uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Rong Tang

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Master of Computer Science

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Transaction Management in Peer-to-peer Multidatabase Systems

TITRE DE LA THÈSE / TITLE OF THESIS

Iluju Kiringa

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

Stan Matwin

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Liam Peyton

Sivarama Dandamudi

Gary W. Slater

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

Transaction Management in Peer-to-peer Multidatabase Systems

by

Rong Tang

A Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
Of the University of Ottawa
In partial fulfillment of the requirements
For the degree in Computer Science

School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa

The Masters program in Computer Science is a joint program with Carleton University, administered by the Ottawa Carleton Institute for Compurter Science

Your file Votre référence
ISBN: 0-494-11427-4
Our file Notre référence
ISBN: 0-494-11427-4

Abstract

Peer-to-peer multidatabase systems (P2P MDBSs) are dynamic networks of peers with total absence of any global schema, any central administrative authority, any data integration, any global access to multiple databases, permanent participation of databases, etc. Global and local transactions are supported in P2P MDBSs. A global transaction generates descendent transactions when it is propagated to other peers over acquaintances in a P2P MDBS. Descendent transactions are translations of the original global transaction based on mappings between attributes in two acquainted peers. We present a serializability theory for transactions in P2P MDBSs, and then a concurrency control protocol is proposed to ensure the global serializability of global histories by controlling the consistency over each single acquaintance. The correctness of the concurrency control protocol is proved by using the developed theory.

Acknowledgements

I extend my sincere gratitude and appreciation to many people who made this master's thesis possible. I am very grateful to my supervisor Dr. Iluju Kiringa who guided me proceed towards the completion of this thesis. His proficient knowledge in the field of database, invaluable suggestions, and profound research experience helped me in all the time of my research for and writing of this thesis.

Many thanks go to Mehedi Masud and Swadhin Saha for the help they provided in the implementation and experiment done for this thesis. Special thanks also go to my friend Quintin Armour for the time that he spent on reviewing my thesis.

I would like to acknowledge the support and facilities I received from the staff of the School of Information Technology and Engineering.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The Problem

Transaction support is one of the most important functions that a Database Management System (DBMS) should provide. Transaction management is intended to ensure the consistency and reliability of a database system when multiple transactions are being processed concurrently in the DBMS.

A transaction is an action, or series of actions, carried out by a single user or application program, which reads or updates the contents of the database [1]. In central database systems, transactions must possess four basic properties, which are the so called *ACID* properties, to maintain the database consistency.

- ***A****tomicity*: Either all operations of a transaction are properly performed in the database or none are.
- ***C****onsistency*: A transaction must transform the database from one consistent state to another consistent. The transaction's execution in isolation preserves the consistency of the database.

- *Isolation*: Each transaction is executed independently in the database system and the DBMS guarantees that intermediate transaction results are not visible to other concurrently executed transactions.

- *Durability*: The values changed by a transaction are permanently recorded in the database after the transaction successfully completes.

Early and traditional, centralized database systems follow a unique central scheme for database management. Later database management system models have been developed along the development of communication and database technologies. These models include distributed DBMSs and Multidatabase Management Systems. A multidatabase system (MDBS) is a database management system that allows users to access data located in multiple autonomous local database systems [2]. The MDBS logically integrates all local DBMSs and creates an illusion of a single database system to hide the intricacies of different local DBMSs. Users can access data contained in multiple local DBMSs through the interface of the multidatabase management system. Two types of transactions are supported in a conventional MDBS:

- *Local transactions*, which are executed by a local DBMS and access data in the single local DBMS. These transactions are outside of the control of the MDBS.

- *Global transactions*, which are executed under the control of the MDBS and access data items located in multiple local database systems. A global transaction is divided into a number of subtransactions, each of which is an ordinary local transaction from the point of view of the local DBMS where the subtransaction is executed.

Peer-to-peer Multidatabase Systems (P2P MDBSs)[3] are a new multidatabase system architecture which combines the functionalities of both P2P computing systems[4] and database systems. P2P Multidatabase Systems are developed based on the concepts of P2P computing systems and conventional database systems. A P2P computing system is a self-organizing information sharing system. Each participating node, called a peer,

provides access to (some of) its resources to other peers over a common network or via the Internet [4]. Each peer acts as both a client and a server, and peers in such a system are fully autonomous. In P2P MDBSs, a peer is a database system located at a node of the P2P network, which is called a peer database. Peer databases share their data to other peers in the common P2P network. By combining the functionalities of P2P computing systems and database management systems, P2P MDBSs provide users approaches to consult a P2P database group by interacting with one peer database system. A P2P database group is called an *interest group*[5]. In general, an interest group is defined as a community of peer databases centered on a common business [5].

The goal of P2P MDBSs is to share and exchange data between autonomously structured peer databases. P2P MDBSs are similar to conventional multidatabase systems in the sense that they both consist of a collection of local database management systems, each of which may follow a different concurrency control mechanism. Each such component database may, in general, be distributed, autonomous, and heterogeneous. However, conventional MDBSs are founded on the key concepts of global schemas, global accesses to data items, and global concurrency control mechanisms, while P2P MDBSs build dynamic networks of peers with the total absence of any global schema, any central administrative authority, any data integration, any global access to multiple databases, and permanent participation of databases, etc [3]. Each peer database is managed by a p*eer data management system*(PDBMS). Typically, a PDBMS has a P2P layer that interoperates its local peer databases with other local peer databases. At any time, new peers may join or leave a P2P network by using the appropriate protocol. To join the P2P network, a peer database establishes acquaintances with other peers that are already part of the P2P MDBS. An *acquaintance* is a connection between two peers and is established by generating mapping tables and mapping expressions on both peers. The acquaintances are transient since each peer is fully autonomous and joins or leaves the network at its own will. The acquainted peers are called *acquaintees*. *Mapping tables*[6] and *mapping expressions*[6] over each acquaintance are proposed as ways of constraining the exchange of data across heterogeneous peers in P2P MDBSs. Mapping tables specify how data might be associated by listing pairs of corresponding values for search domains

that are used by different peers. A mapping expression relates two different schemas and realizes a schema level correspondence between two different peers. Mapping tables and mapping expressions evolve dynamically with the development of two acquainted peers [3].

In this thesis, we focus on the issue of transaction management in P2P MDBSs. Because of the similarity in structure between MDBSs and P2P MDBSs, we investigate the problem of transaction management in P2P MDBSs based on the theories developed for MDBSs. On the other hand, the property of no central control scheme in P2P MDBSs makes it possible for us to relax the database consistency criteria of MDBSs to meet the requirements of P2P MDBSs.

Since a P2P MDBS has a structure similar to that of MDBSs, we also define two types of transactions: *global transactions* and *local transactions* in P2P MDBSs. However, the definitions of local transactions and global transactions in P2P MDBSs are different from the ones defined in conventional MDBSs. The global transactions in P2P MDBSs are not controlled by any global management system, while the global transactions in MDBSs are managed by the MDBSs. In P2P MDBSs, a transaction is always initiated as a local transaction with respect to a peer to which user submits it. Based on the mapping information over each of the peer's acquaintances, a transaction may spawn child transactions to other acquainted peers. A child transaction is a translation of the transaction based on the mapping information over an acquaintance. Thus the transaction is translated into child transactions instead of being divided into subtransactions as MDBSs do. The transaction may be translated *completely* or *partially* into a child transaction depending on how the data, that the transaction accesses, are being mapped over the acquaintance. We say that the child transaction is a *complete translation* of a transaction if all operations of the transaction can be translated over the acquaintance and the child transaction contains all these translated operations; otherwise, the child transaction is a *partial translation* of the transaction if only a subset of the transaction's operations can be translated over the acquaintance. All these issues will be discussed in detail in later sections.

This thesis is a first step to explore the issue of transaction management in P2P MDBSs. Our objective is to provide solutions to some basic problems related to transaction management in P2P MDBSs. The research objectives for this thesis are as follows.

- Provide precise definitions to transactions and other concepts related to the issue of transaction management in P2P DBMSs;
- Figure out the properties of transactions under the management scheme of P2P MDBSs;
- Find out a transaction model that is suitable for the structure and management scheme of P2P MDBSs;
- Analyze the consistency requirements and identify a correctness criteria that can ensure the consistency of P2P MDBSs;
- Develop a concurrency control protocol based on the correctness criteria identified.

We have made some assumptions in the study of the issues above. Some of them are general for MDBSs, while some others are made only for P2P MDBSs. The details of the assumptions are as follows:

- Assumption 1: Each peer ensures its local database consistency though the concurrency control mechanisms may vary in different peers. This is a general assumption for MDBSs.
- Assumption 2: A global transaction spawns at most one child transaction to an acquainted peer when a child transaction spawning is possible over an acquaintance, whether the child transaction is a complete or partial translation of the global transaction. The child transaction is an atomic unit which may contain multiple subtransactions that result from the global transaction's multiple translations over the single acquaintance. This is an assumption made for P2P MDBSs. It simplifies

the complexity of the concurrency control for global transactions. It is also practical in P2P MDBSs.

1.2 Motivation Example

We use a P2P MDBS which consists of four Digital Library (DGL) Databases: p_1, p_2, p_3, and p_4 as our motivating example. We assume that these four digital libraries are autonomous, distributed, and heterogeneous. Digital libraries are similar to normal libraries. The only difference is that digital libraries make their resources available online and provide services to users on their websites. The basic software infrastructure for a digital library is a database system, which manages information pertaining to resources, saves users' personal information, processes transactions submitted to the system, and provides user accesses to online resources, etc. In this example, these four digital libraries p_1, p_2, p_3, and p_4 intend to establish a P2P multidatabase system to share their resources in a common P2P network and provide user services through the P2P network. Users can access the shared resources of these four libraries by interacting with one peer database system. Since digital libraries preexist and are designed independently, they may follow different information management schemas. Each peer has a part of its export schemas (ES) that is shared with acquainted peers, and another part that is local schemas (LS). We use books as the example of resources to illustrate the business of digital libraries. The acquaintances among these four peers are shown in Figure 1.1. In Figure 1.1, each line between two peers represents an acquaintance. A double headed arrow ($\leftrightarrow$) is used to denote an acquaintance. The acquaintances in DGL include $p_1 \leftrightarrow p_2$, $p_1 \leftrightarrow p_3$, $p_2 \leftrightarrow p_3$, and $p_3 \leftrightarrow p_4$.

Each digital library has its own registered users, who are members of the library. Members have privileges to access the library's resources online. Users can read and download a book from the library where heshe registered. However, users will be charged when they read or download a book from the digital library. User can access a book in different units, which include the whole book, chapters, sections, or pages. Downloading a book is charged by the access unit. Each library defines its own rate and rules for its

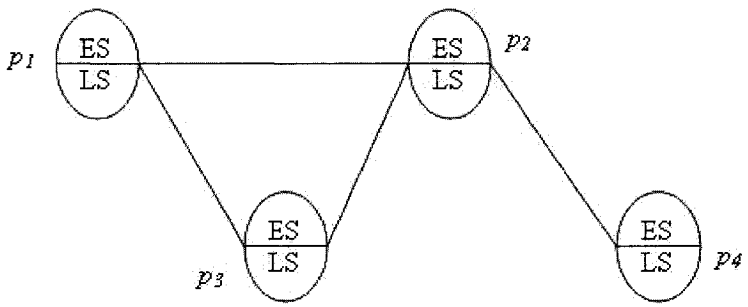

Figure 1.1: A Digital Library P2P MDBS

members to access books. A user may register in more than one library. The digital libraries set up a personal account, which is used to record the user's book accesses and accumulated charges, for each of their members. The balance of the account shows the accumulated charges that the user should pay. Those users, who register in multiple libraries, may use different usernames and passwords in various libraries. Peers may also reach some agreement to share their members. As long as an acquaintance is established, members of one peer can also be members of the other peer over the acquaintance.

1.2.1 Schemas

Here, we instantiate three peers p_1, p_2, and p_3. The schemas for each peer are shown below. We do not instantiate p_4 because we do not need detail data from p_4. But we need such a peer to illustrate the propagation of global transactions in the P2P network.

OTT_Book(call#, title, addr, status, ISBN)
OTT_Member(mid,username,balance)
OTT_Rate(call#,downloadR)

(1) Schemas for p_1: the *OTT* Digital Library Database (*OTT_DGL_DB*)

We assume that *OTT_DGL_DB* includes three schemas: *OTT_Book*, *OTT_Member*, and *OTT_Rate*. The *OTT_Book* stores each book's call number, title, online address, status, and ISBN. The call number is the identifier of a book. The status of a book can be public or private. Only books that are public can be accessed by users from other peers. For each member, the *OTT_DGL_DB* stores his/her identifier, username, password, and balance in *OTT_Member*. Instance relations for the *OTT_DGL_DB* will be shown in the next section. The rates to download books are stored in the schema *OTT_Rate*.

NY_Book(callNo, title, link, status)

NY_Member(mid, name, password, balance)

NY_Rate(callNO, downloadR)

(2) Schemas for p_2: the *NY* Digital Library Database (*NY_DGL_DB*)

NY_DGL_DB has the same structure of schemas as *OTT_DGL_DB*. It includes three schemas: *NY_Book*, *NY_Member*, and *NY_Rate*. It stores the information for books, which includes the identifier, title, access link (address to access a book), and status in the relation *NY_Book*. In addition, it sets up an account for each member to store hisher identifier, name, password, and account balance in *NY_Member*. It also defines its own book access rates in the schema *NY_Rate*.

TOR_Book(bid,title,address,ISBN,status)

TOR_Member(mid, name, balance, registerDate)

TOR_Rate(bid, downloadR)

(3) Schemas for p_3: the *TOR* Digital Library Database (*TOR_DGL_DB*)

Similarly, *TOR_DL_DB* stores for each book its identifier, title, access address, and status in the schema *TOR_Book*. For each member, it stores hisher identifier, name, balance, and register date in *TOR_Member*. Finally, it stores its books' access rates in *TOR_Rate*.

Call#	title	addr	status	ISBN
QA 76.9.D3 E57 1989	Fundamentals of database systems	PDF	public	0805301453
QA 76.545.B47 1997	Principles of transaction processing	PDF	public	1558604154
QA 76.9.D3 D37 1995	On to Java	PDF	public	0201385988

(a) OTT_Book

Call#	downloadR
QA 76.9 .D3 E57 1989	65.00
QA 76.545. B47 1997	75.00
QA 76.9.D3 D37 1995	80.00

(b) Ott_Rate

mid	username	balance
1	Mary	20.00
2	John	18.00
3	Mike	35.00

(c) OTT_Member

Figure 1.2: Instances for p_1: *OTT_DGL_DB*

1.2.2 Instantiated Relations

In this section, we instantiate relations for these three digital libraries: p_1, p_2, and p_3.

1.2.3 Mapping Tables

A mapping table is an instantional relation whose tuples associate attribute values of different acquainted peer databases. In P2P MDBSs, mapping tables have to be created before any data is shared or exchanged between acquainted peers. In the digital library group, the most possible mapping tables we consider are related to books, such as mapping tables on book identifiers, titles, download addresses, and download rates, etc. Additionally, members may also be mapped since a user may register in more than one peer library in the P2P group. The user may use different usernames and passwords in various peers in which heshe gets registered. When two peers are establishing an acquaintance, a mapping table will be created to list all the members' ID who own accounts in both peers. Only those members listed in the mapping table can access books in both

callNo	title	link	status
JSE 89-926	Fundamentals of database systems	PDF	public
JSE 97-84	Principles of transaction processing	PDF	public
JSE 99-718	On to Java	PDF	public

(a) NY_Book

mid	name	balance
M1	Tom	8.00
M2	Mary	12.50
M3	Peter	31.00

(b) NY_Member

callNo	downloadR
JSE 89-926	65.00
JSE 97-84	72.00
JSE 99-718	80.00

(c) NY_Rate

Figure 1.3: Instances for p_2: *NY_DGL_DB*

bid	title	ISBN	address	status
QA76.9.D3U44 1982 c.3	Principles of database systems	0914894366	PDF	public
QA76.545.B47	Principles of transaction processing	1558604154	PDF	public
QA76.9.D3 E57 1994	Fundamentals of database systems	0805317481	PDF	public

(a) TOR_Book

mid	name	balance	registerDate
1	Mary	5.00	2004718
2	Tom	9.45	2004516
3	Cindy	3.35	2004211

(b) TOR_Member

bid	downloadR
QA76.9.D3U44 1982 c.3	70.00
QA76.545.B47	78.00
QA76.9.D3 E57 1994	66.00

(c) TOR_Rate

Figure 1.4: Instances for p_3: *TOR_DGL_DB*

acquainted peers through the P2P network. Some other *identity mapping tables* for the attributes: *title*, *download address*, *download rate*, and *member balance* are also created. The identity mapping tables are mapping tables containing variables. In identity mapping tables, one value in the domain of the attribute in the left column can be mapped to a value in the domain of the attribute in the right column. When the variables in the identity mapping tables are identical, it shows that a data value of the first database is mapped to the same data value in the second database [6]. The instantiated mapping tables for the *DGL* P2P MDBS are shown from Figure 1.5 - 1.7 :

call#	callNo
QA 76.9 .D3 E57 1989	JSE 89-926
QA 76.545 .B47 1997	JSE 97-84
QA 76.73 .J38 W56 1998	JSE 99-718

(a) Call#2callNo

OTT.addr	NY.link
X	Y

(b) addr2link

OTT. title	NY.title
X	X

(c) title2title

OTT.downloadR	NY.downloadR
X	Y

(d) downloadR2downloadR

OTT. mid	NY.mid
1	M2

(e) mid2mid

OTT.balance	NY. balance
X	Y

(f) balance2 balance

Figure 1.5: Mapping tables between p_1 and p_2

1.2.4 Transaction Examples

Most likely, users want to search books and get some of books' information in the digital library P2P MDBS. For example, when a user uses book title to search a book and its download rate, the user intends to find out a link with the cheapest download rate to download the book; when a member checks his/her account balance online, he/she needs

callNo	bid
JSE 89-926	QA76.9.D3 E57 1994
JSE 97-84	QA 76.545 .B47

(a) callNo2bid

NY. mid	TOR.mid
M1	2
M2	1

(b) mid2mid

NY. title	TOR.title
X	X

(c) title2title

NY.link	TOR.address
X	Y

(d) link2address

NY.balance	TOR.balance
X	Y

(e) balance2balance

NY.downloadR	TOR.downloadR
X	Y

(f) downloadR2downloadR

Figure 1.6: Mapping tables between p_2 and p_3

answers from all those peers where he/she registers; when all peers in the entire group agree to use the same access rate for a book, the access rate needs to be updated in every peer database who owns the book. All these are examples of transactions which may be processed on mores than one peer database. We say that a transaction, which is executed on more than one peer database, is a global transaction of the P2P MDBS. Let's see some transaction examples in the *DGL* P2P MDBS. We suppose that all transactions are initiated on p_1.

Example 1.2.1 *Search a book by its title, Principles of transaction processing, and return user the links to access the book.*

T_1: *begin*
read(OTT_Book.title, X);
if(X = Principles of transaction processing) {
read(OTT_Book.addr, Y); }
return Y;
end

call#	bid
QA 76.545 .B47 1997	QA 76.545 .B47
QA 76.9 .D3 E57 1989	QA76.9.D3 E57 1994

(a) call#2bid

OTT.ISBN	TOR.ISBN
1558604154 (pbk.)	1558604154 (pbk.)
0805301453	0805317481

(b) ISBN2ISBN

OTT. mid	NY.mid
1	1

(c) mid2mid

OTT.balance	TOR.deposit
X	Y

(d) balance2deposit

OTT. title	TOR.title
X	X

(e) title2title

OTT.addr	TOR.address
X	Y

(f) addr2address

OTT.downloadR	TOR.downloadR
X	Y

(g) downloadR2downloadR

Figure 1.7: Mapping tables between p_1 and p_3

Since both p_2 and p_3 own the book and the book's status is 'public' in both peers, p_1 propagates T_1 to p_2 and p_3 as a global transaction respectively after T_1 is initiated on p_1.

Example 1.2.2 *Update the download rate of a book:* Fundamentals of Database Systems *in peer databases, which own the book, by 10%.*

T_2 is propagated to p_2 and p_3, and is executed on each of them because both peers own the book and the download rate is being mapped over every acquaintance in the DGL *group.*

T_2: *begin*

read(OTT_Book.title, X)

if(X = FundamentalsofDatabaseSystems) {

read (OTT_Rate.downloadR, Y);

$Y = Y * 10\%$;

write(OTT_Rate.downloadR, Y);

}

end

Example 1.2.3 *The user with a user name* 'Mary' *(mid* = 1*) wants to download a book:* Principles of Database Systems *and the P2P MDBS charges her after she downloads the book.*

T_3: *begin*

read(OTT_Book.title, X);

if (X =" PrinciplesofDatabaseSystems") {

download(Mary, X);

read (Mary, balance, Y);

read(downloadR, Z);

$Y = Y + Z$;

write (balance, Y);

}

else

return "The book is not available!";

end

Since the attributes: title, balance, mid, *and* downloadR *are all being mapped in mapping tables,* p_1 *propagate* T_3 *to both* p_2 *and* p_3*, respectively. Finally,* T_3 *commits on* p_3

because only p_3 owns the book.

1.3 Our Soluton

An acquaintance is a connection unit in the network of a P2P MDBS. All acquaintances in an interest group together construct the network of the P2P MDBS. To ensure the consistency of the P2P MDBS, we need to maintain the consistency of each acquaintance first. In order to maintain the consistency of an acquaintance, we need to keep several elements in a consistent state, which include two acquainted peers, mapping tables and mapping expressions residing in both peers. All these elements of the acquaintance must be in a consistent state when we say that the acquaintance is consistent.

In this thesis, a transaction model is set up for P2P MDBSs after precise definitions are given to transactions and relevant concepts. Our model adjusts the structure and rules of the *open nested transaction model* [7] to fit in P2P MDBSs. We present serializability theories for global transactions when they are being propagated in a P2P MDBS. A sequence of global transactions that are initiated on a peer must preserve a consistent execution order in each peer where their child transactions arrive. A concurrency control protocol is developed based on the serializability theory under the assumption that transactions' propagation is acyclic in P2P MDBSs. The concurrency control protocol controls the consistent execution ordering of global transactions between two acquainted peers. Because of the hierarchical relationships between global transactions and their child transactions, global serializability is ensured by preserving the serializability over each acquaintance recursively.

In Chapter 2, we define transactions, introduce the open nested transaction model, and illustrate the detail how this transaction model fits the structure of P2P MDBSs. Then in Chapter 3, we analyze the consistency constraints, identify the proper correctness criteria for our setting, and infer the serializability theory. Based on the model and

theory developed in Chapter 2 and 3, Chapter 4 presents a concurrency control protocol along with an implementation. Chapter 4 also shows several experimental results. Finally, Chapter 5 concludes our thesis and indicates several directions for future work.

1.4 Related Work

This thesis is the first step to research the issue of transaction management in peer-to-peer multidatabase systems. As we mentioned before, P2P MDBSs are very close to the distributed and heterogeneous multidatabase systems on the architecture. MDBSs and P2P MDBSs are two approaches to solve the problem of data integration. Some other data integration models, such as the *Open Grid Service Architecture*[8] and *ORCHESTRA* [9], have also been developed over these years. All these models intend to solve the problem of data integration and sharing between preexisted database systems, each of which may be distributed, heterogeneous, and autonomous. Since these models utilize different approaches to achieve data integration, we are going to discuss these models, identify the similarity and difference between these database models, and see how the concurrency control are achieved in these models in this section.

1.4.1 Transaction Management in MDBSs

Global transactions and local transactions are both supported in MDBSs. Local transactions access data from a single database where they are submitted and global transactions access data from multiple local databases. A multidatabase management system is a software system built on top of a number of existing local database systems. The Global Transaction Manager(GTM) is a software module included in the MDBS and the GTM is responsible for the execution of transactions that may access data from multiple heterogeneous and autonomous local DBMSs. Because of the autonomy of each local DBMS, the local DBMSs are not aware of the existence of global transactions. Meanwhile, the GTM is also not aware of the existence of global transactions [2].

Based on the distribution of the data items among local databases, a global transaction G is decomposed by the GTM into multiple subtransactions G_i, each of which contain a subset of G's operations. G_i executes as a local transaction on the database system where it is submitted and only access data items located in the database. In MDBSs, it is assumed that each local DBMS ensures the ACID properties of (sub)transactions submitted to it, and the serializability of the local schedule which consists of operations from both local transactions and subtransactions of global transactions. By controlling the execution of global transactions, the GTM guarantees the ACID properties of each global transaction and ensures the global serializability of global transactions. After a subtransaction G_i of the global transaction G has been submitted to a local database system, it is out of the control of the GTM. Since local DBMSs may follow different concurrency control mechanisms, each local DBMS may execute the subtransactions from the same set of global transactions in various ordering. Thus the global serializability may be violated because of the autonomy and heterogeneity of local DBMSs. The concurrency control of global transactions becomes difficult to be achieved because of the splitting of control of global transactions and local transactions [7]. On the other hand, the indirect conflicts between global transactions created by local transactions on each local MDBS, also cause difficulties in ensuring global serializability in MDBSs. The atomicity of global transactions can be broken because local DBMSs can unilaterally abort a subtransaction any time before its commit.

The global serializability and the global atomicity and recovery are two main issues that are related to the transaction management in MDBSs. Lots of research have been done on this issue over the years. The following three approaches are mainly used to address the problems related to the transaction management in MDBSs.

- Some proposed protocols attempt to guarantee the multidatabase global serializability in MDBSs, such as protocols presented in [10] and [11].
- Relaxed correctness criteria, such as quasi-serializability in [12] and predicate-wise serializability in [13], are presented to replace the rigorous serializability in MDBSs.

- Redefined and extended transaction models, such as Open Nested Transaction Model in [7] and polytransactions in [14], have been created for MDBSs to fit the distributed and autonomous structure of MDBSs.

Since P2P MDBSs are closest to MDBSs in the architecture, these three approaches will be utilized to solve problems related to transaction management in P2P MDBSs in this thesis.

1.4.2 Other Related Work

The Open Grid Service Architecture(OGSA) [8] is a technology that enables the integration of services and resources across distributed, heterogeneous, dynamic virtual organizations. An approach for the OGSA that enforces globally serializable schedules in a distributed way is presented in [8]. This approach does not rely on any central coordinator with complete global knowledge of all peers in a Grid system. The concept 'transaction' is used to represent a transaction process for OGSA, which is composed of a set of service invocations on different peers which are executed in a specified order. Obviously, the 'transaction' in OGSA is not a traditional database transaction. Meanwhile, the notion of serializability is also utilized as the correctness criterion for OGSA to ensure the consistency of each transaction. In the approach presented in [8], each transaction manages its own serialization graph which comprises the service conflicts in which the transaction is involved in. The conflicts between service calls are always predefined by each peer. The serialization graph of each transaction is sufficient to decide whether the transaction is allowed to commit based on the commit rule: a transaction is not allowed to commit if it is dependent on another active transaction. This approach ensures the global serializability by enforcing the correctness of each transaction based on the knowledge of the transaction's serialization graph.

Both P2P MDBSs and the OGSA do not rely on any central control scheme. However, transactions in OGSA are different form the traditional database transactions and participating peers that provide services to other peers in OGSA may not be database

systems. A standard peer service is the composing unit of a transaction in OGSA instead of a data operation in a database transaction. P2P MDBSs not only provide users services in terms of queries, but also support the function of transactions in all peer databases. Each participating peer is a real database system and each transaction is an actual database transaction that is precisely defined with respect to the data items contained in a specific peer in P2P MDBSs. To ensure the consistency of a P2P MDBS, we not only need to control the correctness of each transaction, but also maintain the acquaintances' consistency, the serializability of global transactions, and even the consistency between indirectly acquainted peers.

ORCHESTRA is another data integration prototype, which focuses on managing disagreement among multiple data representations and instances, presented in [9]. It intends to create a collaborative data sharing infrastructure to facilitate scientific and academic data sharing in a bottom-up, rapid-to-change fashion. This prototype is still trying to provide solutions to the problem of data sharing and exchange in the perspective of data integration. Participating peers are tied together by shared relations that capture the same information about data, and instance mappings are established between each peer and the shared relation with that the peer is tied. Instance mappings are also used to relate different shared relations to each other. With the specific setting of multi-viewpoint tables, a set of transactions in an ORCHESTRA system can be flattened by reducing all intermediate update steps at each viewpoint of update publishing so as to minimize the conflicts between transactions. A sequence of transactions containing update operations are reduced to the minimal number of steps in a flattening operation. Under the assumption that all operations without data dependencies are independent, the results of an original update transaction sequence can be achieved by applying the flattened updates in three successive steps. The first step removes all tuples that are removed in the original update sequence; the second step modifies tuples that were modified in the original update sequence; the final step inserts new tuples that were created in the original update sequence [9]. Thus, the execution of transactions in ORCHESTRA systems can be easily controlled by applying the flattening operation to transactions. Although an ORCHESTRA system rapidly responds to the updates of underlying participating peers,

it just propagates updates operations at some pre-defined viewpoints. Thus an update is not propagated instantly as long as it is initiated at a specific peer. However, P2P MDBSs intend to process transactions in the same way as transactions are processed in centralized database systems. Transactions are being processed instantly and consecutively in the order that transactions are initiated in P2P MDBSs.

Comparing to all these data integration models, both conventional MDBSs and P2P MDBSs support the function of transactions in the same way as a centralized database system does. In OGSA, service calls are supported in a 'transaction process' instead of data operations. In ORCHESTRA systems, updating transactions are only propagated at some predefined view-points. Still we say that conventional MDBSs are the closest to P2P MDBSs on the architecture. However, MDBSs manage transactions in a global and central scheme, while P2P MDBSs manage transactions in a completely distributed way. Based on the serializability theories developed for MDBSs, we present our solutions to the problem of transaction management in this thesis.

Chapter 2

Transaction Model for Peer-to-peer Multidatabase Systems

A *transaction* is a sequence of read and write operations terminated by either a commit or an abort operation. In P2P MDBSs, each peer database management system is a software layer added on top of an underlying peer database. This layer, called the P2P layer, interoperates a local peer database with remote peer databases. A user submits transactions to the local peer with which he/she is interacting. From the point of view of users, all transactions are defined locally with respect to the single peer database. The peer database, where a transaction is initiated, is called the transaction's *home peer*. The underlying processing of the transaction is transparent to user after the transaction has been submitted. Generally, the user intends to consult all peers in the interesting group through the P2P network instead of consulting only the home peer database when he/she submits the transaction. Thus, all transactions in P2P MDBSs are submitted locally while they may be processed globally in the P2P network. After a transaction is submitted to its home peer, the PDBMS decides to execute it locally or globally based on the conformation of its acquaintances, which includes mapping tables, mapping expressions, and other consistency constraints. When the PDBMS decides to execute the transaction globally, it translates the transaction over each of its acquaintances and propagates the translated transaction to the corresponding acquainted peer accordingly. The

translated transaction is named as a *child transaction* of the global transaction and the global transaction is called the *parent transaction* of its child transactions. Formally, the parent transaction is the transaction that spawns child transactions to acquainted peers. According to our Assumption 2, a parent transaction spawns at most one child transaction to each acquainted peer when the spawning is possible over an acquaintance. The parent transaction may have more than one child transaction if a peer is acquainted to multiple peers. The peer, that the child transaction is defined for and spawned to, is the *destination peer* of the child transaction. After the destination peer receives the child transaction, it processes the child transaction as one of its normal local transactions and the child transaction may also be propagated further to other peers along the acquaintances in the P2P MDBS. Since acquaintances among peers are arbitrary, the global transaction may be propagated from peer to peer by following the propagation protocol. The child transactions and other translated transactions, which are propagated far away from the transaction's home peer, are *descendent transactions* of the transaction.

2.1 Transactions in P2P MDBSs

As we mentioned before, two types of transactions, local transactions and global transactions, are supported in conventional MDBSs. Because P2P MDBSs are similar to MDBSs in the structure, we also distinguish a transaction in P2P MDBSs as either a local transaction or a global transaction according to where the transaction is from. Since there is no central control scheme, global transactions are not in control of any global management system in P2P MDBSs.

Definition 1 *A* Local transaction *of a peer is a transaction which is submitted to the peer by a user, not by other acquainted peers.*

Definition 2 *A* Global transaction *is initiated on a single peer database and its descendent transactions are propagated to multiple local peer databases and executed on these peers. After the transaction is initiated on its home peer, its child transactions are spawned to other acquainted peers over acquaintances by following the spawning protocol.*

Every child transaction may also spawn its own child transactions to acquainted peers further in the group.

A transaction T is initiated as a local transaction with respect to its home peer. But it is also a global transaction as long as its child transaction(s) are spawned out from its home peer. T is a global transaction from the point of views of those acquainted peers to which T's child transactions are spawned.

A transaction T_p initiated on its home peer p, is called the *root transaction* of the global transaction T. All transactions, which are translations of T and propagated to other peers along acquaintances, are descendent transactions of T. Child transactions are descendent transactions spawned to directly acquainted peers.

The transaction examples: T_1, T_2, and T_3 presented in the motivating example in Chapter 1 are all global transactions since their child transactions are spawned to their acquainted peers, respectively. Let's see how a global transaction is propagated and processed in a P2P MDBS after it has been initiated on its home peer. We use the DGL P2P MDBS as our example.

Example 2.1.1 *Let's use the transaction T_1 in the motivating example as a global transaction in the Digital Library P2P MDBS.*

```
T1: begin
      read(OTT_Book.title, X);
      if(X = Principles of transaction processing) {
            read(OTT_Book.addr, Y);
      }
      return Y;
    end
```

After T_1 is initiated on p_1, the transaction management (TM) system in the P2P layer of p_1 will check p_1's acquaintances one by one. For each acquaintance, the PDBMS goes

through all the mapping tables and mapping expressions, and decides if the transaction T_1 can be translated over the acquaintance. If T_1 can be translated, the PDBMS translates T_1 into a child transaction, which is defined with respect to the destination peer, and spawns the child transaction to the destination peer. Since both mapping tables for the book title and download address are identity mapping tables, the data items title and download address, that T_1 accesses, are mapped over all acquaintances of p_1. For example, over the acquaintance between $p_1 \leftrightarrow p_2$, TM discovers that *OTT.title* is mapped to *NY.title* in the mapping table *title2title* by checking mapping tables between p_1 and p_2. The book Principles of transaction processing is mapped to another book with the identical book title held by p_2. Thus, TM translates T_1 into a child transaction $T_{1(1\rightarrow 2)}$, and spawn $T_{1(1\rightarrow 2)}$ to p_2. Although $T_{1(1\rightarrow 2)}$ is a child transaction of T_1, $T_{1(1\rightarrow 2)}$ is a global transaction to p_2 when p_2 receives it. p_2 processes $T_{1(1\rightarrow 2)}$ as a local transaction submitted to it. By following the same rules and procedures, another child transaction $T_{1(1\rightarrow 3)}$ is generated over the acquaintance $p_1 \leftrightarrow p_3$ and spawned from p_2 to p_3. The detail of child transactions $T_{1(1\rightarrow 2)}$ and $T_{1(1\rightarrow 3)}$ are shown as follows:

```
T1(1→2): begin
             read(NY_Bookl.title, X);
             if(X = Principles of transaction processing){
                 read( NY_Books.link, Y);
             }
             return Y;
         end

T1(1→3): begin
             read( TOR_Book.title, X);
             if(X = Principles of transaction processing){
                 read(OTT_Book.address, Y);
             }
             return Y;
         end
```

T_1's two child transactions: $T_{1(1\to2)}$ and $T_{1(1\to3)}$ are siblings to each other since they are from the same parent transaction T_1. After $T_{1(1\to2)}$ and $T_{1(1\to3)}$ have been submitted to p_2 and p_3 respectively, the Transaction Manager in p_2 and p_3 will process these two transactions in the same procedures as p_1 processes T_1. Thus, the child transactions of $T_{1(1\to2)}$ and $T_{1(1\to3)}$ may be spawned further to other acquainted peers of p_2 and p_3 respectively, except for the home peer p_1. According to the acquaintances shown in Figure 1.1, p_2 is acquainted to p_1, p_3, and p_4. Therefore, $T_{1(1\to2)}$'s child transactions may also be spawned to p_2 and p_4. Since p_3 is acquainted to p_1 and p_2, $T_{1(1\to3)}$'s child transactions may be spawned to p_2. Suppose T_1's descendent transactions is propagated to every peer in the *DGL* network. Let $T_{1(1\to2\to3)}$ be the child transaction of $T_{1(1\to2)}$ that is spawned from p_2 to p_3 and $T_{1(1\to3\to2)}$ be the child transaction of $T_{1(1\to3)}$ that is spawned from p_3 to p_2. We define T_1 as a compound transaction which consists of T_1 itself and all its descendent transactions: $T_{1(1\to2)}$, $T_{1(1\to3)}$, $T_{1(1\to2\to3)}$, and $T_{1(1\to3\to2)}$. T_1 is the root transaction of this compound transaction and each transaction in T_1 is a component transaction of the compound transaction T_1.

$$T_1 = \{T_1, T_{1(1\to2)}, T_{1(1\to3)}, T_{1(1\to2\to3)}, T_{1(1\to3\to2)}\}$$

The child transactions $T_{1(1\to2\to3)}$ and $T_{1(1\to3)}$ are two descendent transactions of the same root transaction T_1 which are located on the same peer p_3. As we discussed before, a transaction may be translated completely and partially depending on the mapping information over an acquaintance. $T_{1(1\to2\to3)}$ and $T_{1(1\to3)}$ are two identical transactions when both of them are complete translations of T_1, while $T_{1(1\to2\to3)}$ and $T_{1(1\to3)}$ may be different if they are partial translations of T_1. Whether these two child transactions are identical or not, we need some mechanisms to deal with this issue. Since this is out of the scope of the thesis, we leave it as future work.

2.2 Transaction Translation

In example 2.1.1, we showed a global transaction example T_1 and its child transaction $T_{1(1\to2)}$ and $T_{1(1\to3)}$. T_1 is the parent transaction of $T_{1(1\to2)}$ and $T_{1(1\to3)}$. Both child transactions are translations of their parent T_1.

A transaction is composed of a set of read and write operations, and some arithmetic operations. To translate the transaction, we need to translate every operation contained in the transaction. An operation is always translated into the same operation, with its data items being substituted with their mapping data items. When translating the operation, we keep the same operator and substitute the operands by their corresponding mapping data items. The operation is *translatable to a destination peer* if each of its operands is being mapped in the mapping tables over the acquaintance with the destination peer. Because not all relations and attributes, which are included in two acquainted peers, can be mapped or are necessary to be mapped over the acquaintance, some of the data items accessed by a global transaction may not be mapped over the acquaintance. Thus the global transaction may be translated either partially or completely over the acquaintance. When data items accessed by the global transaction are fully mapped to a destination peer, all operations of the global transaction are translatable to the destination peer and a child transaction which is a complete translation of the global transaction will be spawned to the destination peer. When data items accessed by the transaction are partially mapped to the destination peer, then only those operations with their data being mapped are translatable to the destination peer, then a child transaction, which is a partial translation of the global transaction, will be generated and spawned to the destination peer.

Definition 3 *A child transaction to a destination peer is a* complete translation *of its parent transaction if all operations included in the parent transaction are translatable to the destination peer. In this case, the parent transaction is* completely translatable *to the destination peer.*

Definition 4 *A child transaction is a* partial translation *of its parent transaction if only a subset of operations included in the parent transaction is translatable to the destination peer. We also say that the parent transaction is* partially translatable *to the destination peer.*

Example 2.2.1 *Let G_1 be a global transaction initiated in p_1 in the DGL P2P MDBS presented in our motivating example.*

$$G_1 : r(a)w(b)$$

Suppose data items a and b accessed by G_1 are mapped to a' *and* b' *over the acquaintance $p_1 \leftrightarrow p_2$, respectively; data item* a *is mapped to* a" *over the acquaintance $p_1 \leftrightarrow p_3$. Then G_1 is translated into $G_{1(1\rightarrow 2)}$ and $G_{1(1\rightarrow 3)}$ over p_2 and p_3, respectively.*

$$G_{1(1\rightarrow 2)} : r(a')w(b')$$
$$G_{1(1\rightarrow 3)} : r(a")$$

$G_{1(1\rightarrow 2)}$ is a complete translation of G_1 since both operations in G_1 are translatable to p_2. However, $G_{1(1\rightarrow 3)}$ is a partial translation of G_1 because only one operation $r(a)$ is translatable to p_3. G_1 can be expressed in a layered structure shown in Figure 2.1.

Proposition 1 *Let peer p_1 be acquainted to peer p_2 in a P2P MDBS. If all attributes accessed by a transaction T on p_1 are being mapped in identity mapping tables over the acquaintance between p_1 and p_2, then the child transaction $T_{(1\rightarrow 2)}$ of T is a complete translation of* T *over the acquaintance $p_1 \leftrightarrow p_2$.*

```
G1: begin
      begin
        r(a)
        w(b)
      end
        G1(1→2): begin
                   r (a')
                   w (b')
                 end
        G1(1→3): begin
                   r (a")
                 end
      end
```

Figure 2.1: Layered structure of G_1

Proof:Because all attributes accessed by T are being mapped in identity mapping tables, every attribute can be substituted by a corresponding variable which is defined with the acquainted peer when T is being translated. Therefore, the child transaction $T_{(1\rightarrow 2)}$ of T is a complete translation of T over the acquaintance $p_1 \leftrightarrow p_2$.

2.3 Open Nested Transaction Model for Peer-to-peer Multidatabase Systems

Recall that a global transaction is composed of a collection of component transactions, which include the root transaction and all its descendent transactions. Some of the descendent transactions, which are so called child transactions, are generated directly from the root transaction, while others are generated from its descendent transactions. We need to figure out the relationships between all the component transactions contained in the global transaction, such as the relationship between the parent and its child transactions, and the relationship between siblings, etc. A proper transaction model is needed for P2P MDBSs depending on how component transactions are related to each other in

a global transaction.

The Open Nested Transaction Model presented in [7] is an extended model of the nested transaction model presented by Moss (1981). A nested transaction is composed of a collection of subtransactions or child transactions [1]. A subtransaction may also contain any number of subtransactions. In the open nested transaction model, each component transaction may decide to commit or abort unilaterally. Each of these component transactions T is compensatable, which means that each component transaction T has a compensating transaction $comp\_T$ that semantically undoes the effects of T if it is necessary[7]. The open nested transaction model was designed to be particularly suitable in structuring applications that need to access data stored in pre-existing databases which are managed by systems that do not support any global commit protocol[7]. Therefore, the Open Nested Transaction Model meets the basic needs of P2P MDBSs since a P2P MDBS is exactly such a database application that consists of a collection of autonomous and heterogeneous peer databases, each of which is preexisting. On the other hand, there is no global transaction management scheme in P2P MDBSs. In the following sections, we discuss how to extend the open nested transaction model properly and make it fit in P2P MDBSs.

2.3.1 Structure of a Global Transaction

Let a peer p be acquainted to n ($n > 0$) peers p_1, p_2, ..., and p_n in a P2P MDBS. A transaction T is initiated on the peer p. Suppose that p spawns a child transaction of T to each of p's acquainted peers $p_{i(1 \to n)}$. Then T can be expressed as a two-layer structure shown in Figure 2.2. T is a compound transaction composed of a parent transaction and its child transactions.T is a compound transaction composed of a parent transaction and its child transactions. In this two-layer structure of T, T is located at the layer which is on the top of its child transactions and the child transactions are included in the next lower layer of the parent transaction.

$$
\begin{aligned}
\mathrm{T} = \{ & \\
& \mathrm{T} \\
& \quad T_{(p \to p_1)} \\
& \quad \cdots \\
& \quad T_{(p \to p_i)} \\
& \quad \cdots \\
& \quad T_{(p \to p_n)} \\
\} &
\end{aligned}
$$

Figure 2.2: Two-layer structure of the Global Transaction T

$$T = \{T, T_{(p \to p_1)}, \ldots, T_{(p \to p_i)}, \ldots, T_{(p \to p_n)}\} \tag{2.1}$$

Because acquaintances are arbitrary in P2P MDBSs, each child transaction may also spawn its own child transactions again. Those child transactions that propagate child transactions further are also nested transactions. Therefore, the global transaction may have multiple transaction layers depending on the distance of the global transaction's propagation from its home peer.

We can use a tree to express the structure of T. We call the tree a *global transaction tree* of T. T is the root of the tree and all other component transactions are the nodes of the tree. The tree does not need to be balanced. We define a *propagation level* to facilitate the global transaction's structuring. The propagation level i of a component transaction T is the length of the path along which the global transaction is propagated from the home peer to the destination peer. We use L_i to represent the propagation level i. The component transactions of T may locate in different propagation levels. The propagation level of the root T is L_0. The component transactions, which are located on the propagation level L_1, are the child transactions of T. The child transactions of T are located in those peers that are directly acquainted to T's home peer. Those descendent transactions other than T and its child transactions are located on peers that are not directly acquainted to T's home peer. Then T can be expressed as a global transaction

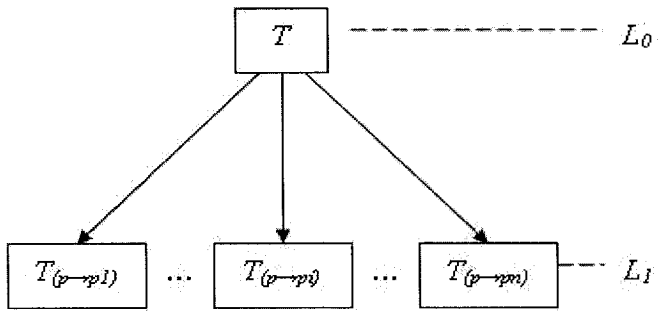

Figure 2.3: Global transaction tree of T

tree shown in Figure 2.3.

Example 2.3.1 *Let* T *be a global transaction in the digital library P2P MDBS and* T *is initiated on* p_1. *Suppose that* T*'s children* $T_{(1\rightarrow 2)}$ *and* $T_{(1\rightarrow 3)}$, *are spawned to* p_2 *and* p_3, *respectively. Then* p_3 *spawns* $T_{(1\rightarrow 3\rightarrow 2)}$ *to* p_2. p_2 *spawns* $T_{(1\rightarrow 2)}$*'s child transactions* $T_{(1\rightarrow 2\rightarrow 3)}$ *and* $T_{(1\rightarrow 2\rightarrow 4)}$ *to* p_3 *and* p_4, *respectively. Another child transaction* $T_{(1\rightarrow 3\rightarrow 2\rightarrow 4)}$ *of* $T_{(1\rightarrow 3\rightarrow 2)}$ *is spawned to* p_4 *as well. Figure 2.4 shows the detail of* T*'s propagation in the DGL P2P MDBS. Figure 2.5 and 2.6 show the structure of* T *in the form of a layered structure, and a global transaction tree, respectively.*

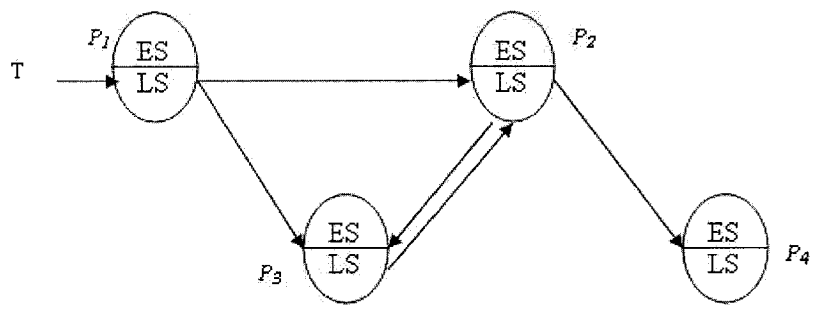

Figure 2.4: Global transaction tree of T

$$T = \{T, T_{(1\to2)}, T_{(1\to3)}, T_{(1\to2\to3)}, T_{(1\to2\to4)}, T_{(1\to3\to2)}, T_{(1\to3\to2\to4)}\}$$

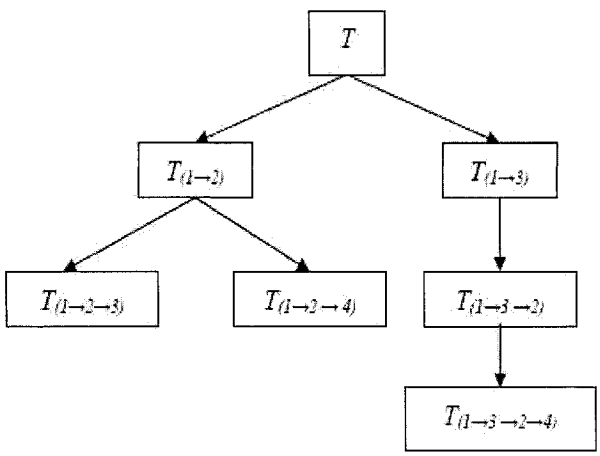

Figure 2.5: Global transaction tree of T

T = {
T
$T_{(1\to2)}$
$T_{(1\to2\to3)}$
$T_{(1\to2\to4)}$
$T_{(1\to3)}$
$T_{(1\to3\to2)}$
$T_{(1\to3\to2\to4)}$
}

Figure 2.6: Global Transaction T's Layer Structure

From both the layered-structure and the global transaction tree of T, we see that T's component transactions locate on four different propagation levels: T on L_0, $T_{(1\to2)}$ and

$T_{(1\to3)}$ on L_1, $T_{(1\to2\to3)}$, $T_{(1\to2\to4)}$, and $T_{(1\to3\to2)}$ on L_2, while $T_{(1\to3\to2\to4)}$ on L_3. Component transactions located on leaves will not propagate further to other peers.

2.3.2 Independence of Component Transactions

From the definition of the open nested transaction model, we already know that each component transaction of an open nested transaction decides to commitabort unilaterally. However, when all component transactions are accommodated in one global transaction, we still need to find out how these component transactions are related to each other and how they affect one another during their processing.

In a conventional MDBS, a global transaction is divided into a number of subtransactions and each subtransaction contains one or more operations from the original global transaction. The atomicity of the global transaction is one of the key issues for concurrency control in MDBSs. Some protocol such as the *Two Phase Commit Protocol* [15] is used to ensure the atomicity of global transactions. On the other hand, it is assumed that each value written by a write operation depends on the values of all the data items that the transaction previously read In a transaction [15]. Thus subtransactions of the global transaction may depend on each other on values.

Example 2.3.2 *Suppose a MDBS consists of two local database management systems* S_1 *and* S_2. S_1 *has data* a, *and* S_2 *has data* b *and* c. *Let* G_1 *be a global transaction in this MDBS:*

$$G_1 : r(a)r(b)w(c)w(a)$$

Then G_1 *is divided into three subtransactions* G_{111}, G_{121}, *and* G_{112} *with respect to the two sites:* S_1 *and* S_2. *Each of them contains a subset of operations included in* G_1.

$$G_{111} : r(a)$$
$$G_{121} : r(b)w(c)$$
$$G_{112} : w(a)$$

From the operations contained in these three subtransactions G_{111}, G_{121}, and G_{112}, we know that these three subtransactions depend on each other on values. On the other hand, G_{111} and G_{112} are two subtransactions submitted to S_1. Because they conflict to each other, they must be executed in the order G_{111} G_{112} in site S_1 so as to preserve their order in G_1 and ensure the consistency of G_1. Each subtransaction is an execution component of G_1. If anyone of the subtransactions is aborted, G_1 has to abort.

Similarly, in P2P MDBSs, a global transaction has its own child transactions when its translations are spawned to other acquainted peers. However, we call a spawned transaction as a child transaction in P2P MDBSs instead of a subtransaction. A global transaction is translated into child transactions when spawning is possible over an acquaintance. As we defined before, there are two cases regarding the translation of the global transaction: complete translation and partial translation.

Example 2.3.3 *Let G_2 be a global transaction initiated on p_1 in DGL P2P MDBS.*

$$G2 : r(a)w(b)w(c)$$

Suppose all data items a *and* b *in the access data set of G_2 are both mapped to data items in p_2 and p_3, respectively. Over the acquaintance $p_1 \leftrightarrow p_2$,* a *is mapped to* a', *and* b *is mapped to* b'*; over the acquaintance $p_1 \leftrightarrow p_3$,* a *is mapped to* a", *and* b *is mapped to* b". *Then G_2 is translated into $G_{2(1\rightarrow 2)}$ over the acquaintance $p_1 \leftrightarrow p_2$ and $G_{2(1\rightarrow 2)}$ is a complete translation of G_2. Meanwhile, G_2 is also translated into $G2(1\rightarrow 3)$ over the acquaintance $p_1 \leftrightarrow p_3$ and $G_{2(1\rightarrow 3)}$ is a complete translation of G_2. $G_{2(1\rightarrow 2)}$ and $G_{2(1\rightarrow 3)}$ are spawned to p_2 and p_3, respectively. Therefore, G_2 can be expressed in a layered structure shown in Figure 2.7.*

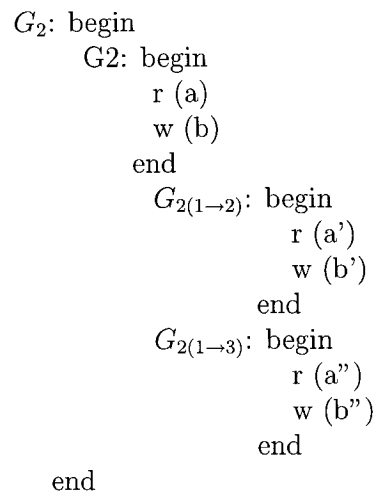

Figure 2.7: Layered structure of G_2

In Example 2.3.3, we see that G_2 is a compound global transaction with three component transactions: G_2, $G_{2(1\rightarrow 2)}$, and $G_{2(1\rightarrow 3)}$. G_2 acts as the root transaction. G_2 is executed on p_1 independently since G_2 only accesses data items stored in p_1 and it does not need any value or intermediate result from any of its child transactions. Its two child transactions $G_{2(1\rightarrow 2)}$ and $G_{2(1\rightarrow 3)}$ contain the same operations as G_2. $G_{2(1\rightarrow 2)}$ only accesses data from p_2, while $G_{2(1\rightarrow 3)}$ only operates data from p_3. After p_2 receives $G_{2(1\rightarrow 2)}$, p_2 processes $G_{2(1\rightarrow 2)}$ independently as a local transaction; so does p_3 processes $G_{2(1\rightarrow 3)}$. The executions of $G_{2(1\rightarrow 2)}$ and $G_{2(1\rightarrow 3)}$ do not depend on any value returned from their parent transaction G_2. Meanwhile, $G_{2(1\rightarrow 2)}$ and $G_{2(1\rightarrow 3)}$ do not depend on each other on values as well when they are being processed on the peers to which they are spawned, respectively. Therefore, each component transaction is an atomic and completed transaction on the peer where it is submitted.

Example 2.3.4 *Let G_3 be a global transaction initiated on p_1 in DGL P2P MDBS.*

$$G_3 : r(a)w(b)w(c)$$

Suppose data items in the access data set of G_3 *are only partially mapped to* p_2 *and* p_3*, respectively. Data item* a *is mapped to* a', *and* b *is mapped to* b' *over the acquaintance* $p_1 \leftrightarrow p_2$*, while* c *is mapped to* c" *over the acquaintance* $p_1 \leftrightarrow p_3$*.* G_3 *is translated into* $G_{3(1\rightarrow 2)}$ *over the acquaintance* $p_1 \leftrightarrow p_2$*, and* $G_{3(1\rightarrow 3)}$ *over the acquaintance* $p_1 \leftrightarrow p_3$*. However,* $G_{3(1\rightarrow 2)}$ *only contains two operation* $r(a')$ *and* $w(b')$ *which are translated from* G_3*, and* $G_{3(1\rightarrow 3)}$ *contains only one translated operation* $w(c")$*. Both* $G_{3(1\rightarrow 2)}$ *and* $G_{3(1\rightarrow 3)}$ *are partial translations of* G_3*.* G_3 *can be described in the structure shown in Figure 2.8.*

```
G3: begin
      G3: begin
            r (a)
            w (b)
            w (c)
          end
            G3(1→2): begin
                       r (a')
                       w (b')
                     end
            G3(1→3): begin
                       w (c")
                     end
end
```

Figure 2.8: Layered structure of G_3

In Example 2.3.4, $G_{3(1\rightarrow 2)}$ and $G_{3(1\rightarrow 3)}$ contains only operations that are translated from a subset of G_3's operations. G_3 is the root transaction. Each component transaction is defined with respect to the peer to which it is submitted. G_3 only accesses data items located in its home peer p_1; so does $G_{3(1\rightarrow 2)}$ on p_2 and $G_{3(1\rightarrow 3)}$ on p_3. Although child transactions are partial translations of the root transaction in Example 2.3.4, we can also observe that component transactions in a global transaction are independent to each other. No component transaction needs any value or intermediate result from any

other component transaction in G_3.

Overall, component transactions included in a global transaction is independently located and processed on their peers. Whether a child transaction is a complete translation or a partial translation of its parent, the child transaction is spawned to the destination peer as an atomic transaction. Since all peers are fully autonomous, the parent and its child transactions are executed independently on the peers to whom they belong. The execution of the parent transaction does not need any value or result returned from any of its child transactions, and vice versa. Sibling transactions may be different from each other because they may consist of operations which are translated from various subsets of the parent transaction's operations. Peers who receive sibling transactions from the same parent transaction might not be acquainted to each other. Sibling transactions do not depend on each other during their execution. Since a global transaction is always propagated from peer to peer in a parent-to-child way, all descendent transactions are independent to each other in a P2P MDBS. Each peer processes the component transaction that it receives on its own. A component transaction accesses only data items that are located in the peer to which the component transaction is submitted.

Therefore, the execution of a component transaction of a global transaction is not dependent on any value or result returned from any other component transaction. There exists no value dependency among component transactions of the global transaction. The component transactions of the same global transaction are only loosely coupled together in P2P MDBSs. Because the component transactions are independent to one another, we can simplify the relationships between component transactions and relax the database correctness criteria in P2P MDBSs.

2.3.3 Commit/Abort Rules of Open Nested Transactions in P2P MDBSs

In P2P MDBSs, each component transaction is independent and there is no value dependency between any two of the component transactions in a global transaction. Based on this observation, we discuss the commitment rules of the global transaction when the global transaction fits in the open nested transaction model in P2P MDBSs.

When a peer database system processes a transaction, some transaction management primitives are also invoked besides the read and write operations. An invocation of a transaction management primitive is called as a *significant event* [7]. In centralized database systems, transactions are generally associated with three transaction management primitives: **Begin**, **Commit**, and **Abort**. **Spawn** is another transaction management primitive found in the nested transaction model [7]. Thus, these four transaction management primitives: Begin, Spawn, Commit, and Abort are all included in P2P MDBSs.

Global transactions in P2P MDBSs are translated into child transactions to acquainted peers based on the mapping information. Thus, a child transaction not only inherits the operations from its parent transaction, but also the final action(commit \abort) of its parent. If the parent transaction commits, then its child transactions inherit the commit action and commit in the peers where they are located. On the other hand, child transactions must abort if the parent transaction aborts. Eventually, the global transaction's action is hierarchically inherited by all its descendent transactions from peer to peer along acquaintances in the parent-to-child way. A descendent transaction's action, whether it commits or aborts on its own site, does not affect the action of the root transaction. However, if a descendent transaction is also a nested transaction, the descendent transaction is the root of a sub-tree of the global transaction and the descendent transaction is in control of all its own descendent transactions. The sub-tree in the global transaction must follow the commitabort rules as well.

The commitabort rules of open nested transactions in P2P MDBSs are as follows:

T_o denotes the root transaction. $Parent(T_o) = \varnothing$.
T_c denotes a child transaction of T_p. $Parent(T_c) = T_p$.
T denotes an arbitrary component transaction.

1. Significant events associated with T_o are *Begin*, *Commit*, and *Abort*.

2. Significant events associated with a component transaction T other than the root transaction T_o are *Spawn*, *Commit*, and *Abort*.

3. **Abort Rule**: All the children T_c must abort if the parent T_p aborts. On the other hand, the abortion of the children will not cause the abortion of its parent.

4. **Commit Rule**: A parent T_p decides the commitment of all its children. However, the commitment of the parent is not dependant to the commitments of its children.

Chapter 3

The Model: Serializability Theory for Peer-to-peer Multidatabase Systems

After a transaction is initiated in its home peer, the PDBMS of the home peer processes it automatically. The processing of the transaction is transparent to the user who submits it. P2P MDBSs may contain different number of peer databases. Then a large amount of transactions may be submitted to a P2P MDBS. When global transactions are being propagated in the P2P MDBS, each participating peer needs to deal with both local transactions submitted by its users and global transactions spawned from other acquainted peers. Since we already assumed that each participating peer database ensures its own local serializability, some concurrency control mechanism is needed for each peer database to reconcile with other peers so as to ensure the consistency of global transactions. Since there is no global management system in P2P MDBSs, the currency control mechanism is implemented in the PDBMS of each peer database.

3.1 Database Consistency Constraints

Conventional MDBSs are founded on the concepts of global schemas, global access to multiple databases, and global transaction management mechanisms. Since global transactions are under control of the global transaction management system, the global transactions' serializability must be enforced to maintain the consistency of MDBSs. P2P MDBSs are similar to conventional multidatabase systems on the architecture. However, the total absence of any global transaction management scheme forces us to relax known consistency constraints in P2P MDBSs.

As we mentioned before, P2P MDBS can be used in many business groups, such as airline, genomic, hospital, and university databases, etc. Peers in these groups intend to achieve data exchange and sharing by following some coordination rules. The mapping tables and mapping expressions on each acquaintance constrain the data sharing and exchanging between two acquainted peers. We now consider some P2P MDBS applications.

Example 3.1.1 *In the Digital Library P2P MDBS, four digital libraries* p_1, p_2, p_3, *and* p_4 *establish a P2P MDBS to share their online resources. Users submit all kinds of transactions to the P2P network. For example, when a user submits a transaction to search for a book with the title* Fundamentals of database system *to* p_1, *the user would expect a result to show the addresses and rates to download the book. It would be perfect if the P2P MDBS could propagate the transaction to every peer in the group and every peer who has the book returns an answer to the user. In the case that the digital library contains hundreds of peer databases or more, the returned result would include a large number of answers from different peers. However, the user's objective is to find one proper address and rate to download the book. So it would be difficult for the user to pick one proper answer from a huge result. Meanwhile, it takes long time to process the transaction in a large scale of P2P MDBS. This long transaction processing from the P2P system is not desired by the user. In reality, although a thorough search to all peers might be desired by*

users, an incomplete answer from the P2P system, which consists of results from some of the peers, would also be acceptable and adequate for users. Therefore, a sound result, which contains part of the correct answers to a global transaction, is acceptable in P2P MDBSs.

Example 3.1.2 *Let global transactions to update books be initiated in two acquainted peer databases: p_1 and p_2, in the DGL P2P MDBS. The following mapping expressions are created when an acquaintance is being established.*

$$\textbf{\textit{ME1}} : OTT\_Book(call\#, location, status) \supseteq NY\_Book(callNo, location, status) \quad (3.1)$$

$$\textbf{\textit{ME2}} : OTT\_Book(call\#, location, status) \subseteq NY\_Book(callNo, location, status) \quad (3.2)$$

The mapping expression ME1 and ME2 together mean that, in the context of data exchange, any book of NY_DGL_DB is considered a book of OTT_DGL_DB, and vice versa [4]. Therefore a new tuple should be inserted into NY_Book accordingly whenever a new book is entered in OTT_Book. In some situations when the peer leaves the system temporarily, the book insertion into NY_Book may not be able to be processed at the same time when the new book is entered in the OTT_Book. p_1 and p_2 are not consistent before the transaction to insert the book has been processed in p_2. Two libraries p_1 and p_2 may negotiate to check the books' updating once a day. If p_1 finishes processing the book insertion transactions spawned from p_2 in a day, the consistency of the acquaintance is not violated, and vice versa for transactions spawned from p_1 to p_2.

From Example 3.1.2, we see that it is allowed that the transaction to update books is delayed by some time duration according to the negotiated constraints of two acquainted peers. On the other hand, to delete a book in the relation OTT_Book in p_1 may require corresponding deletions in both the mapping tables and the mapping relation NY_Book in p_2 in 2 hours. When a book is deleted in the peer who owns it, other peers in the P2P system should not return the book's information to users any more. This example shows

that two peers can tolerate data inconsistency in some allowed time duration. However, the temporal data inconsistency that the system can tolerate varies because different transactions have different constraints. In the first situation, the system can tolerate the data inconsistency for up to a day, whereas in the latter case, the system may only tolerate the data inconsistency for 2 hours.

Some other P2P MDBSs mentioned above, such as airline, genomic, hospital, and university databases, can also tolerate different levels of temporal data inconsistency.

Example 3.1.3 *A medical P2P MDBS consists of hospitals, family doctors, pharmacists, and pharmaceutical companies. Participating peers intend to share and exchange patient information about medical histories, medication, symptoms, treatments and the like. A family doctor will send a patient's medical history to a hospital when a patient needs to see a specialist in the hospital. During the period when the patient is being treated in the hospital, the specialist and the hospital will add the treatment information to the medical history of the patient. However, the added information may not be sent to the family doctor as soon as it is added by the specialist. In general, after the patient finishes his/her treatment in the hospital, a copy of his/her medical history will be sent back to his/her family doctor for a continuous observation. From the flow of a patient's medical history, we can see that medical history of the patient is not transmitted between his/her family doctor and hospital at pre-specified time intervals. Although both the doctor and the hospital have the medical history of the patient, they may be different at some time. Basically the medical history of a patient is required to be consistent in the medical group. However, the flow of the patient's medical history shows that the system can still tolerate inconsistency in some situations.*

Example 3.1.1 - 3.1.3 show that some inconsistency can be tolerated in P2P MDBSs like digital library group and medical group. We can assume that each interest group can tolerate some data inconsistency when their participating peers are connected in P2P

MDBSs. The inconsistency being tolerated for each P2P MDBS is application domain dependent. Unlike centralized distributed databases that require data consistency to be maintained after every transaction, P2P MDBSs do not. We can say that a relaxed database consistency model can be achieved for each specific P2P interest group according to their business conventions.

3.2 Database Correctness Criteria

Recall that a global transaction is a compound transaction containing a bunch of component transactions. In order to simplify the complexity of concurrency control, we assumed that a parent transaction spawns only one child transaction to an acquainted peer when a spawning is possible over an acquaintance with that peer.

We take the serializability as the consistency criteria in P2P MDBSs. To solve the problem of concurrency control of global transactions in P2P MDBSs, we need to find out the serializability which can be enforced in P2P MDBSs to ensure the consistency of the system.

3.2.1 Consistency of a Global Transaction

As we discussed before, a global transaction is a compound transaction which consists of itself and all its descendent transactions. The root of the global transaction decides to commit or abort. When the root transaction aborts, all its descendent transactions must abort; when the global transaction commits, all its descendent transactions are supposed to commit as well. However, in some cases, a descendent transaction does not commit after the root of the global transaction commits. One of such cases is that a peer leaves the P2P MDBS after the peer receives a descendent transaction and has not committed the descendent transaction yet. Because acquaintances in P2P MDBSs are transient, peers can join or quit the network at any time. Nevertheless, the action of the

descendent transaction does not affect the commitment of the global transaction.

A global transaction is always propagated from peer to peer along acquaintances. Child transactions inherit the actions of their parent transactions. A descendent transaction of the global transaction inherits the action of the root transaction from its own parent transaction, which may also be a descendent transaction of the root transaction. In the transaction tree of the global transaction, the action of the root transaction is inherited from peer to peer along the path that the global transaction is propagated from top to down. Therefore, the consistency of the global transaction can be ensured hierarchically and recursively from acquaintance to acquaintance in a top-down way as long as the consistency of the global transaction on each acquaintance can be guaranteed. The consistency of the global transaction can be decomposed into the consistency over each acquaintance that is included in the propagation of the global transaction. What we need is a protocol which ensures that the action of the parent transaction can be inherited by its child transaction over an acquaintance. We say the consistency of the global transaction is preserved if it is ensured that each child transaction inherits the action of its parent transaction.

Example 3.2.1 T *is a global transaction initiated in* p_1 *in the DGL P2P MDBS in our motivating example.* T*'s component transactions and propagation details is shown in Figure 3.1.*

T is the root of the global transaction. p_1 is acquainted to p_2 and p_3. T is connected with its child transactions $T_{(1\rightarrow 2)}$ and $T_{(1\rightarrow 3)}$, while it has no direct connection with other descendent transactions. If T commits, T broadcasts a commit message to its child transactions: $T_{(1\rightarrow 2)}$ and $T_{(1\rightarrow 3)}$. Then $T_{(1\rightarrow 2)}$ broadcasts the commit message to its own child transactions: $T_{(1\rightarrow 2\rightarrow 3)}$ and $T_{(1\rightarrow 2\rightarrow 4)}$, and so does $T_{(1\rightarrow 3)}$. So T's commit message is propagated to all its component transactions in the way from parent to child transactions until the leaves. So T's commitment is achieved hierarchically from peer to peer along the propagation path in a top-down way.

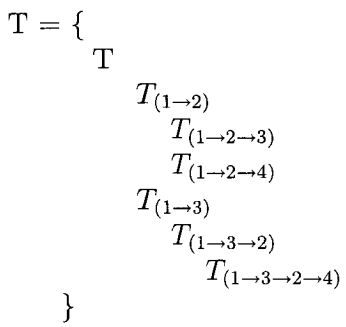

Figure 3.1: The Structure of the global transaction T

3.2.2 Serializability Theory

A P2P multidatabase system needs not only to ensure the consistency of a single global transaction, but also to maintain the consistency of the system when multiple transactions are submitted to the P2P MDBS concurrently.

In conventional MDBSs, all global transactions are submitted to a global transaction management system since global transactions are under control of the global transaction management system. The global transaction management system maintains the consistency of the MDBS by ensuring the global transactions' serializability. However, unlike conventional MDBSs, there is no global control system in P2P MDBSs. Users submit transactions to a peer with which they interact. The peer acts as a global transaction management system which manages the transactions submitted to it. The peer needs not only to ensure the consistency of the peer itself, but also to reconcile with other acquainted peers to maintain the consistency of the global transactions that it sends out and receives.

Quasi Serializability [16] is presented as a relaxed correctness criterion for heterogeneous distributed database environments. In order to achieve the relaxation of serializability, three assumptions have been made in [16].

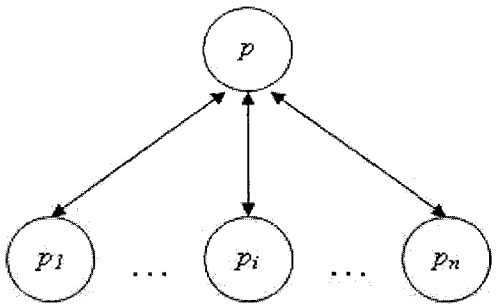

Figure 3.2: The peer p and its acquainted peers

1. A global transaction is allowed to submit only one subtransaction to each local site; and

2. There is no data integrity constraint on data items at different sites; and

3. There is no value dependency between subtransactions of the same global transactions.

As we discussed before, these three assumptions are also followed in P2P MDBSs. Therefore, we take quasi-serializability as the correctness criterion on each acquaintance in P2P MDBSs. We still need to adjust and extend the quasi-serializability to fit in our P2P MDBS model.

Let a peer p be acquainted to multiple peers p_1, p_2, ..., and p_n. Then p is in charge of all transactions processed on it and these transactions' child transactions that p spawns to its acquainted peers. p acts as the global transaction management system in the setting with its acquainted peers. Each peer p_i that is one of p's acquainted peers acts as a local peer of p. Figure 3.2 shows the setting of p and its acquainted peers expressed as a two-level MDBS.

The structure between peer p and its acquainted peers is the same as the structure of a MDBS. p acts as the global transaction management system with two-level transactions: the first level of transactions contains a sequence of parent transactions processed

on p and the second level contains all sequences of child transactions spawned to p's acquainted peers.

Before presenting our serializability theory, we need to introduce some preliminary concepts.

A concurrent execution of a set of transactions T_1, T_2, ... and T_n, in a peer database p is called a *local history* of p, *LH(p)*. A local history of p contains both global transactions received from other acquainted peers and local transactions initiated by its own users.

Let o_i and o_j be operations of two different transactions in a local history *LH(p)* on a peer p, where o_i precedes o_j, denoted $o_i \rightarrow o_j$ $(i \neq j)$. We say that o_i *directly conflicts* with o_j in *LH(p)* if they both operate on the same data item and at least one of them is a write operation.

Let o_i and o_j be operations of two different transactions in a local history *LH(p)*, where o_i precedes o_j, denoted $o_i \rightarrow o_j$ $(i \neq j)$. We say that o_i *indirectly conflicts* with o_j in o_j if there exist operations o_1, o_2, ..., on $(n \geq 1)$ of other transactions such that o_i directly conflicts with o1 in *LH(p)*, and o_1 directly conflicts with o_2 in *LH(p)*, ..., and on directly conflicts with o_j in *LH(p)*.

Let p_i and p_j ($i \neq j$)be two peers which are not acquainted in a P2P MDBS. We say p_i and p_j are *indirectly acquainted* to each other if there exists an acquaintance path between p_i and p_j.

After a set of transactions: $T = \{T_1, T_2, \ldots, T_n\}$, are submitted on a peer p, the set of transactions may be propagated to other peers as global transactions along acquaintances. We call the concurrent execution of the set of transactions T on p the *root history RH(p)*, then *RH(p)* will generate descendent histories on peers along the propagation path of T. A *Descendent History DH(pd)* of *RH(p)* is a local history executed on a destination peer p_d to which descendent transactions, that are translated from a

subset of global transactions included in T, are propagated. A global history *GH(p)* on p includes the root history *RH(p)* and all its descendent histories $DH(p_d)$.

Since global transactions are propagated hierarchically in a P2P MDBS, the serializability of the root history is also propagated to its descendent histories hierarchically and recursively from peer to peer in the parent-to-child inheritance way. We define a *two-level global history, 2LGH(p)*, which includes only a parent history and all child histories of this parent history, to facilitate the propagation of the global history. The two-level global history is defined with respect to a peer and its acquainted peers.

Let a peer p be acquainted to m peers p_1, p_2, ..., and p_m. T be a set of transactions $T = \{T_1, T_2, \ldots, T_n\}$, and $T_{i(1 \rightarrow i \rightarrow n)}$ be either a global transaction or a local transaction submitted to p. The local history of T on p is a parent history H_p. The local history H_i on p_i, which contains all T's child transactions spawned from p to p_i, is a child history H_{c_i} of H_p on p_i.

Definition 5 *A* Two-level Global History *on peer* p, 2LGH(p), *is a set* $\{H_p, H_1, H_2, \ldots, H_m\}$ *of local histories, where* H_p *is the parent history, which contains a set of transactions* $T = \{T_1, T_2, \ldots, T_n\}$*; the local history* H_i *is a child history of* H_p *on the acquainted peer* p_i*. The parent transactions of the child transactions contained in the child history* H_i *is a subset of transactions included in* T*. Either the parent history* H_p *or every child history* H_i *is a component history of* 2LGH(p).

Let G_i and G_j be two global transactions in a two-level global history *2LGH(p)*. We say that G_i *directly conflicts* with G_j in 2LGH(p) if one of G_i's operations directly conflicts with one of G_j's operations in the parent history H_p. We say that G_i *indirectly conflicts* with G_j in *2LGH(p)* if one of G_i's operations indirectly conflicts with one of G_j's operations in the parent history H_p.

Definition 6 *A Two-level Global History,* 2LGH(p), *on peer* p *with a set of transactions* $T = \{T_1, T_2, \ldots, T_n\}$, *is* ***two-level serial*** *if*

1. *All component histories are (conflict) serializable on the peer where they are located; and*

2. *There exists a total order of all global transactions in T such that for every global transactions T_i and T_j where T_i proceedes T_j in the order, all T_i's operations procede T_j's operations in all component histories in which they both appear.*

Example 3.2.2 *Suppose a sequence of transactions* $G = \{G_1, G_2, \ldots, G_n\}$, *are submitted to* p_1 *and executed in a serial order* $G_1G_2 \ldots G_n$ *in the DGL P2P MDBS. Their children* $\{ G_{1(1\rightarrow 2)}, G_{2(1\rightarrow 2)}, \ldots, G_{n(1\rightarrow 2)}\}$, *are spawned from* p_1 *to* p_2. *We say that* $2LGH(p_1)$ *is serial if:*

$$2LGH(p_1) = H(p_1), H(p_2)$$
$$H(p_1) = G_1G_2 \ldots G_n$$
$$H(p_2) = G_{1(1\rightarrow 2)}G_{2(1\rightarrow 2)} \ldots G_{n(1\rightarrow 2)}$$

Since we assumed that each peer ensures its own local serializability, the serializability of $H(p_1)$ *and* $H(p_2)$ *are ensured by* p_1 *and* p_2, *respectively. The set of child transactions are processed on* p_2 *in the same order that their parent transactions are processed on* p_1, *we say that the processing orders of parent history* $H(p_1)$ *and child history* $H(p_2)$ *are consistent over the acquaintance. The execution order of the parent history imposes the ordering of their child transactions. On the contrary, the child history inherits and preserves the execution order of the parent history.*

Definition 7 *A Two-level Global History 2LGH(p) is* ***two-level serializable*** *if it is (conflict) equivalent to a serial two-level global history.*

A *Serialization Graph* of a two-level global history *2LGH(p)*, denoted *SG(2LGH(p))*, is a directed graph whose nodes are the global transactions in *2LGH(p)*, and whose edges are all the relations $G_i \rightarrow G_j$ $(i \neq j)$ in both the parent history and child histories. The subgraph of *SG(2LGH(p))* composed of the subset of global transactions that are contained in a component history H_i are called as a *Component Serialization Graph* $(CSG(H_i))$ of *SG(2LGH(p))*. The component serialization graph with respect to the parent history H_p, $CSG(H_p)$, has the same nodes as *SG(2LGH(p))*.

Lemma 1 *Every component serialization graph* $CSG(H_i)$ *is* acyclic *if the two-level global serialization graph* SG(2LGH(p)) *of a two-level global history* 2LGH(p) *on a peer* p *is acyclic.*

Proof: Suppose there exist some component serialization graph $CSG(H_i)$ that is not acyclic when the serialization graph *SG(2LGH(p))* of a two-level global history *2LGH(p)* is acyclic. This component serialization graph may be defined by either the parent history H_p or a child history H_i. Then there must exist a cycle in the component serialization graph $CSG(H_i)$ because it is not acyclic. Because the two-level global serialization graph *SG(2LGH(p))* includes conflict relations in all component serialization graphs, this cycle must be included in *SG(2LGH(p))*. Therefore, *SG(2LGH(p))* is not acyclic because it includes a cycle. This is a contradiction with the condition that the two-level global serialization graph *SG(2LGH(p))* of a two-level global history *2LGH(p)* is acyclic. Therefore, there exists no component serialization graph that is not acyclic when the serialization graph *SG(2LGH(p))* of a two-level global history *2LGH(p)* is acyclic. All component serialization graphs are acyclic if the two-level global serialization graph *SG(2LGH(p))* of a two-level global history *2LGH(p)* is acyclic.

However, the reverse of Lemma 1 might not be true. Although all component serialization graphs $CSG(H_i)$ of a two-level serialization graph *SG(2LGH(p))* are acyclic, *SG(2LGH(p))* may contain cycle.

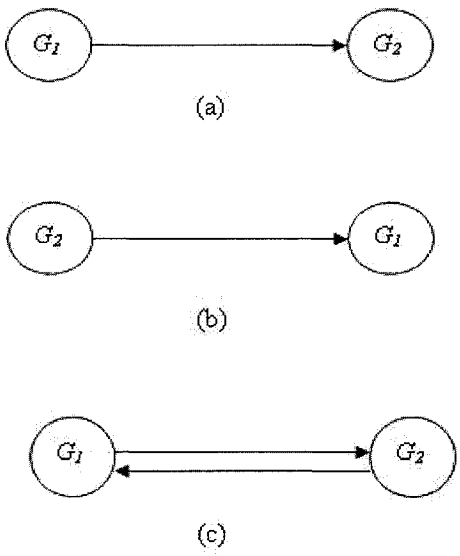

Figure 3.3: Component serialization graph on p_1 (a), Component serialization graph on p_2 (b), Twolevel Serialization graph of the twolevel global history $2LGH(p_1)$

Example 3.2.3 *Let G_1 and G_2 be two global transactions initiated on p_1 in the Digital Library MDBS presented in our motivating example, and $G = \{G_1, G_2\}$. If these two global transactions are executed in the ordering of G_1G_2 on p_1 and $H_p = G_1G_2$. Their child transactions $G_{1(1\to2)}$ and $G_{2(1\to2)}$ are executed in the ordering of $G_{2(1\to2)}G_{1(1\to2)}$ on p_2 and $H_{p_2} = G_{2(1\to2)}G_{1(1\to2)}$. Then the two-level serialization graph of the two-level global history $2LGH(p_1) = \{H_p, H_{p_2}\}$ is shown in Figure 3.3.*

Theorem 3.2.1 *A two-level global history* 2LGH(p) *is* two-level serializable *if and only if all component histories are (conflict) serializable and the two-level serialization graph* SG(2LGH(p)) *is acyclic.*

Proof: (if) Let a peer p be acquainted to multiple peers $p_1, p_2, \ldots,$ and p_m $(m > 0)$. Suppose *2LGH(p)* is a two-level global history on p and $2LGH(p) = \{H_p, H_1, H_2, \ldots, H_m\}$, H_p is the parent history and H_l $(1 \leq l \leq m)$ is a child history of H_p. G is the set of global transactions included in H_p, $G = \{G_1, G_2, \ldots, G_n\}$. Since *SG(2LGH(p))* is

acyclic, it may be topologically sorted. Let i_1, i_2, ..., i_n be a permutation of *1*, *2*, ..., n such that G_{i_1}, G_{i_2},..., G_{i_n} is a topological sort of *SG(2LGH(p))*. For each child history H_l ($1 \leq l \leq m$) on the acquainted peer p_l, assume that $G_{i_1,l}$, $G_{i_2,l}$, ..., $G_{i_n,l}$ are the child transactions that are spawned from p to p_l and appear in H_l. Let H_p' be the serial history G_{i_2}, ..., G_{i_n} on p.

First, we prove that H_p is conflict equivalent to $H_p^{'}$. From Lemma 1, we know that the component serialization graph $CSG(H_p)$ defined with respect to the parent history H_p is acyclic. Therefore, H_p is serializable according to the Serializability Theorem in [17][Page 33].

Second, we claim that there exists another serializable child history $H_l^{'}$ on the acquainted peer p_l, which is conflict equivalent to H_l, such that all of $G_{i_1,l}$'s operations precede $G_{i_2,l}$'s operations in $H_l^{'}$, all of $G_{i_2,l}$'s operations precede $G_{i_3,l}$'s operations in $H_l^{'}$, and so on. We prove this below.

$G_{i_1,l}$, $G_{i_2,l}$, ..., $G_{i_n,l}$ are the child transactions that are spawned from p to p_l and appear in H_l. We need to construct a serializable history $H_l^{'}$ which contains all operations in H_l. Let us group the operations in H_l into n operation sets based on global child transactions. Each group contains all $G_{i_q,l}$'s operations and those operations in H_l that must precede any one of $G_{i_q,l}$'s operations in any history which is conflict equivalent to H_l, but not in previous OP's.

For $q = 1$ to n do {

$$OP(i_q, l) = \{o \in H_l - \bigcup_{k=1}^{q-1} OP(i_k, l) - \bigcup_{k=q+1}^{n} OP(i_k, l) :$$

a. $o \in G_{i_q,l}$, OR

b. o conflicts with one of $G_{i_q,l}$'s operations.

}

Each group operations $OP(i_q, l)$ have the following properties:

1. $OP(i_q, l)$ contains all operations of the child transaction G_{i_q}, but no operation of other child transactions, where $1 \leq q \leq n$.

2. $\bigcup_{k=1}^{n} OP(i_k, l)$ consists of all the operations in H_l.

3. $OP(i_q, l) \cap OP(i_v, l) = \varnothing$ where $l \leq q \neq v \leq n$.

Let SHC be the constructor that uses OP to constructs another serializable history on the peer p_l, which has the same conflict relations as H_l. This can be achieved by ordering all operations in OP in the same way as in H_l. Thus can be constructed as follows:

$$H_l^{'} = SHC(OP(i_l, l) \circ SHC(OP(i_2, l) \circ \ldots \circ SHC(OP(i_n, l))$$

Where $\circ$ is the concatenation operation of subhistories. We claim that:

1. $H_l^{'}$ contains the same transactions as H_l;

2. $H_l^{'}$ is conflict equvalent to H_l;

3. Chile transactions in H_l are executed sequentially in $H_l^{'}$.

We need to prove that $H_l^{'}$ is conflict equivalent to H_l. Let o_a and o_b $(a \neq b)$ be two conflict operations in H_l and . If $b < n$, then o_b either belongs to $G_{i_b,l}$ or conflicts with one of $G_{i_b,l}$'s operations, and so is o_a. Therefore, either where $a < b$, or by the definition of $OP(i_b, l)$. This is also true when $b = n$. So, o_a also conflicts with o_b in.

Let $2LGH_{(p)}^{'} = \{H_p^{'}, H_1^{'}, H_2^{'}, \ldots, H_n^{'}\}$, then $2LGH_{(p)}^{'}$ is a two-level serial global transaction and is conflict equivalent to *2LGH(p)*. Therefore, *2LGH(p)* is a global serializable history in the P2P MDBS.

(only if) Let *2LGH(p)* be a serializable global history on the peer $p$in the P2P MDBS and $2LGH(p) = \{H_p, H_1, H_2, \ldots, H_m\}$. We assume G is the set of global transactions

in H_p and $G = \{G_1, G_2, \ldots, G_n\}$. Let $GH'_{(p)}$ be a serial global history which is conflict equivalent to *2LGH(p)*. Then $SG(2LGH(p)) = SG(2LGH(p)')$. Since G_1, G_2, ..., G_n are executed sequentially in $2LGH'_{(p)}$ and any operation can only conflict with subsequent operations in $2LGH'_{(p)}$, $SG(2LGH'_{(p)})$ must be acyclic. Because *2LGH(p)* is conflict equivalent with $GH'_{(p)}$, *SG(2LGH(p))* is also acyclic.

Definition 8 *A Global History* GH(p) *with respect to a set of transactions* $T = \{T_1, T_2, \ldots, T_n\}$ *on a peer* p *is* ***globally P2P serial*** *if*

1. *All component histories are (conflict) serializable on the peer where they are located; and*

2. *There exists a total order of all global transactions in T such that for every two global transactions T_i and T_j where T_i precedes T_j in the order, all T_i's operations precede T_j's operations in all component histories in which they both appear.*

Definition 9 *A Global History* GH(p) *with respect to a set of transactions* $T = \{T_1, T_2, \ldots, T_n\}$ *on a peer* p *is* ***globally P2P serializable*** *if it is (conflict) equivalent to a serial global history.*

3.2.3 Serializability Theorems in P2P MDBSs

Although the serializability of a root history is always propagated hierarchically and recursively to its descendent histories in the parent-to-child inheritance way, we can not say that the serializability of the root history is always preserved in the global transaction's propagation. In some situations, the serializability of the root history may be violated. For example, when there exist acquaintance cycles between a peer and a destination peer, the propagation of the global history in an acquaintance cycle may violate the serializability. As we discussed before, a global transaction may be translated completely or partially during its propagation. When a transaction is translated partially, its child transactions only include operations that are translated from a subset of its parent's

operations.

Example 3.2.4 *Let* G *be a global transaction initiated on* p_1 *in the DGL P2P MDBS shown in Figure 3.4.*

$$G : w(a)w(b)$$

Suppose data item a is mapped over the acquaintance $p_1 \leftrightarrow p_2$*, and data items a and b are both mapped over the acquaintances* $p_1 \leftrightarrow p_3 \leftrightarrow p_2$*. First,* p_1 *spawns* G*'s child transactions* $G_{(1\rightarrow 2)}$ *and* $G_{(1\rightarrow 3)}$ *to* p_2 *and* p_3 *respectively.* $G_{(1\rightarrow 2)}$ *is a partial translation of* G*. Then* $G_{(1\rightarrow 3)}$*'s child transaction* $G_{(1\rightarrow 3\rightarrow 2)}$ *is spawned to* p_2 *as well. The detail propagation of* G *is shown in Figure 3.4.*

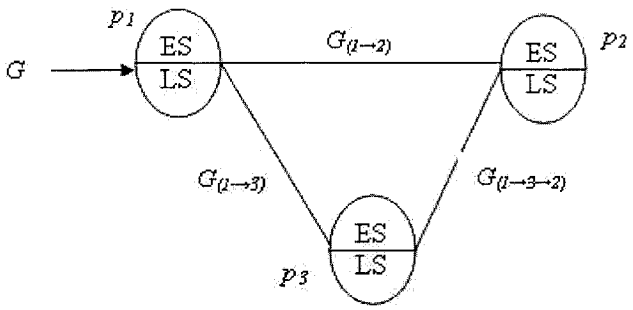

Figure 3.4: Propagation of G in DGL P2P MDBS

$G_{(1\rightarrow 2)} : w(a')$
$G_{(1\rightarrow 3)} : w(a'')w(b'')$
$G_{(1\rightarrow 3\rightarrow 2)} : w(a')w(b')$

$G_{(1\to 2)}$ and $G_{(1\to 3\to 2)}$ are two descendent transactions of the same root transactions G. Since the partial transaction translation is allowed, the operations contained in $G_{(1\to 2)}$ and $G_{(1\to 3\to 2)}$ are different. $G_{(1\to 2)}$ and $G_{(1\to 3\to 2)}$ are two different transactions on p2 although they are from the same root transaction G. Further more, $G_{(1\to 2)}$ and $G_{(1\to 3\to 2)}$ conflict to each other on p_2.

In Example 3.2.4, we see how the consistency of a global transaction is violated when descendent transactions which are partial translation of the root transaction are propagated in an acquaintance cycle. When multiple different descendent transactions of a global transaction G are propagated to a common destination peer p_d, the destination peer pd is highly unlikely to control the consistency of the global transaction because of the autonomy of each peer on the propagation path. However, we observe that these multiple descendent transactions of G are propagated to p_d via different propagation paths between the source peer, where the root transaction of G is located, and the destination peer. To sum up any two of these propagation paths between the source peer and the destination peer, an acquaintance cycle that includes the source peer and the destination peer can be constructed. If there exists no cycle between the source and destination peer, there exists a unique propagation path from the source peer to the destination peer. Therefore, at most one descendent transaction of a global transaction may be propagated to the destination peer via the unique propagation path based on our Assumption 2. In Assumption 2, it is said that a peer only submits one child transaction of a global transaction to a directly acquainted peer when a child transaction spawning is possible, whether it is a complete or partial translation of the global transaction. The consistency of the global transaction can always be maintained if there exists no cycle between the source peer and the destination peer.

Example 3.2.5 *Let G_1, G_2, and G_3 be three global transactions initiated on the peer p_1 in the DGL P2P MDBS. These three transactions are executed in the ordering of $G_1G_2G_3$ on p_1. Suppose that only complete transaction translation is allowed in the DGL P2P MDBS. First, G_1's child transaction $G_{1(1\to 3)}$ is spawned from p_1 to p_3, and G_2's child*

transaction $G_{2(1\rightarrow 2)}$ and G_3's child transaction $G_{3(1\rightarrow 2)}$ are spawned from p_1 to p_2. Then, $G_{1(1\rightarrow 3)}$'s child transaction $G_{1(1\rightarrow 3\rightarrow 2)}$ is spawned to p_2 again. The propagation of these three transactions is shown in Figure 3.5.

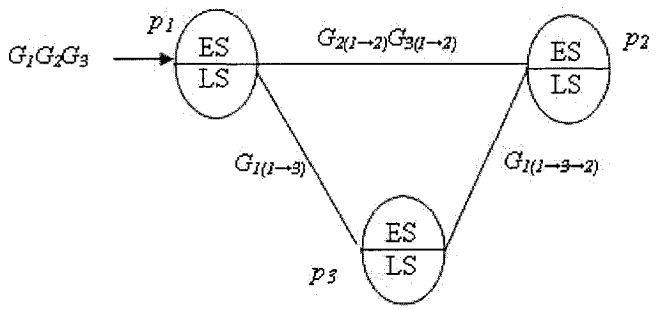

Figure 3.5: The propagation of G_1, G_2, and G_3 in the *DGL* P2P MDBS

According to the two-level global serializability that we defined before, $G_{2(1\rightarrow 2)}$ and $G_{3(1\rightarrow 2)}$ are executed in the ordering of $G_{2(1\rightarrow 2)}G_{3(1\rightarrow 2)}$ on p_2. Since $G_{1(1\rightarrow 3\rightarrow 2)}$ is also spawned from p_3 to p_2, the global serializability $G_1G_2G_3$ of the root history may be violated on p_2. Because of the autonomy of each peer, p_2 has no knowledge what child transactions that it will receive from p_1 and p_3, respectively. p_2 can not guarantee to execute $G_{2(1\rightarrow 2)}$ and $G_{3(1\rightarrow 2)}$ after $G_{1(1\rightarrow 3\rightarrow 2)}$ arrives and has been executed. Thus the global serializability may be violated because these three descendent transactions from the same root history are propagated to p_2 from different peers. The ordering that these three descendent transactions arrive at p_2 is almost impossible to be controlled because p_2 and p_3 are fully autonomous and there is no central mechanism to control the actions of p_2 and p_3. This result is caused by an acquaintance cycle which includes three peers p_1, p_2, and p_3. Transactions are propagated from p_1 to p_2 from two acquaintance path: $p_1 \leftrightarrow p_2$ and $p_1 \leftrightarrow p_3 \leftrightarrow p_2$.

In Example 3.2.5, transactions are not always translatable over every acquaintance in the DGL P2P MDBS although only complete transaction translation is allowed. At

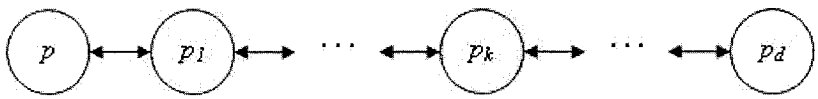

Figure 3.6: Propagation path with no cycle between p and p_d

the first propagation level, only G_2 and G_3 are translatable over the acquaintance $p_1 \leftrightarrow p_2$, while G_1 is translatable over the acquaintance $p_1 \leftrightarrow p_3$. Because of the existence of an acquaintance cycle, descendent transactions from the same root history may propagated to a destination peer included in the cycle from various acquaintance paths eventually. This may violate the serializability of the root history.

Proposition 2 *In a P2P MDBS, a global history GH(p), which is defined with respect to a set of global transactions $G = G_1, G_2, \ldots, G_n$ $(n > 0)$ submitted to a peer p, is serializable on the propagation path between p and a destination peer p_d if*

1. *The acquaintance path is acyclic between p and a destination peer p_d; and*
2. *Global histories are two-level serializable on every acquaintance on the propagation path between p and p_d.*

Proof: We prove Proposition 2 with induction on each acquaintance between p and p_d. Let l be the length of the propagation path of G between p and p_d. The path is shown in Figure 3.6.

When $l = 0$, the length of G's propagation path is *0*, i.e., the set of global transactions G is at the propagation level L_0 and the global history includes only the root history *RH(p)*. Obviously, the root history preserves the serializability of *GH(p)* since the serializability of the root history is the one that we need to ensure along the propagation path.

When $l = 1$, the global history includes the root history and its child history on peer p_1, which is acquainted to p. Since global histories over the acquaintance $p \leftrightarrow p_1$ are two-level serializable, the child history on p_1 certainly preserves the serializability of its parent history on p according to the definition how a set of global transaction are serializable. So Proposition ẽfproposition2 is true when $l = 0$, *1.*

For the induction step, we assume that the statement holds for some k ($0 < k < l$), i.e., that *GH(p)* is serializable along the propagation path between p and a peer p_k, p_k is a peer on the propagation path between p and p_d and descendent transactions of G arrive at p_k before arriving p_d. We need to show that this statement holds for the peer p_{k+1} when $l = k + 1$. p_{k+1} is a peer that is directly acquainted to p_k on the propagation path between p and p_d, and descendent transactions of G are spawned from p_k to p_{k+1}.

By our inductive hypothesis, the serializability of *GH(p)* is preserved in the global history $GH(p_k)$ on p_k. The global history $GH(p_k)$ contains all descendent transactions of G propagated to p_k, and all these descendent transactions are processed in a consistent ordering as their root transactions are processed in the root history. From the statement, we know that the global history $GH(p_k)$ on p_k is two-level serializable over the acquaintance $p_k \leftrightarrow p_{k+1}$. This means that the child history $GH(p_{k+1})$ of $GH(p_k)$ on p_{k+1} preserves the serializability of its parent history $GH(p_k)$, i.e., the set of child transactions in $GH(p_{k+1})$ from p_k preserve the serializability of their parent transactions on p_k. Since the descendent transactions of G on p_{k+1} is a subset of child transactions spawned from p_k, the descendent transactions of G on p_{k+1} also preserve the serializability of their parent transactions on p_k. Thus the descendent transactions of G in $GH(p_{k+1})$ preserves the serializability of *GH(p)* because the serializability of *GH(p)* is preserved in the global history $GH(p_k)$ on p_k.

So we proved that the serializability of *GH(p)* is preserved on p_{k+1} if it is preserved on p_k along the propagation path between p and p_d.

Therefore, by the Principle of Mathematical Induction, we proved Proposition 2.

Definition 10 *A global history is* fully translatable *over an acquaintance if every transaction is completely translatable over the acquaintance.*

However, the serializability of a global history can be controlled when the global history is always fully translatable over every acquaintance between a source peer and a destination peer even though these two peers are contained in an acquaintance cycle.

Example 3.2.6 *Suppose that each one of the transactions G_1, G_2, and G_3 in Example 3.2.5 is translatable over every acquaintance in the DGL P2P MDBS. According to our Assumption 2 that a peer spawns at most one child transaction of a global transaction to an acquainted peer, descendent transactions which are translated from the same global transaction and arrive at the same destination peer are always identical. p_3 just discards the descendent transaction duplicates if it has already received one copy of the descendent transaction from a transaction. Since p_1 and p_3 spawn the descendent transactions of G_1, G_2, and G_3 to p_2 in the same ordering of $G_1G_2G_3$, p_2 is able to receive these three transactions in the ordering of $G_1G_2G_3$. Let's see how the ordering that p_2 receives these three transactions can be controlled. Suppose p_2 receives G_1 from p1 first, then it will discard the descendent transaction copy of G_1 from p_3. When the first descendent transaction copy of G_2 arrives, p_2 receives it as a new global transaction whichever peer this descendent transaction copy is from. The same thing will happen when the descendent transaction copies of G_3 arrive. Thus the ordering of global transactions from the same source peer can be preserved in the ordering that they are spawned out from the source peer.*

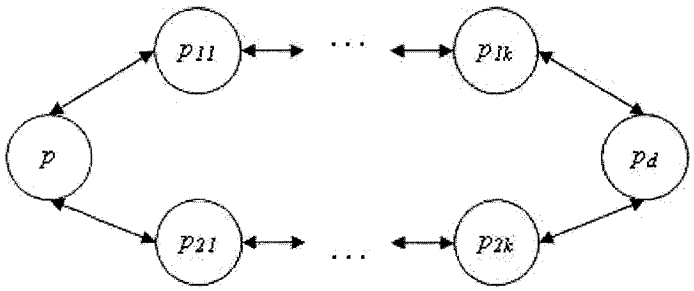

Figure 3.7: Propagation path with no cycle between p and p_d

Proposition 3 *In a P2P MDBS, a global history GH(p), which is defined with respect to a set of global transactions* $G = \{G_1, G_2, \ldots, G_n\}$ *(*$n > 0$*) submitted to a peer p, is serializable on the propagation paths between p and a destination peer* p_d *if:*

1. *p and* p_d *are contained in an unique acquaintance cycle; and*
2. *Anyone of the propagation path between p and* p_d *is acyclic; and*
3. *The set of descendent transactions of G, included in the descendent history* $DH(p_d)$ *located on* p_d*, preserves the order in which their root transactions are executed in the root history* $RH(p)$.

Proof: As it is shown in Figure 3.7, the source peer p and the destination peer p_d are contained in an acquaintance cycle. Thus there are two paths between p and p_d through which the global history *GH(p)* is propagated. These two paths are expressed as follows:

Path 1: $p \leftrightarrow p_{11} \leftrightarrow \ldots \leftrightarrow p_{1k} \leftrightarrow p_d$
Path 2: $p \leftrightarrow p_{21} \leftrightarrow \ldots \leftrightarrow p_{2k} \leftrightarrow p_d$

According to Proposition 2, the global history *GH(p)* is serializable on both sub-path of Path 1 and Path 2 because these sub-paths are acyclic:

Sub-path of Path 1: $p \leftrightarrow p_{11} \leftrightarrow \ldots \leftrightarrow p_{1k}$
Sub-path of Path 2: $p \leftrightarrow p_{21} \leftrightarrow \ldots \leftrightarrow p_{2k}$

This means that all descendent histories executed on peers located on the propagation path preserve the serializability of the root history of *GH(p)*.
From the condition 4) in proposition 3, we know that the descendent history on p_d preserves the serializability of *GH(p)*. So all descendent histories preserve the serializability of *GH(p)*.

Therefore, Proposition 3 is correct.

Example 3.2.7 *Let T_1, T_2, and T_3 be three transactions initiated on p_1 in the P2P MDBS shown in Figure 3.8. These three transactions are processed in the order $T_1 \rightarrow T_2 \rightarrow T_3$. Suppose three child transactions $T_{1(p_1 \rightarrow p_2)}$, $T_{2(p_1 \rightarrow p_2)}$, and $T_{3(p_1 \rightarrow p_2)}$ are spawned from p_1 to p_2 one by one, and two child transactions $T_{1(p_1 \rightarrow p_3)}$ and $T_{2(p_1 \rightarrow p_3)}$ are spawned from p_1 to p_3 one by one. p_2 processes these three transactions in the ordering of $T_{1(p_1 \rightarrow p_2)} \rightarrow T_{2(p_1 \rightarrow p_2)} \rightarrow T_{3(p_1 \rightarrow p_2)}$, and p_3 processes the two transactions in the order of $T_{1(p_1 \rightarrow p_3)} \rightarrow T_{2(p_1 \rightarrow p_3)}$. Suppose p_2 spawns $T_{1(p_1 \rightarrow p_2 \rightarrow p_3)}$, $T_{2(p_1 \rightarrow p_2 \rightarrow p_3)}$, and $T_{3(p_1 \rightarrow p_2 \rightarrow p_3)}$ to p_3 again. However, p_3 ignores $T_{1(p_1 \rightarrow p_2 \rightarrow p_3)}$ and $T_{2(p_1 \rightarrow p_2 \rightarrow p_3)}$ since the descendent transactions of T_1 and T_2 have been received from p_1. p_3 only processes $T_{3(p_1 \rightarrow p_2 \rightarrow p_3)}$ as a new received global transaction. In the same protocol, p_3 also propagates child transactions $T_{1(p_1 \rightarrow p_3 \rightarrow p_2)}$ and $T_{2(p_1 \rightarrow p_3 \rightarrow p_2)}$ to p_2. p_2 just discards these two child transactions when it receives them since the identical copies of these two child transactions have been received form p_1 before.*

Theorem 3.2.2 *In a P2P MDBS, a global history GH(p), which is defined with respect to a set of global transactions $G = \{G_1, G_2, \ldots, G_n\}$ $(n > 0)$ submitted to a peer p, is globally P2P serializable if*

1. *There exists no acquaintance cycle during the propagation of the global history GH(p); and*

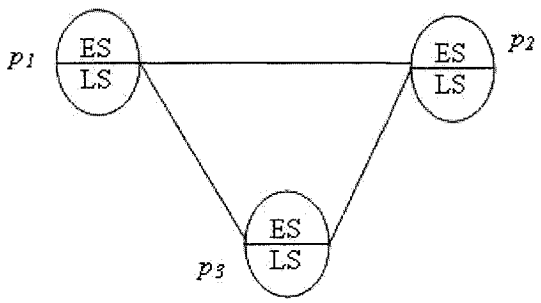

Figure 3.8: Cyclic Propagation in DGL P2P MDBS

2. *Global histories are two-level serializable on every acquaintance that is included in the propagation of the global history.*

Proof: Because there exists no acquaintance cycle during the propagation of the global history *GH(p)*, there exists only one unique acyclic acquaintance path between p and any destination peer pd. Via this unique path, *GH(p)* is propagated from p to p_d. Based on Proposition 2, *GH(p)* is serializable on every propagation path because the path is acyclic. Thus the descendent histories on every propagation path of *GH(p)* preserve the serializability of the root history of *GH(p)*. All together, all descendent histories on all propagation paths preserve the serializability of the root history of *GH(p)* during the propagation.

Therefore, *GH(p)* is globally P2P serializable according to the definition.

3.3 Consistency on an Acquaintance

A peer establishes at least one acquaintance to another peer that is already in a P2P MDBS when it joins the P2P MDBS. All acquaintances together construct the network of the P2P MDBS. Basically, each acquaintance is a connection unit to construct the network. It is fundamental to ensure the consistency of each acquaintance in order to maintain the consistency of the P2P MDBS. In last section, we defined the global serializability of a set of transactions defined with respect to a single peer p. Since a

peer not only spawns global transactions to its acquainted peers, but also receives global transactions from its acquainted peers, global transactions and local transactions are interoperated by the peer DBMS on top of the peer database.

Over two acquainted peers: p_1 and p_2, let $G_1 = \{G_{11}, G_{12}, \ldots, G_{1n}\}$ be a set of transactions submitted to p_1; $G_2 = \{G_{21}, G_{22}, \ldots, G_{2n}\}$ be a set of transactions submitted to p_2; G is the union of G_1 and G_2. G_1's child transactions are spawned from p_1 to p_2. At the same time, G_2's child transactions are also spawned from p_2 to p_1. However, the execution order of the same set of global transactions G may be different on p_1 and p_2. Let's see an example.

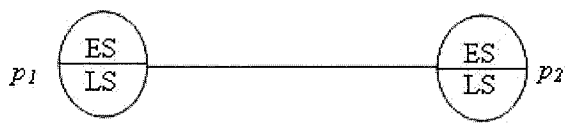

Figure 3.9: Single acquaintance between p_1 and p_2

Example 3.3.1 *Two peers p_1 and p_2 are acquainted. After the acquaintance between p_1 and p_2 has been established, global transactions are propagated between two peers over the acquaintance. Suppose a transaction G_1 is initiated on p_1 and a transaction G_2 is initiated on p_2 at the same time. p_1 spawns G_1's child transaction $G_{1(1\to2)}$ to p_2; p_2 does the same thing and $G_{2(2\to1)}$ is spawned from p_2 to p_1.*

Since two peers are fully autonomous, both peers process the transactions in the order that these transactions are submitted to the peers. Because these two transactions are initiated on their home peers at the same time, the home peer of each transaction receives the transaction first since there is a transmitting delay for a child transaction over the network. So these two transactions are executed in two different orders on p_1 and p_2. p_1 executes them in the order of $G_1 G_{2(2\to1)}$, while p_2 executes them in the order of $G_2 G_{1(1\to2)}$.

If a set of transactions are executed in the same order on some peers, we say that the set of transactions are in *one-copy* transaction ordering on those peers. From Example 3.3.1, we realize that to keep a one-copy transaction ordering for a common set of global transactions from both acquainted peers are highly unlikely to be achieved over an acquaintance. Each peer is fully autonomous and has no control with other peers. The question arises that what kind of global transaction ordering is acceptable over an acquaintance in P2P MDBSs and is sufficient to maintain the consistency of the acquaintance.

3.3.1 Global Timestamping Ordering

Global transactions initiated on different peers are not related to each other when they are being executed on their own peer as local transaction. Their descendent transactions are related to one another after they are spawned to a common destination peer. Some criterion is needed to order global transactions from different home peers gathered on the common peer. Regarding global transactions over an acquaintance, the most intuitive solution is to set a global transaction management scheme for global transactions initiated on both acquainted peers. Similar to conventional MDBSs, we may assign a unique global timestamp to each global transaction initiated in any one of the two peers over the acquaintance. If we do so, the global timestamp can be used to order all transactions on each single peer whether the transactions are submitted as global or local transactions since every transaction has a unique global timestamp. The strategy of the global timestamping ordering can be used to ensure the serializability for global transactions in each peer over the acquaintance.

Example 3.3.2 *Two peers p_1 and p_2 are acquainted. Suppose a transaction G_1 is initiated on p_1 and $ts(G_1)$ is the global timestamp assigned to G_1; a transaction G_2 is initiated on p_2 and $ts(G_2)$ is the global timestamp assigned to G_2. $ts(G_1) < ts(G_2)$. p_1 spawns G_1's child transaction $G_{1(1\rightarrow 2)}$ to p_2; p_2 does the same thing and $G_{2(2\rightarrow 1)}$ is spawned from*

p_2 to p_1. Based on the global timestamp, G_1 and G_2 can be controlled to execute in the same ordering: $G_1 G_{2(2\rightarrow 1)}$ on p_1 and $G_{1(1\rightarrow 2)} G_2$ on p_2. Obviously, the ordering for G_1 and G_2 are consistent on p_1 and p_2.

Over an acquaintance, the global timestamping ordering might be a good solution to control the serializability of global transactions from both acquainted peers. However, when the global timestamping ordering is enforced over each acquaintance in a P2P MDBS, global transactions initiated in any peer are executed in a unique global ordering. We actually set the global timestamp as a virtual global control scheme for transaction management in the P2P MDBS. This is in the same management concept as conventional MDBSs. In some small P2P MDBSs with only several peers and acquaintances, the global timestamp ordering for global transactions might be achieved easily. Regarding large scale of P2P MDBSs which consist of hundreds of peers or more, there may be large number of global transactions being propagated in the P2P network. The global timestamping ordering is highly unlikely to be enforced in large scale of P2P MDBSs. The analysis of the complexity to enforce the global timestamping ordering in P2P MDBSs will be left as future work.

3.3.2 Global Transaction Ordering over an Acquaintance

In P2P MDBSs, peer databases belonging to an interest group are all related to a common business. The applications, such as health care, bio-informatics database systems, airlineticket reservation peer databases, and some ecommerce systems, are suitable for P2P MDBS architecture. The goal of P2P MDBSs is to achieve data exchanging and sharing between acquainted peers. Thus, each of these business groups has its own objective when its participating peers agree to build a P2P MDBS to deal with their business. The objective and business convention of each interest group, and the mapping information decide what kind of global transactions are desired in each P2P MDBS. Obviously, not all kinds of transactions are desired to process as global transactions. Let's analyze the desired global transactions and the global transaction ordering in some P2P interest groups.

In the *Digital Libraries* P2P MDBS, peers intend to share their online resources. The transactions, that may be processed globally, include transactions of books' searching and accessing, books' sharing and updating between peers, and members' sharing, etc. For example, when two global transactions G1 and G2 to insert new books are processed on two acquainted peers p1 and p2, respectively. If p1 processes G1 and G2 in the order of G1 G2 and p2 processes them in the order of G2G1, we still say that the acquaintance is consistent. What we need is to finish all the insertions of new books in predefined time duration, the ordering to process G1 and G2 does not violate the consistency of two acquainted peers. The temporal inconsistency can be tolerated by the digital library P2P system during the time when these two transactions are being processed.

In the *Biological* P2P Database System mentioned in [5], two peers may establish an acquaintance to relate gene data in one peer to the related protein data in another peer. Obviously, such a P2P MDBS is mainly used for information exchanging and sharing. In general, users on the gene peer try to find the related protein of a specific gene, while users on the protein peer need to figure out the gene that matches a specific protein. Most global transactions are data reading operations, while global transactions containing writing operations are rare. Since transactions with only read operations do not conflict to each other, the number of conflicts between global transactions is small. The constraints for the ordering of global transactions do not need to be very rigorous in the Biological P2P MDBSs.

In the *Medical* P2P Database System we presented before, participating peers intend to share and exchange patient information about medical histories, medication, symptoms, treatments and the like. Basically, the system requires that the patient's information is updated in time. However, it is acceptable if the ordering of global transactions to update different patients' information vary in different peers.

All application domains of P2P MDBSs presented above show that different ordering for the same set of global transactions on two acquainted peers are acceptable. The

acquaintance consistency is not violated if two acquainted peers execute the same set of global transactions in different ordering. Although different serial execution of global transactions on a single peer may produce various effects, they are acceptable and tolerated from the point of view of P2P MDBSs. We do not need a criterion which is as rigorous as the global timestamping ordering to manage global transactions over an acquaintance unless it is specified or required by the business group. This is completely different from MDBSs. It is nature to execute global transactions just according to the order in which they are submitted to each peer over the acquaintance. The relaxed ordering for global transactions is application domain dependent. If an interest group has special requirements to the ordering for global transactions, the requirements can be added as consistency constraints over each acquaintance. These consistency constraints can be expressed as *consistency predicates.*

Example 3.3.3 *Two acquainted peers p_1 and p_2 in the digital library P2P MDBS, they agree to use the same book downloading rate. Beside mapping tables and mapping expressions, this constraint can be added on the acquaintance between p_1 and p_2 in terms of consistency predicate.*

$$\text{OTT.downloadR} = \text{NY.downloadR}$$

This predicate is always evaluated whenever a global transaction to update the download rates of books has been executed on both p_1 and p_2. We say that the acquaintance consistency has not been violated if the consistency predicate is evaluated to be true after the global transaction has been processed on both peers.

3.3.3 Consistency over an Acquaintance

When we talk about the consistency over an acquaintance, the acquaintance involves not only two acquainted peers, but also all elements to establish the acquaintance, which include mapping tables, mapping expressions, and consistency predicates. Three aspects need to be considered in order to define an acquaintance to be in a consistent state.

1. We need to ensure the consistency of each peer and this can be guaranteed by the autonomy of the peer.

2. We need to evaluate true all consistency constraints over the acquaintance when global transactions are propagated between two acquainted peers. In general, the consistency constraints over the acquaintance are expressed in mapping tables and mapping expressions. It is also possible for peers to enforce some consistency constraints other than mapping tables and expressions over the acquaintance.

3. We need to preserve the serializability of global transactions defined with respect to either one of two acquainted peers.

4. We need to follow the correctness criteria for global transactions' ordering that is set up by every application domain.

Definition 11 *An acquaintance between two acquainted peers p_1 and p_2 is in a consistent state if*

1. *Global histories on both peers are serializable; and*

2. *The transactions' ordering on each peer follows the correctness criteria of the application domain;*

3. *All consistency constraints over the acquaintance are evaluated to be true.*

3.4 Consistency of P2P MDBSs and Partial Transaction Translations

After we define the consistent state for a single acquaintance, we address the problem of the consistency of a P2P MDBS in this section. The consistency of the P2P MDBS cannot be guaranteed even if the consistency of each acquaintance is ensured in the

P2P MDBS. As we mentioned before, a global transaction may be translated partially or completely into a child transaction when it is spawned to an acquainted peer. The global serializability cannot be ensured with partial transaction translations combined with acquaintance cycles in P2P MDBSs.

Obviously, a global transaction's atomicity is violated if the partial transaction translation is allowed in P2P MDBSs. In order to define a consistent state of a P2P MDBS, we need to solve more problems raised when partial transaction translation is allowed. The first problem is about the atomicity of a global transaction. We need to give a proper definition to the atomicity of the global transaction when partial transaction translation is allowed. The second problem is about the recovery of the global transaction in its propagation. We need to figure out how the consistency of the global transaction will be affected when the descendent transactions from the same root transaction conflict to each other on a peer that is not directly acquainted to the global transaction's home peer. All these problems will leave as the future work of the project.

At this stage, we have no knowledge on how a transaction can be partially translated to its acquainted peers if partial transaction translation is allowed in P2P MDBSs. If we do not consider any factor which may take effect on the partial transaction translations, we suppose that a transaction can be translated freely. Then the number N of the translations of a transaction with n operations ($n > 0$) submitted to a peer p is as follows:

$$N = n + n(n-1) + n(n-1)(n-2) + \ldots + n!$$

Let a peer p be acquainted to multiple peers p_1, p_2, ..., and p_m ($m > 0$). Then there are C_m^N ways to spawn these N translations to m acquainted peers.

$$\mathrm{C}_m^N = \frac{N!}{m!(N-m)!} \tag{3.3}$$

Because there are so many ways to translate and propagate a global transaction in a P2P MDBS, we need to consider what kinds of partial transaction translations are effective and necessary for P2P MDBSs. When partial transaction translation is allowed in the P2P MDBS, we need to know the complexity when a sequence of global transactions is propagated in a P2P MDBS with a certain number of peers. All these issues will leave as our future work.

Chapter 4

Concurrency Control Protocol

Based on Theorem 3.2.2, we develop a concurrency control protocol that ensures the global serializability by controlling the two-level serializability over every acquaintance during the propagation of a set of global transactions under the condition that there exists no acquaintance cycle during the propagation. This protocol is applied on each acquaintance to ensure the two-level serializability of global histories from both acquainted peers. All assumptions mentioned in the first section are followed by the protocol.

From the analysis done before, we know that each component transaction of a global transaction is independent. Every component transaction is located on an autonomous peer and being processed independently. There is no value dependency between any two of the component transactions. Our protocol ensures the consistency of actions, which include abort and commit, between a parent transaction and a child transaction so as to preserve the global serializability between acquainted peers over an acquaintance. Since each peer database is autonomous, the actions' synchronization between all component transactions of a global transaction is not an objective that we must achieve in our protocol for P2P MDBSs. Actually, it is highly unlikely to achieve the actions' synchronization between all component transactions of the global transaction because the transaction may be propagated a long distance from its home peer. A peer which receives a descendent transaction will not wait to commitabort the descendent transaction until

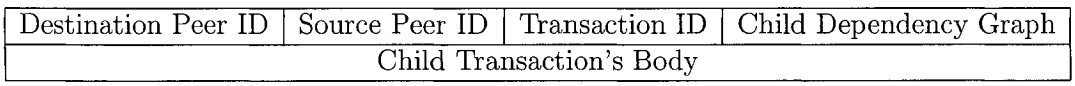

Figure 4.1: Global transaction message format

the end of the global transaction's propagation. In our protocol, the child transactions of a global transaction T are spawned out to acquainted peers right after T commits, while an uncommitted transaction will not spawn any child transaction out to any acquainted peer. In a peer database, the protocol controls the ordering of each child history which are spawned from the different source peers and make the ordering be consistent with their parent histories. No effort is needed to deal with the action consistency of an uncommitted global transaction. By using this strategy, the number of messages being passed between acquainted peers is reduced tremendously. This enhances the reliability of our protocol.

4.1 Our Protocol

A child transaction is encapsulated in a single global transaction message in our protocol. According to Assumption 2, a transaction spawns at most one child transaction to an acquainted peer. Thus at most one global transaction message is transmitted from a source peer to a destination peer. The child transaction is not segmented so as to ensure the atomicity of the child transaction. The format of a message to encapsulate a child transaction spawned to a destination peer is shown in Figure 4.1.

Let G_i and G_j be two global transactions in a two-level global history *2LGH(p)*. We say that G_i *directly depends on* G_j if G_i directly conflicts with G_j in the global history *2LGH(p)*; G_i *indirectly depends on* G_j if G_i indirectly conflicts with G_j in the global history *2LGH(p)*. Obviously, G_i precedes G_j in both direct and indirect dependency relationships.

This protocol utilizes the distributed dependency graph to preserve the global serializability between acquainted peers. A dependency graph on a peer p consists of nodes and edges. A node is a transaction T and an edge is a dependency between two conflict transactions.

Let p be a peer and p_i be an acquainted peer of p, we say that the dependency graph on p is the parent dependency graph $PDG(p)$, and the dependency graph for an acquainted peer p_i a child dependency graph $CDG(p{\rightarrow}p_i)$ of *PDG(p)*. On peer p, a child dependency graph $CDG(p{\rightarrow}p_i)$ is constructed for each of its acquainted peer p_i. A node in $CDG(p{\rightarrow}p_i)$ is a transaction whose child transaction is spawned to p_i and an edge is a dependency between two transactions, which is either a direct dependency or an indirect dependency.

Since the child dependency graph is constructed with respect to the set of transactions whose child transactions are spawned to p_i, the child dependency graph is derived from the parent graph, which includes both direct dependency and indirect dependency between transactions. Two child transactions contained in an indirect dependency preserve the partial order of their parent transactions. The child dependency graph $CDG(p{\rightarrow}p_i)$ is sent to p_i with a child transaction when the child transaction is spawned from p to p_i. The child dependency graph for different destination peers may vary since the set of global transactions spawned to every acquainted peer may be different.

Example 4.1.1 *Over two acquainted peers p_1 and p_2, let $G_1 = \{G_1, G_2, G_3, G_4\}$ be a set of transactions submitted to p_1. The child transactions of G_1, G_3, and G_4 are spawned to p_2. The parent dependency of these four transactions on p_1 is shown in Figure 4.2, and the child dependency graph of G_1, G_3, and G_4 is shown in Figure 4.3. In the parent dependency graph, each dependency is a direct dependency caused by a direct conflict between two transactions. However, both direct and indirect dependencies are included in the child dependency graph. The dependency between $G_{1(1\rightarrow2)}{\rightarrow}G_{4(1\rightarrow2)}$ is*

an indirect dependency derived from two direct dependencies$G_1 \rightarrow G_2$ *and* $G_2 \rightarrow G_4$ *in the parent dependency graph. The child transactions contained in the indirect dependency preserves the partial order of their parent transactions.*

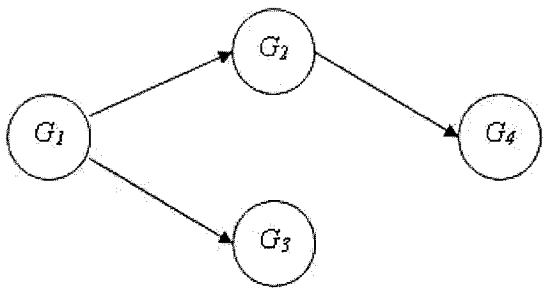

Figure 4.2: Parent dependency graph on p_1

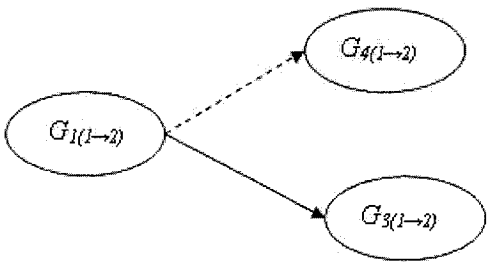

Figure 4.3: Child dependency graph on p_2

When a peer p joins a P2P MDBS, an *acquainted peer table*(APT) that is used to record its acquainted peers' information, such as peer ID, peer IP, peer Port, etc, is built in p. p updates its acquainted peer table each time when a new peer establishes a new acquaintance to it. At the same time, two empty dependency graphs $outCDG(p \rightarrow p_i)$ and $inCDG(p_i \rightarrow p)$ are set up for each acquainted peer p_i. The dependency graph $outCDG(p \rightarrow p_i)$ is constructed for the child transactions that p receives from the acquainted peer p_i, while the other dependency graph $inCDG(p_i \rightarrow p)$ is constructed for the child transactions that p spawns out to p_i.

In peer-to-peer computing systems, a peer acts as both a client and a server. In our protocol, two roles *client* and *server* are included in the peer database management system for each peer p. Client and server are two software modules that implement the concurrency control protocol. Both the client and the server participate in the coordination of transaction management between acquainted peers. The client is in charge of local transactions that are submitted to p from users, while the server is in charge of global transactions that p receives from p's acquainted peers.

When a user submits a transaction to the peer p, the client receives the transaction. The Transaction Manager in the client constructs a local transaction T_i using the local transaction received and assigns a unique transaction ID to T_i. Then T_i is sent to the peer database management system (PDBMS). The PDBMS processes the transaction. When the PDBMS commits T_i, the PDBMS updates the dependency graph of p and sends the commit message to the client. After the client receives the commit message of T_i, the client starts to translate T_i with respect to each of p's acquainted peer p_i. If T_i is translatable over the acquaintance between p and p_i, the client translates T_i into a child transaction $T_{i(p \rightarrow p_i)}$, adds T_i to $outCDG(p \rightarrow p_i)$, construct a global transaction message for $T_{i(p \rightarrow p_i)}$, and spawns the global transaction message of $T_{i(p \rightarrow p_i)}$ to p_i. The child dependency graph $outCDG(p \rightarrow p_i)$ is also encapsulated in the global transaction message of $T_{i(p \rightarrow p_i)}$. When pi receives $T_{i(p \rightarrow p_i)}$, it sends an acknowledgement message $ack(T_{i(p \rightarrow p_i)})$ back to p. The client always sends child transactions out by following the partial order that transactions are being processed so as to guarantee that the child transaction will be received in the destination peer.

All messages from other acquainted peers are sent to the server on p. The server on p sends back an acknowledgement to p_i as long as a global transaction message from p_i is received. At the same time, the server starts processing the global transaction message. First, the server partitions the message and constructs a global transaction G_i. By checking the global transaction's home transaction ID, the server identifies if the global transaction has been received or not. If G_i has not been received before, the server updates the child dependency graph $inCDG(p_i \rightarrow p)$ based on the child dependency graph

that is sent to p_i with G_i, and construct a new transaction ID for G_i by adding a unique ID of p to the be$G_i$nning of G_i's old transaction ID ; otherwise, the server just discards the global transaction message. Then, G_i is sent to the peer database management system and the PDBMS starts processing G_i. When the PDBMS decides to commit G_i, it goes to check the dependency graph $inCDG(p_i \rightarrow p)$ first. If all dependent transactions of G_i have been committed, then the PDBMS commits G_i, updates the dependency graph of p, and delete G_i from $inCDG(p_i \rightarrow p)$; otherwise, the PDBMS has to wait until all of G_i's dependency transactions commit. After G_i commits, a commit message of G_i is sent to the server. After the server receives the commit message of G_i, the server starts translating G_i with respect to every of p's acquainted peer p_j except the source peer p_i. If G_i is translatable to an acquainted peer $p_{j(j \rightarrow i)}$, the server translates G_i into a child transaction $G_{i(p \rightarrow p_j)}$, add G_i to $outCDG(p \rightarrow p_j)$, construct a global transaction message for $G_{i(p \rightarrow p_j)}$, and spawns the global transaction message of $G_{i(p \rightarrow p_j)}$ to p_j. The pseudocode for Client, Server, and some other methods summarizes our protocol.

Client:

```
do forever
  if( a transaction T from user is received)
      assign a transaction ID to T;
      submit T to the peer database management system;
      if (T commits)
          update the dependency graph of p;
          while(p has more acquainted peer) {
          if(T is translatable to the acquainted peer p_i) {
              T(p→p_i) = transactionTranslate(T, p_i);
              construct outCDG(p→p_i);
              spawn(T(p→p_i),outCDG(p→p_i), p_i);
              p waits until acknowledgement for T from p_i received
          endif
      endif
  endif
```

```
Server:
  do forever:
    if( a global transaction G from p_i is received)
        assign a transaction ID to G;
        duplicateExamine(G);
        if(G has not been received before)
            update inCDG(p_i→p);
            submit G to the peer database management system;
            if(p decides to commit G)
                check the inCDG(p_i→p);
                until all dependent transactions of G have been committed) {
                    commit G;
                    update the dependency graph of p;
                    while(p has more acquainted peer p_j except p_i) {
                        G_(p→p_j) = transactionTranslate(G, p_j)(j≠i);
                        construct outCDG(p→p_j);
                        spawn(G(p→p_j), outCDG(p→p_j),p_j);
                        p waits until acknowledgement for G_i from p_j received
                    }
                }
            endif
        endif
    endif

duplicateExamine(G):
    check the home_tid of G;
    if(a descendent transaction of the root transaction of G has been received) {
        return G has been received;
    else
        return G has not been received;
```

transactionTranslate(T, p_j):

```
for(each operation) {
    keep the same operator;
    for (each operand) {
        find the mapping value of the operand;
        substitute the operand by its mapping value;
    }
}
```

spawn($T_{(p \rightarrow p_i)}$, $outCDG(p \rightarrow p_i)$, p_i):

construct the global transaction message msg($p \rightarrow p_i$);
establish a connection with p_i's sever;
send msg($p \rightarrow p_i$) to p_i;
close the connection;

4.2 Correctness Proof of the Concurrency Control Protocol

Theorem 4.2.1 *A global history is globally P2P serializable if the propagation of the global history is acyclic and the global history is two-level serializable over each acquaintance in the propagation.*

Proof: Because the propagation of the global history is acyclic, then there exists no acquaintance cycle during the propagation of the global history. According to Theorem 3.2.2, we need to prove that our concurrency protocol ensures the two-level serializability over an acquaintance in order to prove the correctness of Theorem 4.2.1. Over an acquaintance, we use parent dependency graph and child dependency graph to control the ordering between a parent history and its child history. The child dependency graph is derived from the parent dependency graph. The nodes contained in the child dependency graph are a subset of transactions included in the parent dependency graph, and

this set of transactions preserve the partial order that their parent transactions are being executed. In our protocol, each child transaction commits after all its dependent transactions commits. This commit rule for child transactions ensures that all child transactions commit in the partial order of the child dependency graph. Thus, the set of child transactions preserves the ordering of their parent transaction. So the global history is two-level serializable over the acquaintance.

Therefore, the concurrency protocol ensures the global serializability of a global history if the propagation of the global history is acyclic based on Theorem 3.2.2.

4.3 Implementation

Since the prototype of a P2P MDBS has not been set up at the time of the writing of this thesis, we could not implement the concurrency protocol presented in the last section in a real P2P MDBS. We implemented a peer which includes the peer database management system to interact with the underlying local database system. The peer is a software module implemented in *Java* and we use *MySQL* as the underlying local database system. The implementation architecture is shown in detail in Figure 4.4.

As it is shown in Figure 4.4, a peer is composed of a client, a server, and the kernel information set up for the peer. Either the client or the server is implemented as a multithreaded process in Java. The client and the server share the peer's kernel information, which includes the dependency graph of the peer, the child dependency graph $outCDG(p \rightarrow p_i)$ for spawned transactions to each acquainted peer, and the child dependency graph $inCDG(p_i \rightarrow p)$ for received transactions from every acquainted peer. The peer starts by running the client's main thread and the server's main thread. The client and the server between acquainted peers communicate with each other by sockets. As long as the peer is active, the client keeps running to catch local transaction messages from local users, and the server also keeps running to listen and receive global transaction messages from other acquainted peers. When the client receives a local transaction, it

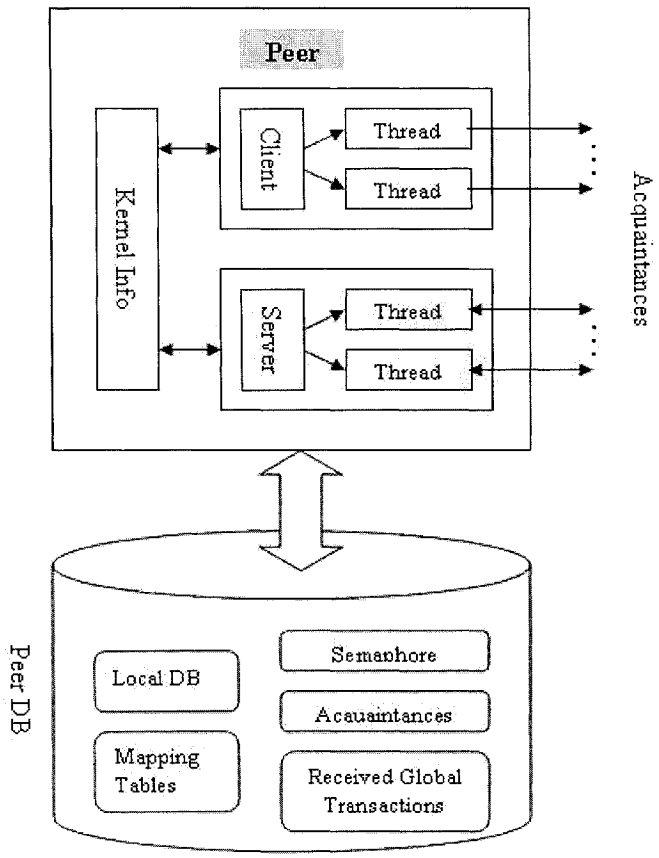

Figure 4.4: Implementation architecture for a Peer

starts a client thread to process the local transaction by following the client's protocol; when the server receives a global transaction from an acquainted peer, it starts a server thread to process the global transaction according to the server's protocol.

Since our protocol is not added in the transaction management system of MySQL, we are not able to extract the serializable information of transactions from the MySQL database management system. We used an auxiliary '*Semaphore*' table [18] to control the serializability of each peer. In each peer database system, an auxiliary 'Semaphore' table is constructed to serialize transactions. The 'Semaphore' table only contains one

attribute which is located by a single row and a single column. Each transaction updates that row before accessing other tables. In this way all transactions happen in a serial order [18]. Therefore, transactions, which are either local or global transactions, are processed in a serial order on each peer and this serial order is propagated to other peers with a transaction's propagation. The serial order is saved as the serializability, which represents the parent dependency graph in our implementation, in the kernel information of a peer. The serializability in the kernel information of the peer is updated either by the client or the server after each transaction commits.

4.4 Experimental Result

4.4.1 Experiment Setting

We tested our implementation in an airline P2P MDBS, which includes four airline peers and each peer is an airline company. Four peers *peerAC*, *peerKLM*, *peerLH*, and *peerUA* are created. Each peer has a table about the flights that it provides. The table structure in each peer is similar in order to simply the problem. We tested a construction of the airline P2P network without acquaintance cycle. The acquaintances of the airline P2P interest group is shown in Figure 4.5. The experiments show the propagation detail of global transactions in the P2P MDBS and the serializability of transactions on each participating peer. The schemas of these four peers are in Figure 4.6.

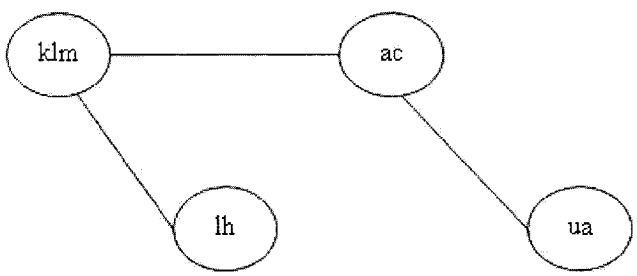

Figure 4.5: Airline P2P MDBS

ac(fno, date, time, dest)
klm(fno, date, time, dest)
lh(fno, date, time, dest)
ua(fno, date, time, dest)

Figure 4.6: Schemas for Airline P2P MDBS

fno	fno
UA1012	AC608
UA668	AC078
UA670	AC700
UA672	AC414
UA690	AC454

(a) fno2fno

dest	dest
YHZ	Halifax
BOS	Boston
CXH	Vancouver
YMQ	Montreal
YOW	Ottawa

(b) dest2dest

fno	fno
X	X

(c) date2date

time	time
Y	Y

(d) time2time

Figure 4.7: Mapping tables between *peerUA* and *peerAC*

Mapping tables between two acquainted peers include *fno2fno*, *date2date*, *dest2dest*, and *time2time*. The mapping tables *date2date* and *time2time* are all identity mapping tables. The mapping table *dest2dest* is also identity mapping over most of the acquaintances in the network except the acquaintance between two peers *peerUA* and *peerAC*. The mapping table *dest2dest* between *peerUA* and *peerAC* relates the code of an airport and the name of a city. The detail of all mapping tables between *peerUA* and *peerAC* are shown in Figure 4.7. Mapping tables over other acquaintances are similar to the mapping table between *peerUA* and *peerAC*.

4.4.2 Result

A set of five transactions are initiated on each peer. Transactions are initiated on peers in the order $LH \rightarrow UA \rightarrow KLM \rightarrow AC$. After a set of transactions is initiated on a peer, a period of 30 seconds is waited to initiate another set of transactions on the next peer in the order. Each set of transactions are fully translatable over every acquaintance in

the Airline P2P MDBSs. In Table 4.1 - 4.4, we show the transactions that are initiated on each peer. In table 4,5, we show the serializability of the 20 transactions on each peer. The result shows that the concurrency control protocol ensures the serializability of global histories;

Table 4.1: Transactions initiated on *PeerUA*

Transactions initiated on PeerUA:
begin insert into UA set fno='UA5526',date='10/08/2003',time='1;50',dest='YHZ' end
begin insert into UA set fno='UA1032',date='11/08/2003',time='6;00',dest='BOS' end
begin select fno from UA where dest='YHZ' end
begin insert into UA set fno='UA576',date='13/08/2003',time='6;00',dest='YMQ' end
begin select fno from UA where dest='BOS' end

Table 4.2: Transactions initiated on *PeerAC*

Transactions initiated on PeerAC:
begin insert into AC set fno='AC608',date='10/05/2003',time='06;45',dest='HALIFAX' end begin insert into AC set fno='AC078',date='10/05/2003',time='06;45',dest='BOSTON' end begin select fno from AC where dest='HALIFAX' end begin insert into AC set fno='AC414',date='10/05/2003',time='14;00',dest='MONTREAL' end begin select fno from AC where dest='BOSTON' end

Table 4.3: Transactions initiated on *PeerKLM*

Transactions initiated on PeerKLM:
begin insert into KLM set fno='KL1761',date='10/07/2003',time='07;00',dest='Halifax' end begin insert into KLM set fno='KL1361',date='11/06/2003',time='07;05',dest='Boston' end begin select fno from KLM where dest='Halifax' end begin insert into KLM set fno='KL0461',date='13/06/2003',time='07;10',dest='Montreal' end begin select fno from KLM where dest='Boston' end

Table 4.4: Transactions initiated on *PeerLH*

Transactions initiated on PeerLH:
begin insert into LH set fno='LH810',date='11/05/2003',time='06;15',dest='Halifax' end begin insert into LH set fno='LH1006',date='12/05/2003',time='06;15',dest='Boston' end begin select fno from LH where dest='Halifax' end begin insert into LH set fno='LH3728',date='14/05/2003',time='06;45',dest='Montreal' end begin select fno from LH where dest='Boston' end

Table 4.5: Serializability of transactions on every peer

Peer	Serializability
LH	LH0 ⟶ LH1 ⟶ LH2 ⟶ LH3 ⟶ LH4-KLM3-AC3-UA2 ⟶ LH5 ⟶ LH6-KLM5-AC4-UA4 ⟶ LH7-KLM7 ⟶ LH8-KLM8-AC6-UA6 ⟶ LH9-KLM9 ⟶ LH10-KLM10-AC9-UA7 ⟶ LH11-KLM12-AC11-UA10⟶ LH12-KLM11 ⟶ LH13-KLM13 ⟶ LH14-KLM14 ⟶ LH15-KLM15-AC14 ⟶ LH16-KLM16-AC16 ⟶ LH17-KLM17-AC17 ⟶ LH18-KLM18-AC18 ⟶ LH19-KLM19-AC19
KLM	KLM0-LH0 ⟶ KLM1-LH1 ⟶ KLM2-LH2 ⟶ KLM3-AC3-UA2 ⟶ KLM4-LH3 ⟶ KLM5-AC4-UA4 ⟶ KLM6-LH5 ⟶ KLM7 ⟶ KLM8-AC6-UA6 ⟶ KLM9 ⟶ KLM10-AC9-UA7 ⟶ KLM11 ⟶ KLM12-AC11-UA10 ⟶ KLM13 ⟶ KLM14 ⟶ KLM15-AC14 ⟶ KLM16-AC16 ⟶ KLM17-AC17 ⟶ KLM18-AC18 ⟶ KLM19-AC19
AC	AC0-KLM0-LH0 ⟶ AC1-KLM1-LH1 ⟶ AC2-KLM2-LH2 ⟶ AC3-UA2 ⟶ AC4-UA4 ⟶ AC5-KLM4-LH3 ⟶ AC6-UA6 ⟶ AC7-KLM6-LH5 ⟶ AC8-KLM7 ⟶ AC9-UA7 ⟶ AC10-KLM9 ⟶ AC11-UA10 ⟶ AC12-KLM11 ⟶ AC14 ⟶ AC13-KLM13 ⟶ AC15-KLM14 ⟶ AC16 ⟶ AC17 ⟶ AC18 ⟶ AC19
UA	UA0-AC0-KLM0-LH0 ⟶ UA1-AC1-KLM1-LH1 ⟶ UA2 ⟶ UA3-AC2-KLM2-LH2 ⟶ UA4 ⟶ UA5-AC5-KLM4-LH3 ⟶ UA6 ⟶ UA7 ⟶ UA8-AC7-KLM6-LH5 ⟶ UA10 ⟶ UA9-AC8-KLM7 ⟶ UA11-AC10-KLM9 ⟶ UA12-AC12-KLM11 ⟶ UA13-AC13-KLM13 ⟶ UA14-AC14 ⟶ UA15-AC15-KLM14 ⟶ UA16-AC16 ⟶ UA17-AC17 ⟶ UA18-AC18 ⟶ UA19-AC19

Chapter 5

Conclusion and Future Work

This thesis is the first touch to the problem of transaction management in P2P MDBSs. The objective of P2P MDBSs is to provide precise data manipulations to a P2P MDBS when user interacts with a peer database which is part of a network of peer databases. Similar architectures, such as traditional MDBSs and the ORCHESTRA system, are still trying to solve the problem of information sharing and exchange in the perspective of data integration. We started our work from analyzing and identifying the transaction properties in the P2P MDBS setting without any central control scheme. We preliminarily defined the local and global transactions that P2P MDBSs support. Actually, the type of transactions is interchangeable. Global transactions are always initiated as local transactions on their home peers. Because part of data items in two acquainted peers are precisely mapped to each other in mapping tables, we figured out how a transaction is translated and propagated over an acquaintance. A transaction spawns a child transaction to an acquainted peer when the transaction is translatable to that acquainted peer. The child transaction not only contains operations translated from its parent transaction, but also inherits the action (commit/abort) from its parent transaction. Because multiple descendent transactions are generated during a transaction's propagation, the global transaction is defined as a compound transaction which includes the root transaction and all its descendent transactions. Each component transaction of the global transaction is independent to each other because of the autonomy of each peer. There exists no value

dependency between any two component transactions contained in the global transaction. Meanwhile, each component transaction executes independently on the peer where it is submitted and the peer commits/aborts the component transaction unilaterally. All these properties for the global transaction fit in the Open Nested Transaction Model. Therefore we adopted the Open Nested Transaction Model as the transaction model and initially adjusted it suitable for the architecture of P2P MDBSs.

After setting up the transaction model, we started to solve the problem of concurrency control in P2P MDBSs. A global history on a peer p, which is defined with respect to a set of transactions submitted to p, consists of a root history on p and all its descendent histories. We take quasi-serializability [16] as our correctness criteria between a peer and its acquainted peers because P2P MDBSs follow all assumptions and conditions set up for the MDBS structure that fits the quasi-serializability [16]. Based on the correctness criteria, serial global histories and serializable global histories are defined. We say that a global history is serial if all descendent histories preserve the serial transaction ordering of the root history. Because transactions are propagated to other peers hierarchically and recursively from peer to peer over acquaintances, a two-level global serializability, that is defined with respect to a parent history and all its child histories, is utilized to solve the problem of concurrency control in P2P MDBSs. We have shown that a global history is serializable if and only if all component histories preserve the serializability of the root history. When the global history is propagated between two peers that contained in an acquaintance cycle, the global serializability of the global history may be violated since multiple descendent transactions of the same global transaction may arrive at the same destination peer. The serializability of the global history is preserved if and only if the global history is propagated along an acquaintance path between two peers that are not contained in any acquaintance cycle and global histories are two-level serializable over each acquaintance on the path.

Based on the serializability theorem, we developed a concurrency control protocol for P2P MDBSs by controlling the global transactions' two-level serializability over every

acquaintance. However, this protocol focuses on controlling global transactions' serializability, not on the actions' synchronization between a set of global transactions and the descendent transactions. The PDBMS starts spawning a global transaction's child transactions to destination peers after the transaction commits. Each child transaction is encapsulated in a single global transaction message to ensure the atomicity of the child transaction.

Finally, we implemented the concurrency control protocol in a simulated P2P MDBS. The experiment results of our implementation show that the global transactions' serializability is ensured by the protocol.

In this thesis, we solved some of the basic problems related to the transaction management in P2P MDBSs. There are still some problems unsolved as far as maintaining the consistency of a P2P MDBS is concerned:

1. We discussed that some level of inconsistency can be tolerated in P2P MDBSs. The inconsistency that can be tolerated varies in different P2P MDBS application domains, i.e., the P2P MDBS inconsistency is application-domain-dependent. Further work is needed to generalize and formalize the levels of inconsistency acceptable for given P2P MDBSs.

2. We introduced the concept of indirect acquaintance between peers. Two peers relate to each other when global transactions propagate between two indirectly acquainted peers over an acquaintance path. We need to identify the correctness criteria for two peers that are indirectly acquainted to each other.

3. Intuitively, a global transaction may be partially translated over an acquaintance in its propagation. Then the atomicity of the global transaction is violated when descendent transactions are partial translations of the transaction. We need to precisely analyze what kind of partial transaction translations are acceptable in P2P MDBSs, and how the consistency of the global transaction can be relaxed and ensured.

Bibliography

[1] Thomas M. Connolly and Carolyn E. Begg, *Database Systems: A Practical Approach to Design, Implementation, and Management*, Addison-Wesley Publishing Company, Reading, Massachusetts, 2003.

[2] Yuri Breitbart, Hector Garcia-Molina, and Abraham Silberschatz, "Overview of multidatabase transaction management," *VLDB Journal: Very Large Data Bases*, vol. 1, no. 2, pp. 181–293, 1992.

[3] Marcelo Arenas, Vasiliki Kantere, Anastasios Kementsietsidis, Iluju Kiringa, Rene J. Miller, and John Mylopoulos, "The hyperion project: from data integration to data coordination," *SIGMOD Record*, vol. 32, no. 3, pp. 53–58, 2003.

[4] Karl Aberer and Manfred Hauswirth, "Peer-to-peer information systems: concepts and models, state-of-the-art, and future systems," in *ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, New York, NY, USA, 2001, pp. 326–327, ACM Press.

[5] V. Kantere, I. Kiringa, J. Mylopoulos, A. Kementsietsidis, and M. Arenas, "Coordinating peer databases using eca rules," 2003.

[6] A. Kementsietsidis, M. Arenas, and R. Miller, "Mapping data in peer-to-peer systems: Semantics and algorithmic issues," 2003.

[7] Panos K. Chrysanthis and Krithi Ramamritham, "Synthesis of extended transaction models using ACTA," *ACM Transactions on Database Systems*, vol. 19, no. 3, pp. 450–491, 1994.

[8] Christoph Schuler Can Trker, Klaus Haller and Hans-Jrg Schek., "How can we support grid transactions? towards peer-to-peer transaction processing.," 2005.

[9] Aneesh Kapur Murat Cakir Zachary G. Ives, Nitin Khandelwal, "Orchestra: Rapid, collaborative sharing of dynamic data," 2005.

[10] Yuri Breitbart, Avi Silberschatz, and Glenn R. Thompson, "Reliable transaction management in a multidatabase system," in *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 1990, pp. 215–224, ACM Press.

[11] K. BARKER, "Transaction management on multidatabase systems," 1990.

[12] P. Chrysanthis and K. Ramamritham, "Correctness criteria and concurrency control," 1998.

[13] H. K. Korth and G. Speegle, "Formal model of correctness without serializabilty," in *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 1988, pp. 379–386, ACM Press.

[14] Amit P. Sheth, Marek Rusinkiewicz, and George Karabatis, *Using Polytransactions to Manage Interdependent Data*, chapter 14, pp. 555–581, Morgan Kaufmann, San Mateo, California, U. S. A., 1992.

[15] Philip A. Bernstein, Vassco Hadzilacos, and Nathan Goodman, *Concurrency control and recovery in database systems*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987.

[16] W. Du and A. Elmagarmid, "Quasi serializability: a correctness criterion for global concurrency control in interbase," in *VLDB '89: Proceedings of the 15th international conference on Very large data bases*, San Francisco, CA, USA, 1989, pp. 347–355, Morgan Kaufmann Publishers Inc.

[17] Vassos Hadzilacos Philip A. Bernstein and Nathan Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1987.

[18] "How to cope with deadlocks," http://dev.mysql.com/doc/refman/5.0/en/innodb-deadlocks.html.