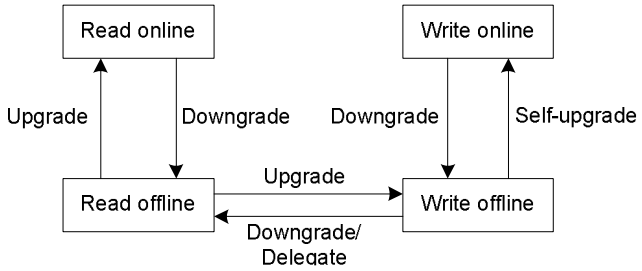

Figure 7.4: Lock sharing operations

A mobile host can carry out a downgrade operation to decrease the level of a lock on a data item. This can happen when a mobile host changes its network status from connected to disconnected. When mobile host MH_i disconnects from the database server as planned or due to a sudden disconnection, all read online locks held by mobile host MH_j are downgraded to read offline locks and all write online locks are converted to write offline locks. The write offline lock ensures that offline transactions at the disconnected mobile host retain the right to update the data item. Furthermore, downgrading online to offline locks avoids the problem of long lasting locks due to disconnections by allowing other transactions to gain access to shared data, for example read online or write offline on shared data.

Furthermore, when a transaction holds a write offline lock on data item X and an update operation is not carried out as planned after all, the transaction can either downgrade or delegate the write offline lock on item X to another transaction. This gives other transactions a chance to carry out their updating operations on the shared data item.

7.4.2 The implementation of the mobile locking system

In this section, we address the implementation of the mobile locking system. The mobile locking prototype has been implemented in the Java programming language. The prototype architecture is presented in Figure 7.5.

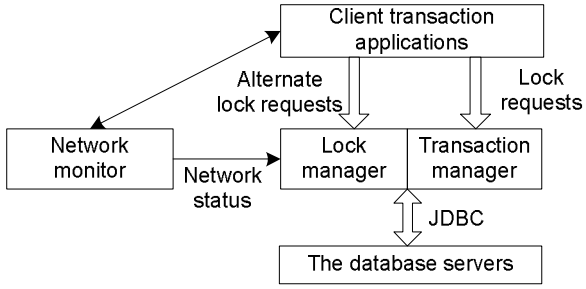

Figure 7.5: Mobile locking prototype architecture

The system components of the mobile locking system are described below:

- **Client transaction applications.** Each client transaction is implemented as a thread *in the system*. The client transaction (expressed as an SQL query) can have either an online or offline status, which means that the transaction will be carried out at the database servers or the disconnected mobile host respectively. The connectivity state of a mobile host can dynamically change during the execution of a transaction. Consequently, the state of locks held by each transaction will change in accordance with the connectivity state of its mobile host.

- **Network monitor.** For each mobile host, there is a network monitor thread that monitors the connectivity of the mobile host. When the network connectivity of the mobile host changes, the network monitor will notify the lock and transaction mangers so that the states of the corresponding locks at this mobile host will be changed.

- **Transaction manager.** The transaction manager creates transaction threads on demand from client transaction applications. It manages the mapping between a client and its corresponding transactions. This mapping is essential because the network monitor only keeps track of the network connectivity of a mobile host, not individual transactions. The transaction manager manages all the events related to the execution of the submitted transactions. Furthermore, the transaction manager provides a method for establishing JDBC-connections and transferring the SQL-queries to the database servers.

- **Lock manager.** The lock manger controls the lock requests from the client transaction applications in accordance with the characteristics of the submitted transactions, i.e., whether online or offline. The lock manager keeps a lock table which contains mappings between the locks on shared data and the transactions holding these locks. Before a lock request is granted, the lock manager checks if there is any conflicting lock and sets the state of the granted lock as active or pending. The lock manager also cooperates with the network monitor for managing the lock changes of the submitted transactions (upgrades, downgrades and delegates).

- **The database servers.** We use a MySQL database [SM05, Dye05] which has many built in features that are already implemented, like the online lock modes and the possibility to switch off the auto-commit functionality. In our implementation, the MySQL locking model is used without the auto-commit functionality.

7.5 The mobile data sharing system

In this section, we describe the design and implementation of the mobile data sharing system that supports the mobile transactions at different mobile hosts to share data while being disconnected from the database servers. The main objective is to increase data availability in mobile environments. The mobile data sharing system has been designed and implemented as a master thesis [HB05].

7.5.1 The design and implementation of the mobile data sharing system

The implementation architecture of the mobile data sharing system is presented in Figure 7.6.

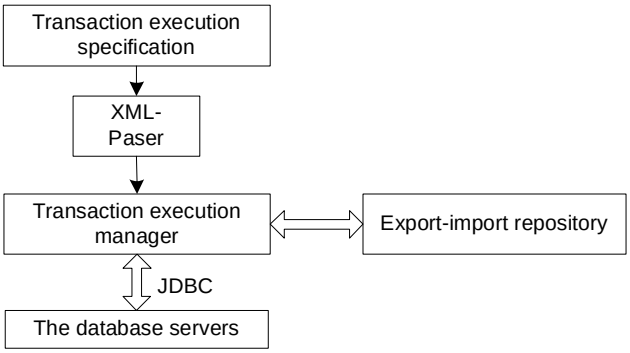

Figure 7.6: Mobile data sharing prototype architecture

The prototype of the mobile data sharing system only focuses on sharing data states among transactions. All the components of the mobile data sharing system are described below:

- **Transaction execution specification**. The specification of a submitted transaction in the mobile data sharing system is described by in an XML document. The standard transactions will have a nested structure, and the shared transactions are initiated and executed as sub-transactions of these standard nested transactions. Therefore, there are three types of transactions in the mobile sharing system: the mobile transaction, the sub-transactions, and the sub-shared transactions. The mobile transaction plays role as a standard transaction (i.e., delegator or delegatee transaction), the sub-transactions are the normal sub-transactions in a nested transaction, and the sub-shared transactions are the shared transactions. The mobile transaction will have the total control to all of the sub-shared transactions. Consequently, the commitment of the export and import transactions is carried out within the local workspace and under control of the standard transaction. Note that this design is not contrasting our mobile transaction processing system as presented in Part 2. The shared data is still stored at the mobile sharing workspace. An export transaction can commit in the local workspace, however, its results are durable only after the delegator transaction has committed. When an export transaction is partially committed within the scope of the nested delegator transaction, it will notify the corresponding import transactions. The export-import repository manager will allow the import transaction to read the shared data item in the mobile sharing workspace.

- **XML-parser.** The specification of a submitted transaction is converted into an internal SQL query representation via an XML parser. We have decided to make use

of the existing XML parser Xerces2 Java Parser[6] to support the transformation of transaction specifications.

- **Transaction execution manager**. The transaction execution manager takes an SQL query as input. When an SQL query is received, the transaction execution manger will submit this to be executed in the database servers (described below) via the standard JDBC connection. If a shared transaction is received, the transaction execution manager will carry out the execution of the share transactions via the write() or read() method of the Java Transaction API.

- **Export-import repository.** The mobile sharing workspace is designed and implemented with the Jini and JavaSpace technology [Jini, FHA99]. The mobile sharing workspace is created by the transaction execution manager when a shared transaction is initiated by the standard transaction. The mobile sharing workspace is allocated at one computer due to the limitation of the JavaSpace technology. We will further discuss the issue related to the mobile sharing workspace in Section 7.5.2.

- **The database servers.** As mentioned before, we have used a MySQL database which has many built in features that are already implemented. In this implementation, the MySQL locking model is used with the standard commit functionality. This does not contrast with switching off the auto-commit functionality in the mobile locking system. In the mobile locking system component, the transaction manager manages both offline and online transactions; therefore, it is possible to integrate both the mobile data sharing system and the mobile locking system.

The performance of the mobile transaction processing system with the support of the mobile data sharing system has also been partially tested. The preliminary test results [RG05], without taking into account the disconnections of mobile hosts from the mobile affiliation workgroups, have shown a significant improvement in system throughput.

7.5.2 The physical distribution of mobile sharing workspaces

The Jini and JavaSpaces technology is used to construct export-import repositories in which export and import transactions interact with each other. In relation to the design and implementation of an export-import repository, there are several engineering challenges.

The first issue concerns the allocation of the mobile sharing workspace. In our mobile transaction processing system, the export-import repository is a truly distributed mobile sharing workspace, i.e., the mobile sharing workspace is distributed over and allocated on several mobile computing hosts (see Figure 5.7(d)). However, the JavaSpaces technology is not designed to fully support the physically distribution of a mobile sharing workspace. The JavaSpaces technology only supports one physical location for a mobile sharing space, i.e., the mobile sharing workspace is entirely located at and bound to a mobile

[6] http://xml.apache.org/xerces2-j/

computing host that provides data sharing services (see Figure 5.7(a, b and c)). Therefore, JavaSpaces can not fully support the design and implementation of our export-import repository. Moreover, a single physical location can also cause bottleneck problems in terms of accessing shared data and single points of failure. Our current solution is to consider a group of several individual sharing workspaces, which are located at several different mobile hosts, as one single mobile sharing workspace. Thus, in a mobile affiliation workgroup, there is a group of mobile sharing workspaces where each of which belongs to one individual mobile host. However, it is not necessary that every mobile host in the mobile affiliation workgroup must possess a mobile sharing workspace.

The second issue concerns the naming service. Service discovery is one of the most important features of the Jini and JavaSpaces technology, and it is relying on the support of a naming service. A mobile host will use the discovery service to detect the existing mobile affiliation workgroup and mobile sharing workspace. The operation of the discovery service requires the support of a naming service that manages the deployment of the mobile affiliation workgroup and the export-import repository. The naming service includes persistent and transient naming services[7] (a persistent naming service provides a permanent naming context of computing hosts, while a transient naming service only maintains a naming context of computing hosts while it is in active) and is normally deployed at a non-mobile server. If mobile hosts are disconnected from the non-mobile naming service provider, it is not possible to apply the discovery service to discover the mobile affiliation workgroup and mobile sharing workspace. Therefore, in our mobile transaction processing system, a naming service must also be deployed for each mobile affiliation workgroup.

The above approach can also be applied to support management of the mobile sharing workspace (that includes management of the physical distribution of the export-import repository and data management in the export-import repository - see Section 5.6). For example, a new mobile sharing workspace can be added to the existing group of mobile sharing workspaces when a mobile host joins the existing mobile workgroup, and a shared data item can be copied from one mobile sharing workspace to another. However, there are several disadvantages with this approach. First, a mobile host has to create and manage its own mobile data sharing workspace; therefore, more mobile resources are needed. Second, there is a need for an additional management layer that manages the organization of the individual mobile sharing workspaces and the naming service of the mobile affiliation workgroups. This may cause extra overhead, for example with setting up or accessing the export-import repository, with mobile data sharing operations among transactions at different mobile hosts.

7.6 Summary

In this chapter, we have presented the abstract and prototype architectures of our MOWAHS mobile transaction processing system. We have successfully designed and

[7] http://java.sun.com/j2se/1.4.2/docs/guide/idl/jidlNaming.html

implemented two essential system components: the mobile locking system and the mobile data sharing system. All the designed functionalities of these two system components have been successfully tested. Because of the constraint of time and resources, the other system components of the MOWAHS prototype architecture have not been implemented yet.

Chapter 8
Discussion and Evaluation

The objective of this chapter is to discuss and to evaluate our research results. First, we discuss how our mobile transaction processing system takes into account the challenging characteristics of mobile environments. We compare our research results with related works. Second, we evaluate how our research results (1) fulfill the requirements of a mobile transaction processing system, and (2) answer the main research question.

8.1 Discussion

In this section, we first answer the question: How are the mobile environments characteristics taken into consideration in our mobile transaction processing system? We compare our research contributions with related research works. And, we discuss challenging issues in relation to the design and implementation of the export-import repository.

8.1.1 Dealing with the challenging characteristics of mobile environments

To recap, the three main characteristics of the mobile environments are: the mobility of mobile hosts, the limitation of wireless networks, and the resource constraints of mobile devices (see Section 3.2). Our mobile transaction processing system is appropriate for mobile environments because it takes into consideration all three characteristics of mobile environments. The following discussion addresses how our mobile transaction system takes care of these characteristics:

- **The mobility of mobile hosts.** The general architecture of the mobile transaction environment requires that: in a mobile cell, in order to either contact other hosts or access shared data a mobile host must connect to the mobile support station. This way, the movement of a mobile host can be managed via the identifications of mobile support stations which the mobile host has connected to. However, if the mobile host is not able to connect to the mobile support station, it has no other means to cooperate with other hosts; and the movement of this mobile host may not be manageable. In our mobile transaction processing system, the mobility of mobile hosts in mobile environments is taken into account via the concepts of the mobile affiliation workgroup. The mobile affiliation workgroup takes advantage of the ability of

wireless communication technologies to support collaborative work among mobile hosts. The mobile host can join either an affiliation workgroup if it can connect to a fixed host or a mobile affiliation workgroup if it can link up with nearby mobile hosts. Via the identifications of mobile affiliation workgroups, the movement of mobile hosts which are not connecting to a mobile support station, can be managed. Thus, through the concepts of non-mobile and mobile affiliation workgroups, the mobility of mobile hosts in mobile environments is taken fully into consideration.

- **The limitations of wireless networks.** The limitations of wireless networks, for example low bandwidth, short connection periods and frequent disconnections, affect data availability in mobile environments and curtail collaborative work among mobile hosts. To cope with the problems, our mobile transaction processing system provides a flexible mechanism to support data sharing among mobile hosts. Data sharing processes are separated from the main transaction processes via the support of shared transactions. The data sharing processes can be divided into a set of smaller and recoverable export and import transaction processes. Furthermore, the mobile data sharing mechanism also takes advantage of close range wireless communication technologies, for example Bluetooth or wireless USB, so that mobile hosts can utilize their networking capacity. This way, a mobile host, which is not able to connect to database servers via a wireless LAN, can obtain shared data from other nearby mobile hosts, i.e., data availability is enhanced. Finally, the export and import transactions can deal with the disconnection problems by supporting mobile transactions to share data in an asynchronous manner.

- **The resource constraints of computing devices.** The resource constraints of mobile devices, for example limited storage capacity or slow processing speed, have a strong impact on the performance of transaction processing systems. To deal with the problems, our mobile transaction processing system provides a dynamic and reconfigurable mobile sharing workspace, called the export-import repository. The export-import repository is physically distributed among mobile hosts (which belong to a mobile affiliation workgroup), and plays the role of an additional workspace through which mobile hosts can support each other. Transaction processes can share or save results in the export-import repository; therefore, the problems of limited storage capacity or failures of mobile hosts can be dealt with. Furthermore, the shared transactions also support sharing data status among transactions at different mobile hosts, i.e., transfer control of shared data from one transaction to another. This means that the mobile transaction processing system can cope with the limited processing capacity of mobile hosts by distributing transaction processes among mobile and stationary hosts.

8.1.2 Comparison with related works

In this section, we compare our research results with other related works. To recap, the main objective of our mobile transaction processing system is to support mobile collaborative work by enhancing the level of data availability in mobile environments in which mobile hosts usually are disconnected from the database servers. We achieve this

objective by (1) allowing disconnected mobile hosts to form temporary and dynamic mobile affiliation workgroups to support their collaborative work, and (2) providing a mobile data sharing mechanism that supports sharing of data among transactions at different mobile hosts. The mobile affiliation workgroups are formed based on short range and peer-to-peer communication technologies. A mobile host, which is disconnected from the database servers, can establish a communication channel with nearby mobile hosts and join mobile affiliation workgroups. This way, collaborative activities among mobile hosts can be carried out without any support from the database servers. The mobile data sharing among transactions at different mobile hosts is carried out through export-import repositories with the support of export and import transactions. Two types of mobile data sharing are supported by the mobile transaction processing system: sharing data states and sharing data status. Moreover, our mobile transaction processing system has the ability to support the mobility of transactions (when a mobile host moves from one place to another) and to improve data conflict awareness in mobile environments.

The comparison is divided into five topics - that are: the *organization of a mobile workgroup*, the *mobile sharing workspace*, the *mobile data sharing mechanism*, the *data consistency and conflict awareness*, and the *transaction mobility* (see Table 8.1).

Table 8.1: The MOWAHS transaction processing system features

Comparison issues	**Related research**	**Our advantages**
Organization of a mobile workgroup	Mobile workgroup management [Liu+05, BCM05].	Mobile affiliation workgroup, supporting collaborative work in horizontal dimension
Mobile sharing workspace	Check-in/Check-out model [HAA02, Ram01], LIME [PMR00]	Dynamic, reconfigurable, and distributed export-import repository
Mobile data sharing mechanism	Client-server architecture [BF03], Delegation [Chr93, Ram01], Inter-processes [PRM00]	Peer-to-peer data sharing via shared transaction, supporting sharing data state and data status
Data consistency and conflict awareness	Compacts in Pro-motion [WC99], Pseudo-transaction [HAA02]	Anchor transactions supporting conflict awareness, supporting conflicting cached modes
Transaction mobility	Kangaroo transaction [DHB97], Pre-write [MB01]	Two types of transaction mobility: across mobile cells and across mobile affiliation workgroups

The advantages of our mobile transaction processing system are as follows:

- *Organization of a mobile workgroup*. Mobile workgroup management in mobile environments is an active research field [BCM05, Liu+05]. The objective is to support mobile users to share resources in a dynamically changing environment that is affected by the physical locations of mobile hosts and the variations of network connectivity. According to our knowledge, the concept of a mobile affiliation workgroup is one of the first attempts to extend the existing collaborative workgroup models to support mobile collaborative works in mobile environments, especially in the horizontal dimension. Currently, there are other related approaches [BCM05, Liu+05] that have been proposed to support the management of dynamic workgroups in mobile environments.

- *Mobile sharing workspace.* The private-common workspace model has been widely applied to support cooperative and collaborative work in distributed environments [HAA02, Ram01, PMR00]. However, this model is not adequate in mobile environments due to, for example the static organization of the common workspaces, and the pre-defined and hierarchical data access paths. Our mobile sharing workspace, i.e., the export-import repository, is a dynamic and reconfigurable sharing workspace that focuses on supporting peer-to-peer mobile data sharing. Furthermore, the export-import repository is a distributed sharing workspace that has capacity to deal with the dynamic organization of the mobile affiliation workgroups and the variations of mobile resources. Via the export-import repository, transactions at different mobile hosts can directly share data without support from the database servers.

- *Mobile data sharing mechanism.* Resource sharing in mobile environments plays a vital role to enhance the performance of mobile work. Existing approaches that support data sharing such as delegation operations [Chr93, Ram01] or inter-process interactions [PRM00] do not have the capacity to support mobile data sharing in mobile environments. These approaches lack the ability to deal with the disconnections of wireless networks. The AMDB mechanism [BF03] is a client-server architecture that supports mobile data sharing among mobile hosts. The limitation of that architecture is that the role of a mobile host is constrained to either the database server or a database client. Our mobile transaction processing system supports the mobile data sharing among transactions at different mobile hosts by (1) separating the data sharing process from the main transaction, and (2) using transactions to support the data sharing process. The shared transactions (i.e., the export and import transactions) are neither under control by the original standard transactions nor the database servers. In other words, the shared transactions can continue carrying out the mobile data sharing operations even if and when the original standard transactions fail. The shared transactions also have the ability to cope with unstable wireless networks by splitting a shared transaction into sub-shared transactions or joining sub-shared transactions into one shared transaction. Furthermore, the mobile data sharing mechanism can support both sharing data state and data status.

- *Data consistency and conflict awareness.* The common approach to support data consistency in mobile environments is through reconciliation processes [HAA02, WC99]. The main disadvantage of that approach is that local transactions at the mobile hosts are not aware of conflicting database operations. This can result in extended transaction aborts. Our mobile transaction processing system supports three different data caching modes – that are non-conflict, read-write conflict and write-read conflict - that minimize the delay of transactions due to conflicts. Potential conflicting operations of transactions are alert via anchor transactions that act as proxy transactions to local transactions at disconnected mobile hosts. When the mobile hosts reconnect to database servers, the anchor transaction will support the integration of local transactions. The main advantages of the anchor transactions are: (1) enhancing conflict awareness among transactions at different mobile hosts, and (2) supporting temporary data and transaction management in mobile environments by keeping track of accessed data sets and termination states of mobile transactions.

- *Transaction mobility.* Existing transaction models can support transaction mobility in the connected mode [DHB97, MB01]. The hand-over or hand-off processes are carried out every time the mobile host enters a new mobile cell. Those approaches can not be applied if there is disconnection in communication during the movement of the mobile host. Our mobile transaction processing system can support the mobility of transactions in two different ways: (1) anchor transactions support handling the mobility of transactions when mobile hosts move across mobile cells, and (2) shared transactions support controlling the mobility of transactions when mobile hosts move across mobile affiliation workgroups. Hand-over processes, which handle the movement of anchor transactions, are initiated by a mobile host when it is connecting to database servers or mobile support stations, i.e., hand-over processes are carried out only when they are needed. According to our knowledge, there is no similar research that has taken the mobility of transactions across mobile affiliation workgroups into account.

8.2 Evaluation

In this section, we evaluate how our research results fulfill the requirements that are presented in Section 3.5, and answer the research questions.

8.2.1 Fulfilling the requirements

Our research results fulfill the designated requirements of the mobile transaction processing system. The fulfillment of each requirement is elaborated as follows:

R1. *The mobile transaction processing system must be able to effectively handle the hand-over control of mobile transactions*.

In our mobile transaction processing system, there are two types of transaction mobility in accordance with the movement of a mobile host: (1) the mobile host is moving across

mobile cells, and (2) the mobile host is moving across mobile affiliation workgroups. The mobility of transactions across mobile cells is supported by the movement of the anchor transaction that is the proxy transaction of these transactions. This way, our mobile transaction processing system handles hand-over processes efficiently, i.e., the hand-over processes are initiated by the mobile host. As long as mobile transactions can be entirely carried out in the local workspace of the mobile host, i.e., the execution environment of the mobile transactions is not changed, it is not necessary to perform hand-over processes. The mobility of transactions across mobile sharing workspaces (i.e., when the mobile host is moving across mobile affiliation workgroups) is handled by re-structuring, i.e., splitting or joining, the export and import transactions.

R2. *The mobile transaction processing system must support interactions among transactions at different mobile hosts.*

Execution processes of mobile transactions can be distributed among mobile hosts of a mobile affiliation workgroup without support from mobile support stations or any non-mobile hosts, by means of mobile sharing workspaces and shared transactions. The mobile data sharing mechanism supports both sharing data states and data status among standard transactions at different mobile hosts. This way, the mobile transaction processing system solves the problem with transactions on a mobile host heavily relying on mobile support stations to carry out interaction operations with other transactions at a different mobile host. As long as the mobile hosts belong to a mobile affiliation workgroup, standard transactions can interact with each other via the export-import repository. Furthermore, export and import transactions ensure that the sharing of data among standard transactions is carried out in a recoverable manner, i.e., the mobile transaction processing system has the ability to deal with data inconsistency and execution schedule problems that may occur when a delegator transaction fails.

R3. *The mobile transaction processing system must support disconnected transaction processing*.

Disconnected transaction processing at mobile hosts is supported via the data hoarding and mobile data sharing stages. In the data hoarding stage, consistent data stored at database servers is downloaded into the mobile hosts with the support of anchor transactions (with three different data caching modes: non-conflict, read-write conflict and write-read conflict). Needed data that is not available during the data hoarding stage can be obtained during the mobile data sharing stage with the support of shared transactions. Local transactions at disconnected mobile hosts are processed based on cached data that is either fully consistent or constrained (with the ones in different workspaces – see Section 6.5.4). Local transactions are allowed to commit locally at the mobile hosts and the results of local transactions are made accessible to other local transactions. The locally committed transactions will be validated in the transaction integration stage to finally commit at the database servers when the mobile hosts reconnect to them.

R4. *The mobile transaction processing system must support distributed transaction execution among mobile hosts and stationary hosts*.

The affiliation workgroup concept provides the means to allow mobile hosts to join non-mobile and mobile hosts in a workgroup. The distributed execution of transactions among mobile and non-mobile hosts is carried out via export and import transactions in an affiliation workgroup.

R5. *The mobile transaction processing system must have the ability to customise the atomicity property of transactions.*

The mobile transaction processing system customizes the atomicity property of standard transactions via the support of shared transactions. The atomicity property of delegator transactions can be relaxed by means of export transactions. Export transactions support long-lived transactions by allowing transactions to save their partial results in mobile sharing workspaces. By supporting mobile transactions to save their partial results while they are being executed, the model prevents losing useful work done by mobile transactions upon failure of standard transactions. Import transactions support delegatee transactions to obtain needed data from the mobile sharing workspaces. If the delegatee transaction aborts, the results of the import transaction can still be useful to other local transactions.

R6. *The mobile transaction processing system must support sharing partial states and status among transactions*.

To avoid long blocking of transactions in mobile environments due to data unavailability, mobile data sharing among transactions at different mobile hosts is supported by means of shared transactions through export-import repositories. Mobile transactions can share their partial results with others by making data accessible in a mobile sharing workspace. The mobile data sharing mechanism supports both sharing data states and data status. Export and import transactions ensure that data sharing processes among mobile transaction will be atomically executed.

R7. *The mobile transaction processing system must assure the durability property of transactions*.

Committing mobile transactions are done in two ways: (1) local commit at the mobile hosts, and (2) final commit at the database servers. The results of locally committed transactions are durable only in the local workspace when the mobile host is disconnected from the database servers. If the local committed transactions have accessed cached data that is consistent in the local workspace, these transactions will be allowed to finally commit at the database servers. The full durability of transactions is achieved after the mobile transactions are finally committed at the database servers.

R8. *The mobile transaction processing system must provide efficient recovery strategies.*

The mobile transaction processing system provides two different transaction recovery strategies via (1) the static and dynamic transaction dependencies and (2) the multiple-abort dependencies. By these dependencies, the relationship among mobile transactions may be flexibly defined or modified so that when a transaction aborts, the execution of the related transactions can be adjusted to assure global data consistency.

R9. *The mobile transaction processing system must support temporary data and transaction management.*

The execution processes of mobile transactions are carried out at different computing hosts that can be either connected or disconnected. So, the temporary state of data and transactions must be managed so that local transactions at a disconnected mobile host will be aware of what shared data has been modified and what transactions have committed or aborted. This is achieved by the support of anchor transactions. An anchor transaction keeps track of the data cached at the mobile host and supports conflict awareness for local transactions at disconnected mobile hosts. The mobile data sharing processes among standard transactions at different disconnected mobile hosts are also kept track of to determine the relationship among these transactions.

8.2.2 Answering the research questions

In this section, we will discuss how the main research questions of this thesis have been answered.

As stated in Chapter 1, the main research question of this thesis is:

> ***How can we furnish a transaction processing system so that it can cope with the constraints of mobile resources and the variations of operating conditions in mobile environments?***

The research question has been answered by the development of our MOWAHS mobile transaction processing system that includes: a thorough study of the characteristics of mobile transactions, a set of requirements that mobile transaction processing systems must have, a research approach based on a mobile collaborative work scenario, the development of a mobile data sharing mechanism, and the design and implementation of the system prototypes. The mobile transaction processing system has been equipped with a mobile data sharing mechanism that supports sharing of data among transactions at mobile hosts that are disconnected from the database servers. This mechanism increases data availability in mobile environments.

To explain in detail our approach, we will answer the four refined questions that have directed the development of this work:

Q1: Current situation.

- What are the current ideas and concepts that have been developed to answer the main research question or to address part of it?

Chapter 4 has surveyed and discussed the related research on mobile transaction models and mobile transaction processing systems. From this review, we have identified the main limitations of these mobile transaction models and processing systems. Each mobile transaction model tries to answer part of the research question, like to support mobility or support disconnected transaction processing at mobile hosts. However, a complete solution has not been achieved yet.

Q2: Characteristics and requirements of mobile transactions.

- What are the challenging characteristics of transactions in mobile environments?
- What are the requirements of a mobile transaction processing system that accomplishes the main research question?

In Chapter 3, we have addressed the challenging characteristics of mobile environments in detail and studied how these characteristics of mobile environments impact the behavior of mobile hosts. We have analyzed the characteristics of transactions in mobile environments. Based on these characteristics, we have proposed a set of requirements that a mobile transaction processing system must have for it to cope with the constraints of mobile resources and the variable operating conditions.

Q3: Approach and solutions.

- What are the concepts and foundations for developing the required mobile transaction processing system?
- How should we design and implement the required mobile transaction processing system?

Our approach is based on a mobile IT-support scenario. From this scenario, we have proposed a new collaborative work model for mobile environments, i.e., the horizontal collaboration. Using this as a starting point, we have developed an adaptive mobile data sharing mechanism that distinguishes two types of mobile data sharing: sharing data states and sharing data status. This mobile data sharing mechanism not only enhances data availability in mobile environments but also takes into account all the challenging characteristics of mobile environments. We have also chosen to design and implement two important components of our mobile transaction processing system: the locking model and the mobile sharing workspace. The mobile locking system supports mobile transactions to cope with disconnections and long locking periods. The mobile data sharing system supports data sharing among transactions at different disconnected mobile hosts.

Q4: Evaluation.

- How well do the research results fulfill the requirements of the mobile transaction processing system?
- How do the research results compare with previous related works?

This chapter (Chapter 8) has discussed how our research results fulfill the designated requirements of a mobile transaction processing system, and answered the main research questions. Furthermore, important parts of the thesis have been published at international conferences and workshops [Sør+02, Ram+03, LNR04, LN05a, LN05b, Sør+05]. This allows our research results to be discussed and compared with related research in the field.

8.2.3 Limitations

We have designed and implemented two important components of our mobile transaction processing system, which are the mobile sharing workspace with the export and import transactions, and the locking protocols for sharing mobile data. However, due to the constraints of time and resources, not all the features of our mobile transaction processing system have been fully implemented or tested.

Chapter 9
Conclusion and Future Work

This chapter summaries our research achievements and addresses several possible extensions in future research.

9.1 Research achievements

The main research achievements of this thesis are:

- *A new model and concepts to support mobile collaborative work.* We have extended the common hierarchical collaborative work model in the horizontal dimension to support collaborative work in mobile environments. The horizontal collaborative work model takes advantage of new mobile technologies, for example mobile computing devices and wireless networks, to promote and support mobile collaborative work. This new working model allows mobile users to dynamically form temporary mobile affiliation workgroups while being on the move and disconnected from the database servers. The mobile affiliation workgroups are formed on demand, and can be dynamically configured in accordance with the behavior of mobile hosts or users. By the support of mobile affiliation workgroups, mobile hosts can interact and support each other to increase the performance of mobile works.

- *New concepts and models for mobile transaction processing.* Our mobile transaction processing model supports both online, i.e., connected mobile hosts, and offline, i.e., disconnected mobile hosts, transaction processing. The model allows both online and offline transactions to be concurrently carried out and be aware of conflicts via the support of anchor transactions (to recap, the anchor transactions play roles as proxy transactions for local transactions at mobile hosts). The anchor transactions and the shared transactions (i.e., export and import transactions) support the mobile transaction processing system to handle the mobility of mobile transactions as the mobile hosts move. We have also proposed a new multiple-abort-dependency rule that allows the mobile transaction processing system to flexibly define the correlation among transactions.

- *Concepts and models for sharing data among transactions at different mobile hosts in mobile environments.* The mobile data sharing model provides a flexible mechanism

for transactions at disconnected mobile hosts to share data with others, i.e., enhance data availability and reduce blocking time of transactions. The sharing information processes are divided into a set of smaller recoverable export or import transaction processes. This will help mobile hosts to cope with the frequent disconnections and low bandwidth of the wireless networks. The model also supports mobile transactions to share data in an asynchronous manner via mobile sharing workspaces in the mobile affiliation workgroups. Moreover, the mobile sharing workspace within the mobile affiliation workgroup is fully distributed among connected and highly available mobile hosts. Therefore, the model can deal with the resource limitation of mobile hosts. Finally, the mobile data sharing mechanism supports both sharing data state and data status.

9.2 Future research

There is still work needed to be carried out in our MOWAHS mobile transaction processing system. The following topics are identified as possible future works in both the scientific and engineering dimensions.

The scientific dimension includes:

- *Mobile transaction agents to enhance the performance of mobile transaction processing systems.* Agents are autonomous programs that have the capacity to adapt to changing environmental conditions. Mobile agents are agent programs that have the ability to reallocate themselves among the active computers to carry out their goals [PRM00, Kan+04]. In our mobile transaction processing system, shared transactions that carry out the mobile sharing operations must handle the dynamic changes of the mobile environments and deal with the mobility of transactions across the mobile sharing workspaces. Therefore, the concepts of mobile agents can be applied in our mobile transaction system to achieve better performance and enhance mobility support. The choice of using the JavaSpaces technology to implement the export-import repository in our mobile transaction processing system can still be applied because mobile agents may be efficiently implemented using JavaSpaces technology [WS03].

- *Commit protocols for mobile distributed transactions.* Our mobile transaction processing system focuses on the mobile data sharing mechanisms, and the standard transactions have capacity to autonomously commit or abort in their operating workspaces (i.e., local commit in the local workspace or final commit at the global workspace). In mobile environments, the commit or abort of a transaction in the local workspace at a mobile host might also depend on the states of transactions that are being executed at other mobile hosts. Therefore, a further work on termination protocols for mobile distributed transactions in mobile environments will be beneficial.

- *Support of sharing database operations in mobile environments*. Our mobile data sharing mechanism focuses on supporting sharing of data state (i.e., values) and status (i.e., locks) among transactions at different mobile hosts. For future work, the mechanism will be extended to support sharing database operations among mobile transactions.

The engineering dimension includes:

- *Integration of all the components into the MOWAHS transaction processing system.* Due to time and resource constraints, we have not been able to carry out a full integration of our components in the mobile transaction processing system. Therefore, an important future work is to integrate all these individual components into the mobile transaction processing system. The integration will further allow us to carry out a full system testing.

- *Thorough performance testing of the mobile transaction processing system.* We have performed preliminary testing on the mobile data sharing mechanism, and the preliminary results have shown that there is significant improvement in the system throughput. However, these tests have not been carried out while taking into account dynamic changes of environmental conditions such as disconnections of mobile hosts from the mobile affiliation workgroups. Currently, we have only tested the performance of the individual system components separately.

- *Development of a mobile support system for physical allocation of mobile sharing workspace.* This is the engineering challenge related to the physical allocation of the export-import repository. In our mobile transaction processing system, the mobile sharing workspace is distributed over and allocated on several mobile computing hosts. Currently, the JavaSpaces technology is not designed to fully support the physically distribution of a mobile sharing workspace. Therefore, a possible future work is to design and develop a mobile support system for physical allocation of a mobile sharing workspace that matches the designated export-import repository.

References

[AC04] Y. J. Al-Houmaily and P. K. Chrysanthis: *1-2PC: the one-two phase atomic commit protocol*, ACM Symposium on Applied Computing (SAC), 2004, pp 684-691.

[Bar99] D. Barbará: *Mobile Computing and Databases - A Survey*, IEEE Transactions on Knowledge and Data Engineering (TKDE), 11(1), 1999, pp 108-117.

[BCM05] D. Bottazzi, A. Corradi and R. Montanari: *A context-aware group management middleware to support resource sharing in MANET environments*, International Conference on Mobile Data Management (MDM), 2005, pp 147-151.

[BCW95] W. Booth, G. G. Colomb and J. M. Williams: *The Craft of Research*, University Of Chicago Press, 1995.

[BF03] A. Brayner and J. d. A. M. Filho: *Sharing Mobile Databases in Dynamically Configurable Environments*, 15th International Conference on Advanced Information Systems Engineering (CAiSE), 2003, pp 724-737.

[Bha03] S. Bhalla: *Evolving a model of transaction management with embedded concurrency control for mobile database systems*, Information & Software Technology, 45(9), 2003, pp 587-596.

[BHG87] P. A. Bernstein, V. Hadzilacos and N. Goodman: *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.

[CC02a] S. Chang and D. Curtis: *An Approach to Disconnected Operation in an Object-Oriented Database*, International Conference on Mobile Data Management (MDM), 2002, pp 19-26.

[CCB02] D. Conan, S. Chabridon and G. Bernaro: *Disconnected Operations in Mobile Environments*, International Parallel and Distributed Processing Symposium (IPDPS), 2002, pp 192-199.

[CDK00] G. H. Coulouris, J. Dollimore and T. Kindberg: *Distributed Systems: Concepts and Design*, Pearson Education, 2001.

[Chr93] P. K. Chrysanthis: *Transaction Processing in Mobile Computing Environment*, IEEE Workshop on Advances in Parallel and Distributed Systems, 1993, pp 77-83.

[CLL03] E. Y. M. Chan, V. C. S. Lee and K.-W. Lam: *Using Separate Processing for Read-Only Transactions in Mobile Environment*, International Conference on Mobile Data Management (MDM), 2003, pp 106-121.

[Con+01] R. Conradi, M. Nygård, A. I. Wang and H. Ramampiaro: *Mobile Work Across Heterogeneous Systems*, 2001.

[CP98] R. Cáceres and V. N. Padmanabhan: *Fast and Scalable Wireless Handoffs in Support of Mobile Internet Audio*, Mobile Networks and Applications (MONET), 3(4), 1998, pp 351-363.

[CR94] P. K. Chrysanthis and K. Ramamritham: *Synthesis of Extended Transaction Models Using ACTA*, ACM Transactions on Database Systems (TODS), 19(3), 1994, pp 450-491.

[DG00] R. A. Dirckze and L. Gruenwald: *A pre-serialization transaction management technique for mobile multidatabases*, Mobile Networks and Applications (MONET), 5(4), 2000, pp 311-321.

[DHB97] M. H. Dunham, A. Helal and S. Balakrishnan: *A Mobile Transaction Model That Captures Both the Data and Movement Behavior.*, Mobile Networks and Applications (MONET), 2(2), 1997, pp 149-162.

[DK98] M. H. Dunham and V. Kumar: *Location Dependent Data and its Management in Mobile Databases*, Database and Expert Systems Applications (DEXA) Workshop, 1998, pp 414-419.

[DK99] M. H. Dunham and V. Kumar: *Impact of Mobility on Transaction Management*, ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE), 1999, pp 14-21.

[DMW01] Z. Ding, X. Meng and S. Wang: *O2PC-MT: A Novel Optimistic Two-Phase Commit Protocol for Mobile Transactions*, Database and Expert Systems Applications (DEXA), 2001, pp 846-856.

[Dye05] R. J. T. Dyer: *MySQL in a nutshell*, O'Reilly Media, 2005.

[Elm+90] A. K. Elmagarmid, Y. Leu, W. Litwin and M. Rusinkiewicz: *A Multidatabase Transaction Model for InterBase,* International Conference on Very Large Data Bases, 1990, pp 507-518.

[Elm+92] A. K. Elmagarmid: *Database Transaction Models for Advanced Applications*, Morgan Kaufmann, 1992.

[EN00] R. Elmasri and S. B. Navathe: *Fundamentals of Database Systems*, Addison Wesley, 2000.

[Esw+76] K. P. Eswaran, J. Gray, R. A. Lorie and I. L. Traiger: *The Notions of Consistency and Predicate Locks in a Database System*, Communications of the ACM (CACM), 19(11), 1976, pp 624-633.

[FHA99] E. Freeman, S. Hupfer and K. Arnold: *JavaSpaces : principles, patterns, and practice*, Addison-Wesley, 1999.

[FS99] J. Flinn and M. Satyanarayanan: *Energy-aware adaptation for mobile applications*, Symposium on Operating Systems Principles (SOSP), 1999, pp 48-63.

[GG00] M. M. Gore and R. K. Ghosh: *Recovery of Mobile Transactions*, Database and Expert Systems Applications (DEXA) Workshop, 2000, pp 23-27.

[GMS87] H. Garcia-Molina and K. Salem: *Sagas*, ACM SIGMOD International Conference on Management of Data, 1987, pp 249-259.

[GR93] J. Gray and A. Reuter: *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers, 1993.

[Gra78] J. Gray: *Notes on Data Base Operating Systems*, in (Ed): *Advanced Course: Operating Systems*, Springer, 1978, pp 393-481.

[Gra81] J. Gray: *The Transaction Concept: Virtues and Limitations*, Very Large Data Bases, 1981, pp 144-154.

[GUW01] H. Garcia-Molina, J. Ullman and J. Widom: *Database Systems: The Complete Book*, Prentice Hall, 2001.

[GW82] H. Garcia-Molina and G. Wiederhold: *Read-Only Transactions in a Distributed Database*, ACM Transactions on Database Systems (TODS), 7(2), 1982, pp 209-234.

[HAA02] J. Holliday, D. Agrawal and A. E. Abbadi: *Disconnection Modes for Mobile Databases*, Wireless Networks, 8(4), 2002, pp 391-402.

[Har84] T. Härder: *Observations on optimistic concurrency control schemes*, Information Systems, 9(2), 1984, pp 111-120.

[HB05] R. Høivik and G. G. Bergem: *Customizing Isolation Properties for Mobile Transactions*, Dept. of Computer and Information Science, NTNU, Norway, 2005.

[Hir+01] R. Hirsch, A. Coratella, M. Felder and E. Rodríguez: *A Framework for Analyzing Mobile Transaction Models*, Journal of Database Management, 12(3), 2001, pp 36-47.

[HM04] E. R. Harold and W. S. Means: *XML in a Nutshell*, O'Reilly Media, 2004.

[Hof02] Ø. Hoftun: *Mobile Nagging Geek Organizer*, Master thesis, Dept. of Computer and Information Science, NTNU, Norway, 2002.

[IBM] IBM DB2 Everyplace, IBM Corporation *http://www-306.ibm.com/software/data/db2/everyplace/*.

[JBE95] J. Jing, O. A. Bukhres and A. K. Elmagarmid: *Distributed Lock Management for Mobile Transactions*, International Conference on Distributed Computing Systems (ICDCS), 1995, pp 118-125.

[JHE99] J. Jing, A. Helal and A. K. Elmagarmid: *Client-Server Computing in Mobile Environments*, ACM Computing Surveys, 31(2), 1999, pp 117-157.

[Jini] Jini Specifications and API Archive, Sun Microsystems, Inc. *http://java.sun.com/products/jini/*.

[Kan+04] T. Kaneda, M. Shiraishi, T. Enokido and M. Takizawa: *Mobile Agent Model for Transaction Processing on Distributed Objects*, International Conference on Advanced Information Networking and Applications (AINA), 2004, pp 506-511.

[KK00] K.-I. Ku and Y.-S. Kim: *Moflex Transaction Model for Mobile Heterogeneous Multidatabase Systems*, Research Issues in Data Engineering (RIDE), 2000, pp 39-46.

[KLH03] S. S. Kim, S. K. Lee and C.-S. Hwang: *Using reordering technique for mobile transaction management in broadcast environments*, Data & Knowledge Engineering, 45(1), 2003, pp 79-100.

[KR81] H. T. Kung and J. T. Robinson: *On Optimistic Methods for Concurrency Control*, ACM Transactions on Database Systems (TODS), 6(2), 1981, pp 213-226.

[KTW97] J. Klingemann, T. Tesch and J. Wäsch: *Enabling Cooperation among Disconnected Mobile Users*, International Conference on Cooperative Information Systems (CoopIS), 1997, pp 36-46.

[KU99] E. Kayan and Ö. Ulusoy: *An Evaluation of Real-Time Transaction Management Issues in Mobile Database Systems*, The Computer Journal, 42(6), 1999, pp 501-510.

[Kum+02] V. Kumar, N. Prabhu, M. H. Dunham and A. Y. Seydim: *TCOT-A Timeout-Based Mobile Transaction Commitment Protocol*, IEEE Transactions on Computers, 51(10), 2002, pp 1212-1218.

[Liu+05] J. Liu, D. Sacchetti, F. Sailhan and V. Issarny: *Group management for mobile Ad Hoc networks: design, implementation and experiment*, International Conference on Mobile Data Management (MDM), 2005, pp 192-199.

[LLK01] K.-W. Lam, V. C. S. Lee and T.-W. Kuo: *Group Consistency for Read-Only Transactions in Mobile Environments*, International Parallel and Distributed Processing Symposium (IPDPS), 2001, pp 1009-1016.

[LN05a] H. N. Le and M. Nygård: *A Mobile Affiliation Model for Supporting Mobile Collaborative Work*, Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS), 2005, pp 649-660.

[LN05b] H. N. Le and M. Nygård: *Mobile Transaction System for Supporting Mobile Work*, 7th International Database and Expert Systems Applications (DEXA) Workshop on Mobility in Databases and Distributed Systems (MDDS), 2005, pp 1090-1094.

[LNR04] H. N. Le, M. Nygård and H. Ramampiaro: *A Locking Model for Mobile Databases in Mobile Environments*, International Conference on Database and Applications (DBA), 2004, pp 49-55.

[Mad+02] S. K. Madria, M. K. Mohania, S. S. Bhowmick and B. K. Bhargava: *Mobile data and transaction management*, Information Sciences, 141(3-4), 2002, pp 279-309.

[MB01] S. K. Madria and B. K. Bhargava: *A Transaction Model to Improve Data Availability in Mobile Computing*, Distributed and Parallel Databases, 10(2), 2001, pp 127-160.

[MB98b] S. K. Madria and B. K. Bhargava: *A Transaction Model for Mobile Computing*, International Database Engineering and Application Symposium (IDEAS), 1998, pp 92-102.

[MBB02] S. K. Madria, M. Baseer and S. S. Bhowmick: *A Multi-version Transaction Model to Improve Data Availability in Mobile Computing*, International Conference on Cooperative Information Systems (CoopIS/DOA/ODBASE), 2002, pp 322-338.

[Mic] Microsoft SQL Server CE, Microsoft Corporation *http://msdn.microsoft.com/library/*.

[Mil+00] D. S. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler and S. Zhou: *Process migration*, ACM Computing Surveys, 32(3), 2000, pp 241-299.

[Mos85] J. E. B. Moss: *Nested transactions: an approach to reliable distributed computing*, Massachusetts Institute of Technology, 1985.

[MRX03] R. k. Majumdar, K. Ramamritham and M. Xiong: *Adaptive Location Management in Mobile Environments*, International Conference on Mobile Data Management (MDM), 2003, pp 196-211.

[Mur01] V. K. Murthy: *Seamless Mobile Transaction Processing: Models, Protocols and Software Tools*, International Conference on Parallel and Distributed Systems (ICPADS), 2001, pp 147-156.

[MV00] K. A. Momin and K. Vidyasankar: *Flexible Integration of Optimistic and Pessimistic Concurrency Control in Mobile Environments*, Advances in Databases and Information Systems - Database Systems for Advanced Applications (ADBIS-DASFAA), 2000, pp 346-353.

[Ora] Oracle Database Lite, Oracle Corporation *http://www.oracle.com/technology/products/lite/index.html*.

[OV99] M. T. Özsu and P. Valduriez: *Principles of Distributed Database Systems*, 1999.

[PB99] E. Pitoura and B. K. Bhargava: *Data Consistency in Intermittently Connected Distributed Systems*, IEEE Transactions on Knowledge and Data Engineering (TKDE), 11(6), 1999, pp 896-915.

[PKH88] C. Pu, G. E. Kaiser and N. C. Hutchinson: *Split-Transactions for Open-Ended Activities.*, Very Large Data Bases (VLDB), 1988, pp 26-37.

[PLZ05] S. Pradhan, E. Lawrence and A. Zmijewska: *Bluetooth as an Enabling Technology in Mobile Transactions*, International Symposium on Information Technology: Coding and Computing (ITCC), 2005, pp 53-58.

[PMR00] G. P. Picco, A. L. Murphy and G.-C. Roman: *Developing mobile computing applications with LIME*, International Conference on on Software Engineering (ICSE), 2000, pp 766-769.

[PP00] E. M. Phillips and D. S. Pugh: *How to get a PhD - A handbook for students and their supervisors*, Open University Press, 2000.

[PR02] C. Pedregal-Martin and K. Ramamritham: *Support for Recovery in Mobile Systems*, IEEE Transactions on Computers, 51(10), 2002, pp 1219-1224.

[Pra+04] N. Prabhu, V. Kumar, I. Ray and G.-C. Yang: *Concurrency Control in Mobile Database Systems*, International Conference on Advanced Information Networking and Applications (AINA), 2004, pp 83-86.

[Pre+00] N. M. Preguiça, C. Baquero, F. Moura, J. L. Martins, R. Oliveira, H. J. L. Domingos, J. O. Pereira and S. Duarte: *Mobile Transaction Management in Mobisnap*, Advances in Databases and Information Systems - Database Systems for Advanced Applications (ADBIS-DASFAA), 2000, pp 379-386.

[PS98] E. Pitoura and G. Samaras: *Data Management for Mobile Computing*, Kluwer Academic Publishers, 1998.

[PSP00] S. Papastavrou, G. Samaras and E. Pitoura: *Mobile Agents for World Wide Web Distributed Database Access*, IEEE Transactions on Knowledge and Data Engineering, 12(5), 2000, pp 802-820.

[Rak98] A. Rakotonirainy: *Adaptable Transaction Consistency for Mobile Environments*, Database and Expert Systems Applications (DEXA) Workshop, 1998, pp 440-445.

[Ram01] H. Ramampiaro: *CAGISTrans: Adaptable Transactional Support for Cooperative Work*, Dr.ing thesis, Norwegian University of Science and Teachnology (NTNU), 2001.

[Ram+03] H. Ramampiaro, A. I. Wang, C.-F. Sørensen, H. N. Le and M. Nygård: *Requirement Indicators for Mobile Work: The MOWAHS Approach*, IASTED International Multi-Conference on Applied Informatics, 2003, pp 1153-1160.

[Rat+01] D. Ratner, P. L. Reiher, G. J. Popek and G. H. Kuenning: *Replication Requirements in Mobile Environments*, Mobile Networks and Applications (MONET), 6(6), 2001, pp 525-533.

[RC96] K. Ramamritham and P. Chrysanthis: *Advances in Concurrency Control and Transaction Processing*, IEEE Computer Society Press, 1996.

[RK99] P. K. Reddy and M. Kitsuregawa: *Speculative Lock Management to Increase Concurrency in Mobile Environments*, International Conference on Mobile Data Access (MDA), 1999, pp 82-96.

[RN99] H. Ramampiaro and M. Nygård: *Cooperative Database System: A Constructive Review of Cooperative Transaction Models*, Database Applications in Non-Traditional Environments (DANTE), 1999, pp 315-324.

[RRP04] D. Ratner, P. L. Reiher and G. J. Popek: *Roam: A Scalable Replication System for Mobility*, Mobile Networks and Applications (MONET), 9(5), 2004, pp 537-544.

[SAE01] R. Sher, Y. Aridor and O. Etzion: *Mobile Transactional Agents*, International Conference on Distributed Computing Systems (ICDCS), 2001, pp 73-80.

[Sch02] R. Schneiderman: *The Mobile Technology Question and Answer Book A Survival Guide for Business Managers*, American Management Association, 2002.

[Ser02] P. Serrano-Alvarado: *Defining an Adaptable Mobile Transaction Service*, Extending Database Technology (EDBT) Workshops, 2002, pp 616-626.

[SGS94] K. Salem, H. Garcia-Molina and J. Shands: *Altruistic Locking*, ACM Transactions on Database Systems (TODS), 19(1), 1994, pp 117-165.

[SM05] R. Sheldon and G. Moes: *Beginning MySQL*, Wiley Pub., 2005.

[Sør+02] C.-F. Sørensen, A. I. Wang, H. N. Le, H. Ramampiaro, M. Nygård and R. Conradi: *The MOWAHS Characterisation Framework for Mobile Work*, IASTED International Conference on Applied Informatics, 2002, pp 258-264.

[Sør+05] C.-F. Sørensen, A. I. Wang, H. N. Le, H. Ramampiaro, M. Nygård and R. Conradi: *Using the MOWAHS Characterisation Framework for Development of Mobile Work Applications*, International Conference on Product Focused Software Process Improvement (PROFES), 2005, pp 128-142.

[SRA04] P. Serrano-Alvarado, C. Roncancio and M. E. Adiba: *A Survey of Mobile Transactions*, Distributed and Parallel Databases, 16(2), 2004, pp 193-230.

[TLP99] C. L. Tan, K. M. Lye and S. Pink: *A Fast Handoff Scheme for Wireless Networks*, ACM International Workshop on Wireless Mobile Multimedia (WOWMOM), 1999, pp 83-90.

[Var03] U. Varshney: *Location management for mobile commerce applications in wireless Internet environment*, ACM Transactions on Internet Technology (TOIT), 3(3), 2003, pp 236-255.

[WC99] G. D. Walborn and P. K. Chrysanthis: *Transaction Processing in PRO-MOTION*, ACM Symposium on Applied Computing (SAC), 1999, pp 389-398.

[Wei91] G. Weikum: *Principles and Realization Strategies of Multilevel Transaction Management*, ACM Transactions on Database Systems, 16(1), 1991, pp 132-180.

[WR96] C. D. Wilcox and G.-C. Roman: *Reasoning About Places, Times, and Actions in the Presence of Mobility*, IEEE Transactions on Software Engineering, 22(4), 1996, pp 225-247.

[WS03] A. I. Wang and C.-F. Sørensen: *A Comparison of Two Different Java Technologies to Implement a Mobile Agent System*, IASTED International Conference on Applied Informatics, 2003, pp 1039-1044.

[Xio+02] M. Xiong, K. Ramamritham, J. A. Stankovic, D. F. Towsley and R. M. Sivasankaran: *Scheduling Transactions with Temporal Constraints: Exploiting Data Semantics*, IEEE Transactions on Knowledge and Data Engineering, 14(5), 2002, pp 1155-1166.

[YAC04] M. Younas, I. Awan and K.-M. Chao: *Performance Analysis of Preemptive Resume Scheduling in Mobile Transactions*, International Conference on Advanced Information Networking and Applications (AINA), 2004, pp 249-254.

[YHW04] A. Yendluri, W.-C. Hou and C.-F. Wang: *Improving Concurrency Control in Mobile Databases*, 9th International Conference on Database Systems for Advances Applications (DASFAA), 2004, pp 642-655.

[Zha+99] Y. Zhang, Y. Kambayashi, X. Jia, Y. Yang and C. Sun: *On Interactions Between Coexisting Traditional and Cooperative Transactions*, International Journal of Cooperative Information Systems (IJCIS), 8(2-3), 1999, pp 87-110.

Notations

Symbol	Description
S_i	a database server
X	a shared data item
V_X	the value of shared data item X
X_{DA}	the dependency awareness set on shared data item X
X_{CA}	the conflict awareness set on shared data item X
l_X	a lock operation on shared data item X
ul_X	an unlock operation on shared data item X
X_{lock_mode}	a lock applied on shared data item X
X_R	a read lock on X
X_W	a write lock on X
X_{Rp}	a pseudo-read lock on X
Op_i	a database operation
R_X	a read operation on shared data item X
W_X	a write operation on shared data item X
MH_i	a mobile host
T^i	a transaction
T^c	a conflicting transaction
T^{Dor}	a delegator transaction
T^{Dee}	a delegatee transaction
T^E	an export transaction
T^I	an import transaction
$T^{Dor.E}$	an export transaction of delegator transaction T^{Dor}
$T^{Dee.I}$	an import transaction of delegatee transaction T^{Dee}
T_i^A	the anchor transaction of mobile host MH_i
T_i^k	a local transaction T^k at mobile host MH_i
T_i^{PD}	a pseudo-delegator transaction of delegator transaction T_i^{Dor}
$\mathfrak{I}_i$	a set of transactions

Symbol	Description
D_i	a data set
D_i^R	a read data set
D_i^W	a write data set
L_i	the lock set corresponding to data set D_i
L_i^R	the read lock set corresponding to data set D_i^R
L_i^W	the write lock set corresponding to data set D_i^W
D_i^A	the acquired data set at mobile host MH_i
D_i^{RA}	the acquired read data set at mobile host MH_i
D_i^{WA}	the acquired write data set at mobile host MH_i
L_i^A	the lock set corresponding to data set D_i^A
L_i^{RA}	the read lock set corresponding to data set D_i^{RA}
L_i^{WA}	the write lock set corresponding to data set D_i^{WA}
D_i^G	the granted (i.e., locked and cached) data set at mobile host MH_i
D_i^{RG}	the granted (i.e., locked and cached) read data set at mobile host MH_i
D_i^{WG}	the granted (i.e., locked and cached) write data set at mobile host MH_i
L_i^G	the lock set corresponding to data set D_i^G
L_i^{RG}	the read lock set corresponding to data set D_i^{RG}
L_i^{WG}	the write lock set corresponding to data set D_i^{WG}
D_i^{GR}	the replica of granted data set D_i^G
D_i^{RGR}	the replica of granted read data set D_i^{RG}
D_i^{WGR}	the replica of granted write data set D_i^{WG}
L_i^{GR}	the replica of lock set L_i^G
L_i^{RGR}	the replica of read lock set L_i^{RG}
L_i^{WGR}	the replica of write lock set L_i^{WG}
L_i^{AR}	the additional read lock set requested by anchor transaction T_i^A
L_i^{AW}	the additional write lock set requested by anchor transaction T_i^A
L_i^{DR}	the delegated read lock set released by anchor transaction T_i^A
L_i^{DW}	the delegated write lock set released by anchor transaction T_i^A
D_i^j	the acquired data set of transaction T_i^j
D_i^{jR}	the acquired read data set of transaction T_i^j
D_i^{jW}	the acquired write data set of transaction T_i^j

Symbol	Description
$T_i^k \rightarrow T_j^l$	transaction T_i^k must be executed before transaction T_j^l
$T_i^k \rightarrow\bullet T_j^l$	transaction T_j^l must be executed after transaction T_i^k and before any other updating transaction T_i^n where $T_i^k \rightarrow T_i^n$
LAD_i	the locally aborted delegator transaction (*LocalAbortedDelegator*) set at mobile host MH_i
LC_i	the locally committed transaction (*LocalCommitted*) set at mobile host MH_i
PC	the pending commit transaction (*PendingCommit*) set
GAD	the globally aborted delegator transaction (*GlobalAbortedDelegator*) set
GC	the globally committed transaction (*GlobalCommitted*) set