Schema Mediation and Query Processing in Peer Data Management Systems

by

Jie Zhao

B.Sc., Fudan University, 2003

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

The Faculty of Graduate Studies

(Computer Science)

The University Of British Columbia

October 2006

Abstract

P2P Data Management Systems (PDMSs) allow the efficient sharing of data between peers with overlapping sources of information. These sources share data through semantic mappings between peers. In current systems, queries are asked over each peer's local schema and then translated using the semantic mappings between peers. In this thesis we propose that a mediated schema can benefit PDMSs by allowing access to more data and by making that access comprehensible. We present our system - MePSys, which uses the mediated schema as a media for query translation in relational PDMSs. We show how to create a mediated schema in PDMSs automatically using the semantics of the existing mappings provided to translate queries. We further discuss how to update the mediated schema in a stable state, i.e., after the system setup period.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank all of people giving me support and help during my study at UBC. Without your support, this thesis cannot be done.

First of all, I would like to thank my supervisor, Rachel Pottinger, for always giving me support, encouragement and excellent guidance throughout my thesis work. I spent a wonderful time working with Rachel for the past one and half years.

I would like to thank George Tsiknis for dedicating time and effort in reviewing my thesis and giving me valuable comments and advise.

I would like to thank all my friends in the department and in the database lab for your continuous help, and especially the following people: Shaofeng Bu, Gang Peng, Xun Sun, Shuan Wang, Wendy Wang, Jian Xu, Chao Yan, Suling Yang and Xiaodong Zhou.

Last, I would like to thank my parents and Alex Liu for their endless love and support.

Jie Zhao
October 2006

Chapter 1

Introduction

1.1 Overview

A peer-to-peer (P2P) network is a network in which every participating node in the network provides power, bandwidth and other resources rather than only relying on a small number of servers. It is a decentralized and distributed system. Compared to a client-server architecture where the number of servers are fixed, P2P is more flexible and extensible because when new nodes arrive, the total capacity of the system increases accordingly while in client-server environment, the adding of new clients means the slowing down of the whole system. P2P systems are very useful and famous nowadays among users who want to share files while users are everywhere across the world. For example, Napster [3] is a successful system for music sharing using P2P infrastructure.

A Peer Data Management System (PDMS) (e.g. [7, 8, 14, 24]) is the result of blending the benefits of peer-to-peer (P2P) networks, such as lack of a centralized authority, with the richer semantics of a database. Peer Data Management System (PDMS) is a Data Management System using P2P architecture. Each peer in the system holds a database. It can be extensively used for data exchanging, query answering, information sharing, etc. In the areas of scientific research, the idea of setting up a PDMS for research in the related area to share data among peers has already been widely discussed and admitted.

A PDMS, as a P2P system itself, keeps the properties of all P2P systems: Every peer may join and leave the network at any time. All peers are autonomously created and managed. However, A PDMS also requires that each peer, upon entering the network, publishes its database schema so that

it can be seen by other peers in the network, and the entering peer must create a mapping between itself and one or more neighbors. We call peers between which there exists a direct mapping *acquaintances*. These inter-peer mappings allow query translation schemes [8, 24, 26] which allow users to easily, semantically query data from sources that are not their own. PDMSs are a data management architecture where not all nodes in the system need to be there constantly.

PDMSs are often compared with a Data Integration System (DIS) which uses a client-server architecture. A major differences between a DIS and PDMS is the centralized control. In a traditional DIS, there is always one or more central servers to store a global view for all clients' database schemas. Queries are typically posed over the global schema and then translated to local schemas. All the query processing work such as query translation, query forwarding, receiving answers, and translating answers entirely relies on those servers. Thus, if the servers break down, or become unavailable, the whole system will fail. Considering the workload of the whole system, the centralized control definitely results in a severe bottleneck. On the other side, a PDMS is a dynamic and loosely-coupled environment. There are no central servers in the PDMS. Every peer is both a server and a client. Even though, some nodes might take more responsibilities, e.g., super-peer-structured P2P system, these nodes are still not taking the responsibilities of controlling all nodes, and the resulting architecture is more decentralized. In most current PDMSs, a query is posed over a local peer schema, and answers will be computed in this peer database. The query will also be forwarded and reformulated to other peers in the network, and query results will be sent back. The final result set returning to the user will be answers from the local peer as well as those from other peer databases.

The goal of this thesis is to study the properties of a PDMS, compare and survey existing PDMSs, and look for more opportunities to improve the current PDMS design to let the users have a better understanding of the query processing result and to speed up the query reformulation process among peers.

1.2 A Motivating Example of PDMS

Given the above properties of flexibility and decentralization, a PDMS is very suitable for scientific research data sharing and exchange [21]. There are many scenarios where a PDMS would be useful. In bioinformatics this has already been studied from both the data management approach [17, 21] and also from the bioinformatics approach [25]. Imagine that the Biology labs from different universities are doing research in similar areas. Some might have a partial gene sequence data of a sparrow, and some might have another partial gene sequence data of a sparrow. Researchers from either lab wants more data for further experiment. Even though their results have been published, the data still need some way to be shared by others. Thus a PDMS where each peer comes from a biology lab doing similar gene expression research can handle such a task well. In general, the types of scenarios that we can imagine that would benefit the most from a PDMS are those scenarios where people have overlapping structured data, and they want to access other sources' additional structured information.

Rather than relying on readers having an understanding of the intricacies of biological data, we present a scenario which is familiar to readers of Computer Science area: bibliography management in Example 1.

Example 1 Assume that there are two database servers of bibliography information: UBC and UW, and that they have the following relations storing information about conference papers:

UW.conf-paper(title, venue, year, pages)
UBC.conf-paper(title, conference, year, url)

As in this example, it is very likely that each database server only stores part of the record about conference papers, e.g., only UW has page information and only UBC has url. Additionally, even though they may have overlapped record for the same paper, the content might still be different. For instance,("Data Management for Peer-to-Peer Computing: A Vision", "Madison, Wisconsin, USA", "2002", 6) is one data entry in UW.conf-paper;

("Data Management for Peer-to-Peer Computing: A Vision","WebDB", "2002", "citeseer.ist.psu.edu/bernstein02data.html") is the data entry for the same paper in UBC.conf-paper. Given the similarity of relations above, along with the difference in available data, it would be beneficial for users of the two databases to be able to share their data on conference papers.

□

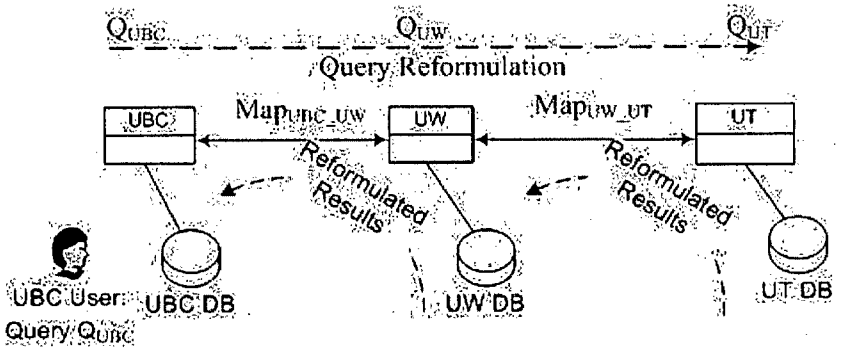


Figure 1.1: Query Processing in traditional PDMS

Figure 1.1 is a picture to explain how queries are propagated and translated back in the P2P systems. In most PDMSs (see figure 1.1) a user at peer UBC will ask a query posed over UBC's schema. Answers are computed at UBC, and the query is also reformulated and forwarded to other peers through mapping paths in the network. For example, assume UBC has a mapping Map_{UBC_UW} to UW. Using Map_{UBC_UW}, a query Q_{UBC} will be asked over UBC and then reformulated to Q_{UW}, a query over UW's schema. Q_{UW} will be processed at peer UW. UW will reformulate Q_{UW} to query over any additional acquaintances' schemas in the same way. Thus queries can be forwarded as far along the mapping path as possible, subject to system constraints. Query results will be sent back to the peer UBC after a similar series of reformulations. Thus, the user at UBC will get answers not only from local database but from other peer database as well.

1.3 Challenges and Contributions

The methods in Example 1 have been well studied and allow translation and answering of a large class of queries [26]. However, when the full context is looked at, there are additional opportunities for improving the usefulness and comprehensibility of query translation in PDMSs, which we tackle in this thesis.

First, we get rid of the restrictive query format. In previous methods, only information supported by the local schema can be queried. Further, a user at a peer cannot know what other information is available in the network. The user can only see the local schema, despite the fact that actually there is much more information in the PDMS. This is clearly suboptimal; since the information is present, users should be able to fully utilize these resources and not be restricted by the format of local schemas. For example, in Example 1, users at UBC would be unable to access the page information at UW because they can only ask queries over their own schema.

Second, we improve the comprehensibility of the PDMS. Current systems can either fail to take into account some expressiveness available to the system, or result in systems that are difficult to understand, as is shown in Example 2:

Example 2 Consider the following mappings, where (2.1) relates source A to target B, and (2.2) relates source B to target C. $\subseteq$ means the left-hand side is a subset of the right-hand side. Attributes in different sources that can be mapped to each other are using the same variable names.

$$\mathsf{A.grandpa}(x, y) \subseteq \mathsf{B.father}(x, x_1), \mathsf{B.father}(x_1, y) \tag{2.1}$$

$$\mathsf{B.father}(x, x1), \mathsf{B.father}(x_1, x_2), \mathsf{B.father}(x_2, y) \subseteq \mathsf{C.greatgrandpa}(x, y) \tag{2.2}$$

This is isomorphic to a mapping in [20]. (2.1) says that if x is the grandpa of y, then there is some x_1 such that x is x_1's father, and x_1 is y's father. Mapping (2.2) says that if x and x_1, x_1 and x_2, x_2 and x_3 all have father relationship, then x and y will be the great grandpa of y. Using the

algorithm described in Piazza [20], composing mappings (2.1) and (2.2) to map source A to source C results in:

$$\mathsf{A.grandpa}(x, x_2), \mathsf{A.grandpa}(x_2, y) \subseteq \mathsf{C.greatgrandpa}(x, y_1) \qquad (2.3)$$

(2.3) says that if x is x_2's grandpa and x_2 is y's grandpa, then x is somebody's great grandpa. However, (2.3) does not express any constraint on y_1: from the given information, it is clear that y_1 is the father of y and x_2 is the father of y_1; Piazza is not expressive enough to express that restriction. While the mapping language used by Fagin et al. [10] has enough expressive power to handle the same example, the final results using [10]'s algorithm are difficult to comprehend. The reason for this is that [10] uses second-order logic dependencies to express their mappings. Thus for all those mappings with *unbound attributes* (attributes that appear in the target but do not appear in the source) in the first-order logic, e.g. x_1 in (2.1), [10]'s mapping language can use $\exists$ and $\forall$ to explain the constraints for these attributes. However, when there are too many constraints in the mapping, the readability of the mapping itself will be decreased.

□

In our system, we make the system more comprehensible by providing a comprehensible mediated schema, and each local schema will be related to the mediated schema so that users of both the local schema and the mediated schema can easily understand the system.

Third, we provide more semantic query processing. In traditional P2P file systems, copies of the same file on different peers can be viewed as grouped into the same file; each may contain different parts of the original file. Consider the scenario of a PDMS. Data of the same topic (which is expressed as "*Concept*" in this thesis) are dispersed among peers. When data are grouped according to the topic in advance, it will be very easy to help the peers exchange data and make the query processing fast.

Fourth, we reconsider the difficulties and complexity raised by mapping composition, and see whether such complexity can be avoided in real systems.

Fifth, we make good use of indirect mapping information. We call the information found in the composed mapping that is not included in the direct mapping *indirect mapping information.* Previous methods tend to reformulate queries based on single sources of mapping information. However, in real systems, it is very likely that different mapping routes can complement each other for query reformulation. Query answers will be improved a lot if we consider different mapping routes and indirect mapping information.

Taking into consideration of all these opportunities, this thesis only considers the mappings that can group data into topics, and proposes automatically creating and maintaining a mediated schema to take advantage of the above insights to improve functionality for PDMSs. For other mappings, the same method proposed in Piazza is adopted in our system. This has many advantages. First and foremost, users are able to access more information residing in the network than in previous systems. Second, we create the mediated schema using the pre-existing mappings, thus users have the flexibility to use either the mediated schema or use their local schema as they use previously existing methods, (e.g., those in Piazza [26]). Third, using our method, the translation of queries over the mediated schema into the sources is simple and effective. Finally, as will be shown in the experiment, our query processing method using a mediated schema is much faster than other methods, which won't slow down the whole system but will ensure a comprehensible PDMS.

However, creating a mediated schema in a PDMS setting requires answering difficult questions such as:

- How can the mediated schema be created without a centralized authority?
- How can an automatically created mediated schema be comprehensible to users?
- Since the system is dynamic, the maintenance of a mediated schema must be automatic and adaptive; how can human intervention be minimized?

- Where is the mediated schema stored, and how is it updated?

In this thesis, we begin to answer the above problems. We make the following specific contributions:

- We describe our PDMS system, MePSys (Mediation supported Peer Data Management System), in which a mediated schema can be created dynamically and any information in the network can be queried without requiring any additional global services in the network.
- We outline our Peer Schema Mediation algorithm (PSM algorithm) which is used to efficiently create the mediated schema representing all schemas in the network, and introduce MappingTables as a mechanism for relating the mediated schema to the sources.
- We study the semantics of mappings and introduce the idea of automatically detecting specific *Concepts* in mappings. A mediated schema based on *Concepts* is much easier for users to understand.
- We study how mapping composition impacts query reformulation and how to discover common *Concepts* in composed mappings to ensure comprehensible query results.
- We study how to solve the problem of updating the mediated schema in our infrastructure.
- We design experiments to test the efficiency and scalability of MePSys.

This thesis is organized as follows. In Chapter 2, we introduce background knowledge related to this thesis. We discuss related works in Chapter 3. In Chapter 4, we define and analyze the semantic information that can be presented in a conjunctive mappings. We present our approach to creating a mediated schema in a PDMS and creating the mappings from the mediated schema to local sources in Chapter 5, including algorithms and examples. We further describe how to update the mediated schema in Chapter 6. We show our system implementation setup and the experimental

results in Chapter 8. We then study the mapping composition problem that can impact the correctness and capability of query translation in the system in Chapter 7. Chapter 9 concludes and discusses future work.

Chapter 2

Preliminaries

2.1 Definitions w.r.t. Mediated schema creation

Definitions in this section are related to the mediated schema creation.

2.1.1 Datalog

Datalog, a subset of Prolog, is a query and rule based language. It is a logic-based query language for the relational model.

Take the following clause as an example:

Example 3 grandpa(x, z) :- father(x, y), father(y, z)

□

Each clause in Datalog is composed of two parts: head and body, which are separated by ":-". In Example 3, grandpa(x, z) is the head, and father(x, y), father(y, z) are collectively the body. The body can be viewed as conditions known, and the head can be regarded as the query. The above clause can be interpreted as: if x is y's father and y is z's father, then x is z's grandpa. Each relation (e.g. father) is called a predicate. Each term in the predicate (e.g., x and y) can either be a variable or a constant. For safety, only variables that appear in the body can appear in the head.

2.1.2 Mediated Schema, LAV, GAV and GLAV

A data integration system is a system that combines data residing at different sources and provides the user with a unified view of these data. A mediated schema, also called a global schema, is usually provided in a Data Integration System. Every local source has its own database schema, which is called the local schema. A mediated schema is a unified view of all these

local databases so that through the mediated schema, users can query over only the mediated schema and get results from all local sources.

There are two basic approaches in a data integration system to specify mappings between the mediated schema and the local schemas: Local-As-View (LAV) and Global-As-View (GAV). In LAV, the content of each local source s should be characterized in terms of a view q_G over the global schema; in GAV, the content of each element g of the global schema should be characterized in terms of a view q_S over the local sources [19].

Example 4 To explain these definitions, consider Example 1 again. Two local databases UW and UBC are presented with the database relations as below.

UW database:
UW.conf-paper(title, venue, year, pages)

UBC database:
UBC.conf-paper(title, conference, year, url)

A mediated schema M is manually created as:
M.conf-paper(title, conference, venue, year)

Using LAV, UW.conf-paper and UBC.conf-paper will be explained as a view of the mediated schema:

UW.conf-paper(title, venue, year, pages) :-
M.conf-paper(title, conference, venue, year)
UBC.conf-paper(title, conference, year, url) : -
M.conf-paper(title, conference, venue, year)

Using GAV, M.conf-paper can be expressed as the view of all local schemas.

M.conf-paper(title, conference, venue, year) :-
UW.conf-paper(title, venue, year, pages)
M.conf-paper(title, conference, venue, year) :-
UBC.conf-paper(title, conference, year, url)

□

GLAV, firstly introduced in [12], is a more generalized approach to specify mappings between sources. GLAV mapping language adopts both GAV and LAV in its presentation, which allows a flexible schema definition and combines the expressive power of both GAV and LAV. Example 5 shows an example of GLAV mapping relating different schemas.

Example 5 Assume UW and UBC have the following schemas.

UW database:

UW.research-area(area, groupleader, department)
UW.grad-student(stu-name, area, advisor)

UBC database:

UBC.faculty(name, office, department)

A mediated schema M is manually created as:

M.people(name, department)

Using GLAV, relationship between the source schema and the mediated schema can be freely expressed.

GLAV mapping between UBC and M

UBC.faculty(name, office, department)
→ M.people(name, department)

GLAV mapping between UW and M

UW.grad-student(stu-name, area, advisor),
UW.research-area(area, gl, department)
→ M.people(name, department)

The above GLAV mappings can also be rewritten in conjunctive mappings.

GLAV mapping between UBC and M
LAV:
Q_1(name, department):-M.people(name, department)
GAV:
Q_1(name, department):-UBC.faculty(name, office, department)

GLAV mapping between UW and M
LAV:
Q_2(name, department):-M.people(name, department)
GAV:
Q_2(name, department):-UW.grad-student(name, area, advisor),
UW.research-area(area, gl, department)

□

2.1.3 Query and Mapping

A mapping is the medium for exchanging data and reformulating queries among different schemas; in particular, the mapping defines the overlapping parts of acquaintances' schemas. As such, it is the most basic and important part of the system. We use conjunctive mappings [27] as our mapping language. These are the same mappings as in [22] and [20]. We choose conjunctive mappings as our input mapping because they can easily express commonality among different schemas and allow the use of existing algorithms if the mediated schema is not used, e.g., through the methods in [26] which just compose the mappings between peers. A *conjunctive mapping* is a set of *conjunctive queries* relating a pair of local schemas: i.e., a set of simple Datalog queries. We briefly review the syntax of conjunctive queries in Example 6. In a conjunctive mapping, if a set of conjunctive queries has the same IDB name (i.e., name of the answer relation), that set expresses the overlapping information in the sources. Note that we still use mapping to refer to the general sense of correspondences between schemas. Each conjunctive query in the conjunctive mapping is also called a *component*, which relates to one schema for that mapping. In a conjunctive query, variables

that appear in both the head and the body are called *distinguished variables*; variables that appear only in the body are called *existential variables.*

Example 6 Consider the following conjunctive mapping on the schemas in Example 4:

conf-paper(title,venue,yr):-UW.conf-paper(title,venue,yr,pages)
conf-paper(title,venue,yr):-UBC.conf-paper(title,venue,yr,url)

This mapping shows how UBC and UW can share information about conference papers through the reuse of the IDB name conf-paper. Each line is a conjunctive query. conf-paper(title,venue,yr) is the head of both queries, and title, venue, and yr are head variables, indicating that these are the common attributes in the two schemas. As in general Datalog, the semantics imply that conf-paper information can be found by taking the union of title, venue, and yr attributes found from the bodies of the two queries. The bodies express the conditions required to form an answer tuple, and reuse of a variable indicates that those values must be the same. Each relation in the body of a query is referred to as a subgoal. Together, the conjunctive queries show that information about conf-papers can be found through either UW.conf-paper or UBC.conf-paper, and all information that can be obtained from the topic conf-paper are title, venue, yr, page and url information. Note that the IDB name of a subgoal is the subgoal name. □

2.2 Definitions w.r.t. PDMS

2.2.1 Query Reformulation and Result Reformulation

Query reformulation is the process of translating a query over database schema A to that over database schema B so that queries over A can be understood by schema B. The reformulation process follows the mappings between these two schemas. Accordingly, result reformulation is the process of translating the results using database schema A to those over database schema B so that results obtained at schema A can be understood by schema B.

2.2.2 Mapping Composition

Mapping composition is a hard problem which draws much attention recently. Intuitively, when given semantic mappings M_{12} from schema A to B, and M_{23} from B to C, we hope we could compute a direct mapping M_{13} from A to C which is equivalent to the composition of mappings M_{12} and M_{23}.

Halevy et al. provided a formal definition of mapping composition: "The mapping $M_{A\to C}$ is a composition of the mappings $M_{A\to B}$ and $M_{B\to C}$ w.r.t. a query language $\mathcal{Q}$ if for all databases D_A for R_A, and for all queries q over R_C such that q is in the language $\mathcal{Q}$, the certain answers for q w.r.t. $M_{A\to C}$ are the same as the certain answers w.r.t. $M_{A\to B}$ and $M_{B\to C}$." [20]

Fagin et al. defined the mapping composition in more logical sense: "Let M_{12} =(S_1, S_2, Σ_{12}) and M_{23} =(S_2, S_3, Σ_{23}) be two schema mappings such that the schemas S_1, S_2, S_3 have no relation symbol in common pairwise. A schema mapping M = (S_1, S_3, Σ_{13}) is a composition of M_{12} and M_{23} if Inst(M) = Inst(M_{12}) $\circ$ Inst(M_{23}), which means that Inst(M) = {<I_1; I_3> | there exists I_2 such that <I_1; I_2> $\in$ Inst(M_{12}) and <I_2; I_3> $\in$ Inst(M_{23})}." "A schema mapping is a triple M = (S, T, Σ_{st}), where S and T are schemas with no relation symbols in common and Σ_{st} is a set of formulas of some logical formalism over <S, T>." [11]

Mapping composition is a problem that costs high complexity when considering all classes of mappings. In Chapter 5, we explore and analyze mapping composition problem in more depth.

Chapter 3

Related Work

3.1 Peer Data Management Systems

For a long time, P2P computing and Database research groups were very independent. The idea of incorporating the database research into P2P was proposed and discussed early this century by Gribble et al. [13] in 2001, Bernstein et al. [7] in 2002. The paper by Bernstein et al. introduces the Local Relational Model (LRM) as a data model specifically designed for P2P applications. Most research projects in PDMS build their architecture by extending LRM, (i.e. Piazza [14], HePToX [8], Hyperion [24]). LRM assumes that the set of all data in a P2P network consists of local (relational) databases, each with a set of acquaintances, which define the P2P network topology. For each acquaintance link, translation rules between data items and semantic dependencies between the two databases are predefined [7]. Our system also uses such a model.

Piazza [14, 15, 20] is a well-known prototype for Peer Data Management System. Piazza uses GAV/LAV style mappings to describe the semantic relationships between two peers. Our mapping is based on the Piazza mapping, but changed into conjunctive mapping for better understanding. Piazza also provides a query reformulation algorithm [26] based on XQuery to translate the queries among peers using different schemas. Unlike our solution, they do not support a mediated schema in their prototype, which is flexible for query translation. Our solution uses the similar initial setup but ensures a better understanding of the system, access to more information and a faster query reformulation. However, since our system uses the same mappings as Piazza, users of our system could choose to use either our query answering system or translate their queries using Piazza's system.

HePToX [8, 9] is a P2P XML database system. A novel query translation algorithm has been developed for a simple but a significant fragment of XQuery. Both Piazza and HePToX assume that all queries are asked over a local schema; they focus on finding a query reformulation algorithm to translate a query over one peer schema to a query over its acquaintance.

The works of Fagin et al. [10, 11] mainly consider how to process data exchange and mapping composition without information loss using mappings under second-order logic dependencies. They provide a nice solution for mapping composition which they uses for their query translation.

Hyperion [5, 24] is also a Peer Data Management System which uses both data-level and schema-level mappings to specify the correspondences between acquainted peer databases and process query reformulation based on these mappings. Mapping tables are used to specify correspondences between data values of acquainted databases. This is the key difference between Hyperion and all other systems.

In PeerDB [21], local peer schemas are not published. Instead, local users input keywords from relation name, attributes, and records for each peer database, and such information is used to match relations.

As in previous PDMSs: a new peer entering the network chooses one or more acquaintances and provides mappings to one or more acquaintances, either created by hand, or through some schema matching tool (see [23] for a survey). Our system has the following minimal features:

Peer: each peer stores both its local schema and one or more versions of mediated schema that local peer has an application built on. Only the mappings from mediated schema to its local schema will be stored at this local peer, which is used for query answering. Each peer will also store MappingTables, which help create the mediated schema and determine how to relate the mediated schema to the peer's schema.

Query answering: users can either query the local schema (as in previous systems) or use the automatically-created mediated schema, which provides additional information. In either case, user queries are automatically reformulated and forwarded to the peers' acquaintances.

3.2 Mediated Schema Creation

Another aspect of our research is mediated schema creation (MSC). MSC has been studied in data integration area. Batini, Lenzerini, and Navathe [6] conducted a survey for database schema integration early in 1986. In [6], methodologies for schema integration and a comparison of all available methodologies are provided. Later, Lenzerini [19] and Ullman [28] both discussed data integration techniques in very theoretical perspective. However, most early work ignores creating the mappings between the mediated schema and the sources.

Pottinger provided a method for creating a mediated schema and mappings from the mediated schema to sources [22]. However, [22] does not consider the complications of the PDMS situation, including the fact that mappings can exist between any pairs of schemas, rather than those dictated by a centralized authority. Our schema mediation method is based on [22]. However, we have the following improvements: our MSC method can handle any numbers of local schemas, while the one in [22] can only handle a limited number of local sources. In our approach, any ordering of mediating local schemas will get an equivalent result while in [22], different ordering of local schemas might get totally different mediated schema and mappings to local sources. This is let's assume that we use a different approach from theirs. For example, when we have three peers A, B, C mapping Map_{A_B} between A and B, and mapping Map_{B_C} between B and C. Pottinger's approach to create a mediated schema M_{AB} is firstly based on schema A, B and mapping Map_{A_B}. Then mappings Map_{AB_B} and Map_{B_C} are composed to get the mapping Map_{AB_C}. Using schemas M_{AB}, C and mapping Map_{AB_C}, the mediated schema of ABC can be created. Using our approach, we create the intermediate mediated schemas M_{AB} and M_{BC} first. Then we create a mapping Map_{AB_BC} based on the overlapped subgoals in the mappings M_{AB} and M_{BC}, and create the final mediated schema and mappings to local sources.

Chapter 4

Introducing Concept into Conjunctive Mappings

Before we delve into the PSM algorithm, we start by studying two related problems: what are the things that can be represented by the same relation in the mediated schema, and what is the complexity of mapping composition in such a system. We explore the first problem in this chapter, and explore the second one in chapter 7.

The definitions of Conjunctive Mapping and Conjunctive Query have been introduced and explained in section 2.1.3. In this chapter, we explore how to support semantics in the conjunctive mapping by applying some restrictions on the conjunctive mappings. We first present the motivation of introducing the idea of "*Concept*" into conjunctive mappings in section 4.1. We then use some examples to analyze what kind of restrictions should be applied to the mappings in order to let the mappings have a *Concept*. We give the formal definition of a *Concept* in section 4.2.

4.1 Motivation

One of our contributions is to introduce the notion of *Concept* into conjunctive mappings. Previous work either ignores the semantics that the mappings might contain [11, 20] or simply assumed that if the IDB name of the queries were the same, then they described the same *Concept* [22]. The later view, though using the semantic information that is embedded in the mappings, is overly simplistic. However, reusing the same IDB name inside a conjunctive mapping likely indicates the cases that (1) the semantics of the mapping may make it impossible to construct an understandable mediated schema if this is the only information taken into account and (2)

consistency of IDB names cannot be assumed between mappings - there is no guarantee that the creator of Map_{A_B} will use the same IDB name in the same way as the creator of Map_{C_D} to represent the same *Concept*. In this thesis we begin to understand the semantic issues entailed by *Concept*. However, as an introductory step, throughout this thesis we assume that when more than one conjunctive mappings express the same *Concept*, their IDB names are the same.

Intuitively, a *Concept* describes the common object among different schemas. This is different from a mapping which expresses the overlapped attributes across schemas. When a mapping is said to have a *Concept*, the mapping should have a clear object, which might have some aspects to describe it in detail. In Example 6, clearly the mapping describes the object "conf-paper", which uses five aspects, title, venue, yr, pages and url, to describe this object. However, even though there are common relation names (e.g., father) and attributes (e.g., x, x1) occurred in Example 2, these two mappings together do not express a common object or an idea because mapping (2.1) explains the *Concept* grandpa and mapping (2.2) explains the *Concept* greatgrandpa.

Deciding whether or not one mapping can express a *Concept* evolves much AI research topics such as natural language processing, ontology, etc, and thus is not our focus in this thesis. However, deciding whether two mappings with the same IDB name together can express the same *Concept* is very useful, especially when we want to mediate local schemas and propagate queries along the mapping path. There are two reasons. First, users tend to provide incomplete or wrong mapping information when two mappings between two pairs of peer schemas are considered. Second, we want to know whether the pre-defined mappings are truly relating identical parts from different schemas. In this thesis, we only discuss this *Concept* problem based on two or more mappings.

Throughout this thesis, the input mappings for MePSys are required to be conjunctive mappings with the same concept (Definition 3) without self-joins in each component. The PSM algorithm is sound and complete for such input mappings.

4.2 Definitions and Analysis

In Chapter 2, we introduced conjunctive mappings, including Example 6, which defines one *Concept*: conf-paper. However, in many cases determining a *Concept* from the mapping is not straightforward. Consider Example 7 (originally from [11] but rewritten in the form of a conjunctive mapping).

Example 7 Assume the following conjunctive mappings:
Conjunctive Mapping 1: between A and B

```
Manager(x, y) :- A.Mgr(x, y)                    (7.1a)
Manager(x, y) :- B.Mgr1(x, y)                   (7.1b)
```

Conjunctive Mapping 2: between B and C

```
Manager(x) :- B.Mgr1(x, x)                      (7.2a)
Manager(x) :- C.SelfMgr(x)                      (7.2b)
```

Both conjunctive mappings use the IDB name Manager. However, while the first conjunctive mapping defines the Manager relationship, the second does not. Though the relation Mgr1 appears in both conjunctive mappings, in the second mapping, it has more restrictions on its attributes: x must manage itself. Thus, Mgr1(x, x) can no longer represents the idea of Manager but Self-Management instead. Therefore, Example 7 has two *Concepts*: Manager and Self-Management rather than only one. We call the attributes in B.Mgr1(x, x) in (7.2a) a *self-restrictive* attribute, which is defined to be an attribute that is in a subgoal that appears in two conjunctive mappings with the same IDB name but has more restrictions than the same attribute in the other mapping. Correspondingly, component (7.2a) is a self-restrictive component (Definition 6). □

Example 7 shows that it is hard to judge whether two mappings describe the same idea. This thesis provides the first pass at understanding a *Concept* which is some common topic or idea that can help make the mediated schema more comprehensible. But what is a *Concept* in the conjunctive mappings?

We begin with the following definition of a *Concept* in conjunctive mappings, which allows a careful analysis of the cases that are clear-cut. There are additional issues to consider, but in this thesis we focus on these cases that are clear and essential for deciding a *Concept*.

A *Concept* is defined by a set of *aspects*. As in relations in relational databases, the *aspects* of a *Concept* are referred to as *attributes*. For example, the concept of flight includes the attributes flightNo, date, time, departure, destination, crew, etc, which are the *aspects* that are used to describe the *Concept* flight. In a conjunctive mapping, the name of each aspect is the corresponding variable name in the mapping, which represents one aspect of the *Concept* that the mapping describes.

Definition 1 (Concept in Conjunctive Mappings): A *Concept* in conjunctive mappings is defined as an idea, notion or entity that is common in all schemas that the conjunctive mappings are relating. Formally, a *Concept* C is expressed by a set of *aspects* $\mathcal{S}$. □

For ease of discussion, we provide the following terms and definitions.

Definition 2 (Component): Each conjunctive query Q in the conjunctive mapping is a component [22]. □

Definition 3 (Same Concept): We say that two conjunctive mappings CM_1 and CM_2 define the same concept if the overlapped subgoals in CM_1 and CM_2 are equivalent subgoal sets, which can be checked as follows:

1. Assume $\forall$ component $c \in CM_1 \cup CM_2$, $\nexists\ c'$ s.t. (1) $c \equiv c'$ (i.e., $c \subseteq c'$ & $c' \subseteq c$) and (2) $|subgoals(c)| > |subgoal(c')|$;

   Assume the overlapped schema of CM_1 and CM_2 is X;

   Let C_1 be a component from CM_1 over X;

   Let C_2 be a component from CM_2 over X.

2. Let $name(sg)$ be the relation name of the subgoal sg;

   Let $sg\_names(Q)$ be the names of all of the relations in query Q;

Let $overlap\_names = \{name(sg) \mid name(sg) \in sg\_names(C_1)$ and $name(sg) \in sg\_names(C_2)\}$;

Let $C_{1overlap} = \{sg \mid name(sg) \in overlap\_names$ and $sg \in body(C_1)\}$;

$C_{2overlap} = \{sg \mid name(sg) \in overlap\_names$ and $sg \in body(C_2)\}$.

3. We now create new queries Q_1 and Q_2 that describe the overlapping parts of C_1 and C_2 respectively. The goal is to see if the overlapping parts are equivalent:

   Let $IDB(Q_1)$ be $IDB(CM_1)$, and let $IDB(Q_2)$ be $IDB(CM_2)$;

   Let $subgoals(Q_1) = C_{1overlap}$, and $subgoals(Q_2) = C_{2overlap}$;

   Let all variables in Q_1 and Q_2 be distinguished. That is let $vars(head(Q_1))$ $= vars(subgoals(Q_1))$ and let $vars(head(Q_2)) = vars(subgoals(Q_2))$.

Then the following conditions should hold:

1. $IDB(CM_1) = IDB(CM_2)$ (which, by the above requirement, is also equal to $IDB(Q_1)$ and $IDB(Q_2)$);
2. $overlap\_names \neq \varnothing$;
3. Q_1 contains Q_2, and Q_2 contains Q_1 (i.e., they are equivalent).

□

Additionally, condition (1) is actually the assumption we have made at the beginning. This condition can be removed for further study in the future.

Condition (2) says that if schema X exists in two conjunctive mappings of the same *Concept*, at least one relation r should appear in both conjunctive mappings. This ensures that the *Concept* is compatible in the representation of schema X.

Condition (3) ensures that if one subgoal name appears in different conjunctive mappings for the same concept, these two subgoals should be exactly the same after a substitution of variable names.

Example 8 There are other cases when two conjunctive mappings follow the definition of same concept but still are not representing the same *Concept*. Consider the following conjunctive mappings.

Conjunctive Mapping 1: between A and B

```
student(sid, name) :- A.stu(sid, name)                          (8.1a)
student(sid, name) :- B.stu(sid, name, program)                 (8.1b)
```

Conjunctive Mapping 2: between A and C

```
student(sid, name, advisor) :- A.stu(sid, name),
                               A.advisor(sid, advisor)          (8.2a)
student(sid, name, advisor) :- C.student(sid, name, advisor),
                               C.student(sid1, advisor, name)   (8.2b)
```

Conjunctive mapping 1 and conjunctive mapping 2 together do not violate property 1 and 2, but if only consider conjunctive mapping 2, it is easy to find out that C.student(sid, name, advisor), C.student(sid1, advisor, name) is not talking about the *Concept* "student". This component is actually expressing the *Concept* of a student that has an advisor and the student is also his advisor's advisor. However, in this thesis, we only deal with whether two conjunctive mappings can describe the same *Concept*, rather than deciding whether the given conjunctive mapping itself is correct. □

Chapter 5

Creating a Mediated Schema in PDMS

In our scenario, a mediated schema of all peer schemas and the mappings from the mediated schema to local sources are necessary for query propagation and translation. We presented the preliminary knowledge of schema mediation in Section 2.1 and Section 2.2. We also discussed *Concept* which can be represented as the semantic information in mappings in Chapter 4. In this chapter, we present the definition of schema mediation in P2P settings, and present our approach of creating a mediated schema for a PDMS.

5.1 Pottinger's Schema Mediation Algorithm for DIS

Pottinger [22] provides a novel approach for the data integration system to automatically create a mediated schema M based on two local schemas E and F, and a mapping Map_{E_F} between them. Using her algorithm, not only a mediated schema M but the mappings from M to both local sources E and F, Map_{M_E} and Map_{M_F}, will also be generated [22]. In this section we describe Pottinger's algorithms of creating a mediated schema and mappings to local sources in a data integration system. The algorithm of translation queries is also discussed in this section.

Figure 5.1 shows Pottinger's mediated schema creation algorithm. There are mainly two categories of relations that will be generated in the mediated schema M: One is directly from local relations. They are not sharing a *Concept* with other schemas so that those relations do not appear in any of the mappings. The other is created from mappings. If there is a mapping between E and F which expresses something that E and F bear in common,

```
Procedure CreateMediatedSchema(E, F, Map_E_F)
/* E,F are schemas, and Map_E_F is a conjunctive mapping between them. */
    C = ∅
    ξ = ∅
    Let R = { r ∈ E ∪ F | r ∉ a projection-only component }
    For each relation r ∈ R
      Let m be a new relation
      name(m) = name(r)
      attributes(m) = attributes(r)
      Add ξ(r, m) to ξ
      Add m to M
    For each IDB name q ∈ IDB(Map_E_F)
      Let m be a new relation
      Let vars_q be the duplicate-free union of variables
        of queries that define q in Map_E_F
      name(m) = name(q)
      attributes(m) = vars_q
      Add ξ(q, m) to ξ
      Add m to M
    Return M and ξ
```

Figure 5.1: The CreateMediatedSchema Algorithm from [22]

there will be a relation in the mediated schema representing this common idea across the schemas.

A simple subset of GLAV mappings [12] is used to express the mapping from M to $E \cup F$. The heads of queries in $Map_{M\_E\cup F}$ can be treated as an intermediate schema, IS, which is used to indicate how each mediated schema relation relates to each particular source. Each component (defined in Section 2.1.3) c from Map_{E_F} creates two views in $Map_{M\_E\cup F}$. The first view, lv_c, is called a local view for c. It is a conjunctive query from M to IS. LV_M consists of all such local view definitions for M. The second view, gv_c, is called a global view for c. gv_c is a query from E to IS, and is included in GV_M, the global view definitions for M. Figure 5.2 shows the algorithm of creating the mapping $Map_{E\cup F}$.

Given M and $Map_{M\_E\cup F}$, any query that is over M can be reformulated to local sources simply following LV_M and GV_M. The query reformulation algorithm is shown in Figure 5.3.

Procedure **CreateMapping**(E, F, Map_{E_F}, M)
/* E and F are schemas, Map_{E_F} is a conjunctive mapping between them, and M and ξ are the outputs from CreateMediatedSchema(E, F, Map_{E_F}) */
$LV_M = \emptyset$
$GV_M = \emptyset$
For each relation m $\in$ M
If e $\in$ E$\cup$ F and ξ(e, m)
Let q be a fresh IDB name
Let lv_m = q(attributes(m)) :- m(attributes(m))
Let gv_m = q(attributes(m)) :- e(attributes(e))
Add lv_m to LV_M
Add gv_m to GV_M
For each component c $\in$ Map_{E_F}
Let cname = IDB(c)
Let m be the relation in M s.t. ξ(cname, m)
Let q be a fresh IDB name
lv_c = q(vars(c)) :- m(attributes(m))
gv_c = q(vars(c)) :- body(c)
Add lv_m to LV_M
Add gv_m to GV_M
Return LV_M and GV_M

Figure 5.2: The CreateMapping Algorithm from [22]

Formal definitions of a mediated schema in Data Integration System and mapping from mediated schema to local sources, as well as the mediated schema criteria are provided in [22].

5.2 Problem Definition

Just as we have discussed in Chapter 1, a mediated schema which can be used to improve the comprehensibility of query translation will be very desirable in a PDMS. To make this idea more clear, look at Example 9.

Example 9 Assume that UW, UBC and UT are the three peers with databases storing the following information about conference papers shown in Figure 5.4:

UW.conf-paper(title, author, conference, venue, pages)
UBC.conf-paper(title, conference, year)

```
Procedure RewriteQuery(M, Map_{M_E∪F}, Q)
    Q' = maximally-contained rewriting for Q using LV_M
    /* Q' is over intermediate schema IS */
    Q" = expansion of Q' using GV_M
    /* Q" is over E ∪ F */
    Return Q"
```

Figure 5.3: Query Rewriting Algorithm from [22]

UT.conf-paper(title, author, abstract, area)

The mappings between UBC and UW, UW and UT are given as follows:

Mapping Map_{UBC_UW}:

```
conf-paper(title,conference):-
        UBC.conf-paper(title,conference,year)
conf-paper(title,conference):-
        UW.conf-paper(title,author,conference,venue,pages)
```

Mapping Map_{UW_UT}:

```
conf-paper(title,author):-
        UW.conf-paper(title,author,conference,venue,pages)
conf-paper(title,author):-
        UT.conf-paper(title,author,abstract,area)
```

Assume that a mediated schema M is created for all three peers using the above information:

M.conf-paper(title,conference,year,author,venue,pages,abstract,area)

We further assume that the Mappings from M to each local peer can be obtained using GLAV mappings:

Mapping Map_{M_UBC}:

LAV:

```
conf-paper(title,conference,year):-
    M.conf-paper(title,conference,year,author,venue,pages,abstract,area)
```

GAV:

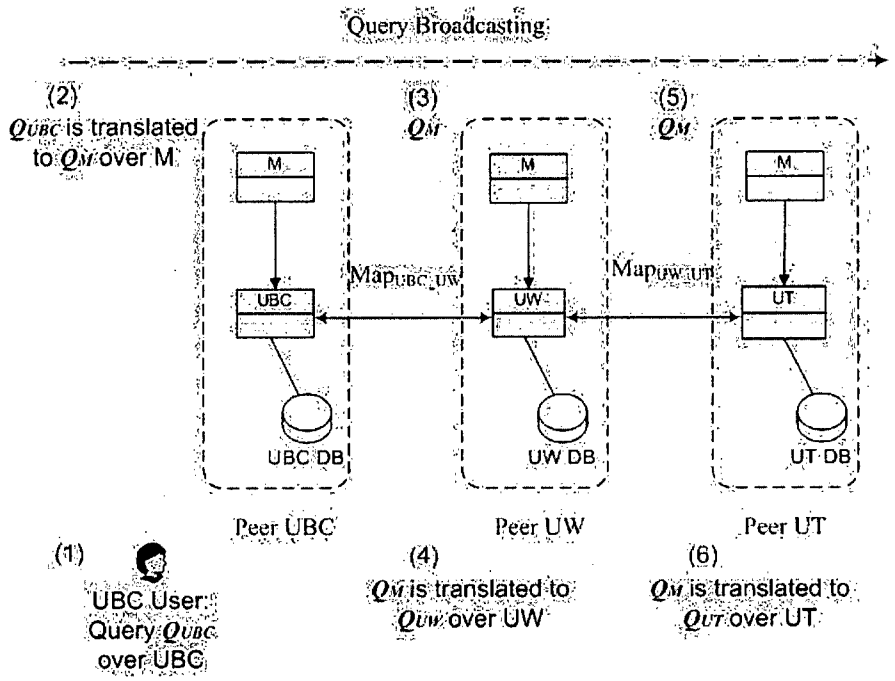


Figure 5.4: Query Processing in MePSys

conf-paper(title,conference,year):-
UBC.conf-paper(title,conference,year)

Mapping Map_{M_UW}:
LAV:
conf-paper(title,author,conference,venue,pages):-
M.conf-paper(title,conference,year,author,venue,pages,abstract,area)
GAV:
conf-paper(title,author,conference,venue,pages):-
UW.conf-paper(title,author,conference,venue,pages)

Mapping Map_{M_UT}:
LAV:
conf-paper(title,author,abstract,area):-
M.conf-paper(title,conference,year,author,venue,pages,abstract,area)

GAV:

conf-paper(title,author,abstract,area):-
UT.conf-paper(title,author,abstract,area)

Now, using these GLAV mappings, queries can be easily translated from over M to any local schema or from any local schema to M.

A user at UBC can ask a query in UBC's own schema:

q(title):-UBC.conf-paper(title,conference,year)

This query is first translated to that over M using Map_{M_UBC}:

q(title):-M.conf-paper(title,conference,year,author,venue,pages,abstract,
area)

Then peer UBC broadcasts this query to all of this acquaintances, and acquaintances to acquaintances. For example, UW receives this query. Using the Mapping Map_{M_UW}, this query can be translated to that over UW:

q(title):-UW.conf-paper(title,author,conference,venue,pages)

In the same way, answers can be reformulated and passed back to UBC. Thus, UBC can get not only answers from its own peer database but also from other peer database.

Users at UBC can also pose their query directly over the mediated schema, by using which users are able to query more information in the network. □

Since Pottinger has already provided the related algorithm in [22] (described in Section 5.1), are these algorithms already enough to create a mediated schema in a PDMS setting? Unfortunately, the answer is no. In a PDMS, any peer can join and leaves the network at any time, so the schema mediation algorithm needs to be commutative and associative and satisfy all the requirement of a P2P system. Though [22] considers to mediate more than two schemas, when the order of local schemas changes, the final mediated schema might also change. Furthermore, [22] requires that to include a new local schema E into the previously created mediated schema M, a

mapping from M to E should be specified. This could be achieved in a data integration system. However, in a P2P system, nobody can be expected to take the responsibility for this task. Also, in a PDMS, it is very likely that two different versions of mediated schema (containing information of different sets of peer schemas) might meet at some point, a merging operation to merge the two mediated schemas would be inevitable. Such an instance is never expected in a data integration system, nor in [22]. Thus, these new tasks will be tackled in MePSys.

The algorithms of how to create a mediated schema and mappings to local sources in a PDMS setting as shown in Example 9 will be described in the following sections of this chapter.

To create a mediated schema, we extend the algorithms in Section 5.1. This includes (1) using *Concepts* to create a more comprehensible mediated schema, (2) ensuring that the algorithm is commutative and associative (since any peer can enter or leave the network at any time). Here we provide the formal definitions of schema mediation in P2P environment. We begin by informally defining Concept Mediation, and then formally define it in Definition 4. We use the definition of Concept Mediation to formalize our definition of Mediated Schema In P2P in Definition 5.

Informally, concept mediation can be explained as the following: Assume that a set of peer schemas $S = \{P_1, P_2, \ldots, P_k\}$ and a set of mappings M between some pairs of the schemas are given. Each mapping may contain several conjunctive mappings. Each conjunctive mapping only specifies one aspect of the commonality between the pairs of peers. We assume that all conjunctive mappings with the same IDB name in the set M are talking about the same thing, i.e., more formally they refer to the same concept (Definition 3). We put all the information provided by these conjunctive mappings (defined in Section 2.1.3) together and form a global relation for this concept.

Definition 4 (Concept Mediation): given a set of peer schemas $S = \{p_1, p_2, \ldots, p_k\}$ and a set of mappings M between some pairs of the schemas, a Concept Mediation is the process of creating a representation (i.e., relation

and mapping to sources) in the mediated schema that corresponds to those conjunctive mappings in the set M with the same IDB name. □

Informally, the mediated schema in P2P can be defined as the union of all concepts of its peers. Note that if every concept is calculated by using Concept Mediation, then the Mediated Schema is the union of all mediated concepts. A Mediated Schema should also follow the Information Capacity criteria: all queries that can be asked over the source schemas can be asked over the mediated schema and the same results are returned [22].

Definition 5 (Mediated Schema in P2P): given a set of peer schemas $S = \{P_1, P_2, \ldots, P_k\}$ and a set of mappings M between some pairs of the schemas, a Mediated Schema is the union of all resulting relations from Concept Mediation (commonalities between schemas) and those relations existing in S but not existing in any of the subgoals of the mappings in set M (specialities for local peer schema). □

Further, a mapping Map_{M_E} from the mediated schema M to each local source E, i.e. the GLAV mapping between M and UBC in Example 5, is necessary so that a query over M can be reformulated to that over E. In our algorithm, the mapping Map_{M_E} is in the form of GLAV, and each peer E maintains their own GLAV mapping Map_{M_E}.

To make the mediation process easier, we introduced a construct *MappingTable*. A MappingTable contains all local information about a specific *Concept*. With the use of MappingTable, the presence of indirect mappings that identify additional information about relationships betweens schemas can be fully used. Consider Example 10.

Example 10 : Consider the indirect mapping information in the following mappings:
Conjunctive Mapping A_B:

```
author(name) :- A.auth(name, affiliation, contactInfo)
author(name) :- B.author_paper(name, paper)
```

Conjunctive Mapping B_C:

author(name) :- B.author paper(name, paper)
author(name) :- C.author(name, affiliation, title)

Conjunctive Mapping A_C:

author(affiliation) :- A.auth(name, affiliation, contactInfo)
author(affiliation) :- C.author(auth-name, affiliation, title)

□

For mapping A_B and mapping B_C, it is clear that the first attributes in A.auth, B.author and C.author are the same since they are mapped to the same variable: name. In mapping A_C, the first attributes of A.auth and C.author are not mapped to the same variable, so by A_C only, it is impossible to tell that they represent the same information. However, by composing A_B and B_C, it is clear that in the first attributes of A.auth and C.author represent the same information. We call the information found in the composed mapping that is not included in the direct mapping *indirect mapping information.* One advantage of our work is that such indirect information can complement information in the original mappings. Previous work tends to focus on translating queries on a single source of information, either from the original mapping or from the composed mapping. Since users are likely to provide incorrect or incomplete mapping information, our system helps users check whether their mappings are correct, and the system will automatically combine all mapping information together, regardless of where the information comes from.

In Section 5.3, we introduce the use of a MappingTable to merge all mapping information, not just the direct information. In Section 5.4 we introduce the idea and the algorithm of a mapping compatible check to ensure that those relations in the mediated schema are concept-based. We present the schema mediation algorithm for P2P in Section 5.5.

5.3 MappingTable Creation

5.3.1 Intuitions

A MappingTable both helps create the mediated schema, and is used to create the mediated schema to source mappings. Each MappingTable represents a single concept, including both the direct and indirect mapping information in relating a concept. As shown in Example 11, each source relation is given a row, and each attribute is represented by a column. Each column represents one aspect of that concept.

Example 11 : The MappingTable created for the mapping in Example 6 is shown in Figure 5.5:

Relation: M1.conf-paper(title, venue, year, url, pages)

M1.conf-paper	1	2	3	4	5
UBC.conf-paper	1	2	3	4	
UW.conf-paper	1	2	3		4

Figure 5.5: MappingTable for Example 6

Each entry in the table refers to the location of the attribute in the source (e.g., the fourth attribute in UW.conf-paper gives information about the attribute pages in the mediated schema relation). □

As the mediated schema grows to encompass more peers and more mappings, sometimes a new MappingTable will be created to represent the same concept as one created for the previously-existing mediated schema. Let's assume that conf-paper is an existing concept represented by the MappingTable MTold. Let the new MappingTable representing conf-paper be MTnew. MTold and MTnew need to be merged to form one concept for conf-paper in the mediated schema. Example 12 gives the intuition of merging two MappingTables of the same concept.

Example 12 : Let MTold be the MappingTable in Example 11. Let MTnew be the MappingTable shown in Figure 5.6.

Relation: M2.conf-paper(title, venue, year, url, author)

M2.conf-paper	1	2	3	4	5
UBC.conf-paper	1	2	3	4	
UT.conf-paper	1	2	3		4

Figure 5.6: MappingTable for Example 12(a)

Since MTold and MTnew are both about "conf-paper", they can be merged to be one MappingTable, shown in Figure 5.7.

Relation: M3.conf-paper(title, venue, year, url, pages, author)

M3.conf-paper	1	2	3	4	5	6
M1.conf-paper	1	2	3	4	5	
UBC.conf-paper	1	2	3	4		
UW.conf-paper	1	2	3		4	
M2.conf-paper	1	2	3	4		5
UBC.conf-paper	1	2	3	4		
UT.conf-paper	1	2	3			4

Figure 5.7: MappingTable for Example 12(b)

Both "UBC.conf-paper" in line 3 and 6 are kept for clarity. However, in real implementation and after optimization, only one of them needs to be kept.

□

5.3.2 Algorithm

Figure 5.8 shows how to create a MappingTable for each relation that is constructed from a mapping Map_{E_F} in the mediated schema. Procedure createMediatedSchema(E, F, Map_{E_F}) in Figure 5.1 will first be called.

Given Map_{E_F} and M_{EF}, the MappingTables will be constructed for each concept in Map_{E_F}.

When a peer have two different versions of mediated schema M_1 and M_2 maintaining different information, if both M_1 and M_2 contain concept q, in order to merge r_1 and r_2, corresponding MappingTable $MT_1(q)$ and $MT_2(q)$ needs to be merged. (r_1 and r_2 are the corresponding relations in M_1 and M_2 representing the concept q.)

The MappingTable merging process follows the same general principles:

1. Related attributes should be positioned in the same column;
2. Un-related attributes are in different columns;
3. Overlapping local relations in the two MappingTables are used to determine each column in one MappingTable corresponds to that in the other MappingTable.

Procedure mergeMappingTable(mt_1, r_1, mt_2, r_2) in Figure 5.9 merges any two MappingTable of the same concept mt_1 and mt_2, and returns a merged MappingTable. It is required that there be overlapped relations in mt_1 and mt_2; otherwise, they cannot be merged. Since it is likely that each MappingTable contains indirect mapping information that the other one does not have, the first step for merging two MappingTable is to update such information for each MappingTable shown in Figure 5.10.

5.3.3 Section Summary

In this section, we presented by examples the motivation of introducing a MappingTable for each relation in the mediated schema that comes from a mapping. We also discussed the case of merging two MappingTables. We further gave three algorithms that are related to operating MappingTables. They are creating a MappingTable, merging MappingTables and updating current MappingTable using additional mapping information in another MappingTable.

```
Procedure CreateMappingTable(Map_{E_F}, M_{EF})
    /* M_{EF} is the mediated schema created from Map_{E_F} */
    MT = ∅
    For each relation m ∈ M constructed from IDB(Map_{E_F})
      Let mt be a new MappingTable with #attr(m)+1 columns
      /* The next three lines add m into mt */
      mt(0, 0) = name(m)
      For i = 1 to # attr(m)
        mt(0, i) = i
      Let q_E be component from Map_{E_F} over schema E
      Let q_F be component from Map_{E_F} over schema F
      Add(q_E, mt)
      /* call Procedure Add(q_E, mt) to add all subgoals of q_E to mt */
      Add(q_F, mt)
      /* call Procedure Add(q_F, mt) to add all subgoals of q_F to mt */
      Add mt to MT_{E_F}
    Return MT_{E_F}

Procedure Add(q, mt)
    /* Add all subgoals of q to MappingTable mt */
    Let m be the first relation in mt
    Let j be # next row to insert in mt
    For each subgoal g of q
      mt(j, 0) = name(g)
      For each attribute x of g
        Let x be the ith attr of g
        Find k s.t. var(g.attr(i)) = var(m.attr(k))
        mt(j, k) = i
        j = j + 1 //move to the next row
```

Figure 5.8: Create MappingTable Algorithm

5.4 Mapping Compatible Check

Two mappings that are created at different peers by different users might have the same IDB name, indicating that these two mappings represent the same concept. However, as is discussed in Section 4.2, two mappings having the same concept should follow Definition 3 (i.e. these two mappings should have the same IDB name, they have overlapped subgoal names and their overlapped subgoal sets are equivalent). The module of Mapping Compatible Check is to make sure that the mappings are really representing the same

```
Procedure MergeMappingTable(mt1, r1, mt2, r2)
    /* mt1 and mt2 are two MappingTables with the same concept;
       r1 and r2 are the corresponding relations in the mediated schema for
       mt1 and mt2 */
    /* First update each MappingTables using indirect mapping information
       specified in the other */
    UpdateMappingTable(mt1, mt2) //see Figure 5.10
    /* Second, merge the MappingTables*/
    Let S be the set of overlapped relation names in mt1 and mt2
    Let overlappedCol = # column(S)
    newMTCol = # column(mt1) + # column(mt2) + 1 - overlappedCol
    /* Construct a new MappingTable mtnew of newMTCol columns*/
    hash1 = ∅
    /* how columns in mt1 map to mtnew */
    For ∀ column i of mt1
      hash1(i) = new(k)
      /* k is a column in mtnew that has not been assigned */
    hash2 = ∅
    /* how columns in mt2 map to mtnew */
    For ∀ column i of mt2
      If ∃ row j s.t. mt2(j, 0) ∈ S and mt2(j, i)≠ blank
        Find k, l s.t. mt1(k, 0) = mt2(j, 0) and mt1(k, l) = mt2(j, i)
        hash2(i) = hash1(l)
      Else
        hash2(i) = new(k)
    /* Add the first row to mtnew */
    mtnew(0, 0) = name(r1)
    For i = 1 to newMTCol
      mtnew(0, i) = i
    /* Add mt1 to mtnew */
    For each row i of mt1
      Let p be the next row in mtnew
      For each column j of mt1
        If mt1(i, j) is not blank
          mtnew(p, hash1(j)) = mt1(i, j)
    /* Add mt2 to mtnew */
    For each row i of mt2
      Let p be the next row in mtnew
      For each column j of mt2
        If mt2(i, j) is not blank
          mtnew(p, hash2(j)) = mt2(i, j)
    Return mtnew
```

Figure 5.9: Merge MappingTable Algorithm

Procedure **UpdateMappingTable**(mt_1, mt_2)
  ∀ relation name x_1 and x_2 from mt_1
  If x_1 and x_2 both have attributes in column j of mt_1, but in mt_2, the corresponding attributes of x_1 and x_2 are in different columns k and l
    Combine column k and l of mt_2
  ∀ relation name x_1 and x_2 from mt_2
    If x_1 and x_2 both have attributes in column j of mt_2 but in mt_1, the corresponding attributes of x_1 and x_2 are in different columns k and l
      Combine column k and l of mt_1.

Figure 5.10: Update MappingTable Algorithm

concept when merging the information contained by two mappings.

Since every peer only maintains its own mappings and does not know other peers' mapping, the mapping compatible check needs to rely on the MappingTable that can infer the original mappings. The algorithm in Figure 5.11 inputs the original MappingTable of concept q for the mediated schema, $MT_m(q)$, and a new mapping of the concept q, $Map_{E\_F}(q)$. The algorithm first checks whether there exists a case when two attributes of a certain subgoal or two attributes of different subgoals in $Map_{E\_F}(q)$ have the same value but have different value in a previous mapping, which is represented by a row or a set of rows in $MT_m(q)$. The algorithm further checks if there exists a case when in $Map_{E\_F}(q)$, two attributes in one subgoal or in two subgoals from one component have the same value, but in $MT_m(q)$, the corresponding attributes have different values. If either of the above case is true, the new mapping of concept q is not compatible with a previous existing mapping of concept q. The mapping needs to be modified by the user.

We assume that at least one subgoal should be in common if one schema is involved in several conjunctive mappings with the same IDB name.

5.5 Peer Schema Mediation

We have briefly described MePSys in Section 5.2. In this section, we describe our Peer Schema Mediation algorithm (short for PSM algorithm), which is composed of three small algorithms, schema mediation, computing

```
Procedure MappingCompatibleCheck(Map_{E_F}(q), MT_m(q))
    /* Map_{E_F}(q) is a new mapping of concept q;
       MT_m is the MappingTable of concept q for the mediated schema */
    /* check whether Self-Restrictive component exists */
    /* Return True when compatible; False when incompatible */
    Divide all rows in MT_m(q) into several sets s.t. all consecutive rows from
      one peer are in one set
    For each set S of MT_m(q)
      If ∃ values v1, v2 in the same column from rows i, j which belong to S
        Find subgoal sg1 in Map_{E_F}(q) s.t. name(sg1) = MT_m(q)(i, 0)
        Find subgoal sg2 in Map_{E_F}(q) s.t. name(sg2) = MT_m(q)(j, 0)
        If sg1.attr(v1) ≠ sg2.attr(v2)
          Return False
    For ∀ subgoal sg1, sg2 from the same component of Map_{E_F}(q)
      If ∃ x and y s.t. sg1.attr(x) = sg2.attr(y)
        Find row i, j in MT_m(q) s.t. MT_m(q)(i, 0) = name(sg1),
          MT_m(q)(j, 0) = name(sg2)
        If x in row i and y in row j are not in the same column
          Return False
    Return True
```

Figure 5.11: Mapping Compatible Check Algorithm

GLAV mapping and query reformulation. In Chapter 6, there will be more discussion on how to handle the case when new peers join the network after database application has been built upon the earlier mediated schema, and how to handle the case when some peer leaves the network, or some local database schema evolves as time goes by.

5.5.1 System Setup Phase - Schema Mediation

System Work

For ease of discussion, we assume there is a setup phase for MePSys: several peers have joined the network, found their acquaintances and created mappings to the acquaintances but no mediation has been started. The system setup phase starts from a first peer starting mediation to a time point when all peers in the system get the most updated mediated schema of the whole network.

We assume that, at any time, each peer P maintains all of the following

information:

1. P's local database schema
2. A list L of mappings with P's acquaintances
3. A current version of mediated schema M
4. MappingTable set corresponds to M
5. GLAV mappings from M to P

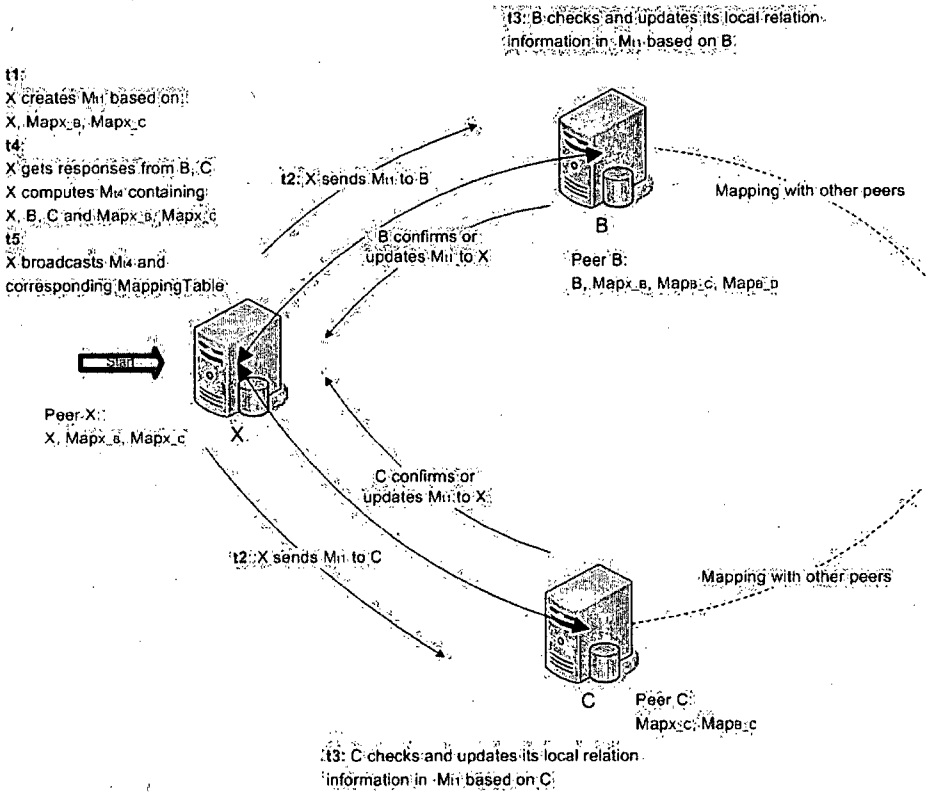


Figure 5.12: Schema Mediation Start-up

As shown in Figure 5.12, when mediation starts from peer X, X gets mappings from L one by one. At time t_1, X mediates information based on

each mapping map_{X_Y}. However, as is described in Section 5.1, there are two kinds of relations in the mediated schema: one is from local sources and another is from the mapping. Since in a PDMS, every peer does not know its acquaintance's schema, the mediated schema M_{t1} that X first creates is only composed of two parts: local relations from X and relations from the mappings. Information of Y's local relation is missing from M_{t1}. At time t_2, X sends M_{t1} to each acquaintance, Y and asks Y to confirm or update M_{t1}. At time t_3, each acquaintance confirms M_{t1} and updates M_{t1} based on its local schema information, i.e. adding those local schema information that X does not know. At t_4, X receives all confirmation from its acquaintances, X will compute a new mediated schema, say M_{t4}, which contains all schema information of X and all of its acquaintances as well as the mappings between X and its acquaintances. X further computes the GLAV mapping from M_{t4} to X. X also updates MappingTables and let the mediated schema to be M_{t4} in its list. At time t_5, X sends M_{t4} and corresponding MappingTables to all of its acquaintances.

When a peer E upon receiving a mediated schema M_{tn}, first checks whether it already has a mediated schema. If it has not, E will do the same as X does. Otherwise, merge the two mediated schemas. If the mediated schema that E maintains changed after this step, E sends out this new mediated schema $M_{t(n+1)}$ and corresponding MappingTables to all of its acquaintances; otherwise, no message will be sent out. E also computes the new GLAV mapping and maintains the updated information in his list L.

After a period when every peer E has received the most updated mediated schema $M_{t(n+k)}$ and computed the GLAV mapping from $M_{t(n+k)}$ to E, the system setup finishes. The algorithm of creating the mediated schema in the setup phase is shown in Figure 5.13.

Theorem 1 The time of creating a mediated schema using the above algorithm has an upper bound.

Proof Sketch: Assume there are n peers in the network. Let t be the time for one hop in the network. Two kinds of messages need to be considered: sending out the partial mediated schema and receiving confirmation of local

Algorithm **SystemSetUp**
/* to be processed at each peer E when receiving a *mediation* message from other peer */
Input: 1. mediation message containing a partially created mediated schema M_{tmp} of previous peers, a MappingTable MT_{tmp} corresponding to M_{tmp}
2. local schema mapping list
Output: A new mediated schema and MappingTable which will be sent out to all of E's acquaintances

If previously there is no mediated schema at peer E
    Mediate schemas M_{E_tmp} based on E and all its mappings to acquaintances
    Send out check message with M_{E_tmp} and wait for confirmation from the acquaintances
    Received updated mediated schema M_{EF} from each acquaintance F with F's local schema information in it
    Compute a local mediated schema M_E for E and all its acquaintances based on all M_{EF} ($F \in E$'s acquaintance list)
    Merge M_E and M_{tmp} to be M'_E
Else (there exists previous version of mediated schema M_E at E)
    Merge M_E and M_{tmp} to be M'_E
If M'_E does not equal to M_E
    Send M'_E and its corresponding MappingTable to all the acquaintances
Else
    No message to be sent

Figure 5.13: System Setup Algorithm

relations, and sending out the mediated schema. The upper bound for the first part is $2nt$. So we only consider the second part. Each peer maintains the mapping information with its acquaintances. We can model this problem to the following graph problem: when does each node get all the edge information? Since each node knows its edge information, this problem can further be transferred to the problem: When does each node get all the node information? A node E knows the information of node F, either from F directly, or from F's neighbor. The maximum time difference between these two is one hop t. We drop this one hop difference temporarily. So in order to let E get a full set of node information, E should receive a message from all other nodes. E receives the earliest message from F after (shortestPath(A, F) + shortestPath(F, E)) hops. So E receives all nodes information at hop

hop_E:

$$hop_E = \max_F\{shortestPath(A, F) + shortestPath(F, E)\}$$

Thus the number of hops for all nodes receiving all other node information is hop_{all}:

$$hop_{all} = \max_E hop_E = \max_E\{\max_F\{shortestPath(A, F)+shortestPath(F, E)\}\}$$

Considering the messages to confirm local relations, the total time before every peer gets the ultimate mediated schema is T_{all}:

$$T_{all} = (hop_{all} + 2n) * t$$

Considering the fact that information about F can be obtained from F's neighbor, T_{all} can sometimes be

$$T_{all} = (hop_{all} + 2n - 2) * t$$

This is the upper bound for the whole mediation process. □

Merging Two Mediated Schemas

In a PDMS, different versions of mediated schemas are broadcasted in the network before a final mediated schema for all peers in the network has been built. At any time in the setup phase, a peer E that already maintains a mediated schema M_1 can receive a different mediated schema M_2 that contains non-overlapped information with M_1. In this case, M_1 and M_2 need to be merged before E sends any of them to other acquainted peers.

Our mediation process follows the following strategy:

1. When E gets a mapping E_F, a mediated schema M_{EF} will be created first. This can use algorithm in Figure 5.1.

2. Then consider previous existing mediated schema M_E, merge M_{EF} and M_E to get M'_E, and update M_E to be M'_E.

3. When E originally has a mediated schema M and receives a new mediated schema M_{tmp}, M and M_{tmp} will be merged.

Figure 5.14 shows the algorithm of how to merge two mediated schema.

```
Algorithm MergingTwoMediatedSchema(M1, M2, MT1, MT2)
/* M1, M2 are the two mediated schemas, MT1, MT2 are the corresponding
   MappingTables */
   Mnew = ∅
   For ∀ mt1 ∈ MT1
     If ∃ mt2 ∈ MT2 s.t. name(mt1(0, 0)) = name(mt2(0, 0)) = q
     /* both mt1 and mt2 are about concept q */
        Find the corresponding relation r1 and r2 from M1 and M2
        mtnew =MergeMappingTable(mt1, r1, mt2, r2)
        Create a new relation rnew
        name(rnew) = q
        Let n be column size of mtnew
        Let the first row number of mt1 in mtnew be index1
        Let the first row number of mt2 in mtnew be index2
        For i = 0 to n
          If mtnew(index1, i) contains value x
            rnew.attr(i) = r1.attr(x)
          Else (mtnew(index2, i) contains value x)
            rnew.attr(i) = r2.attr(x)
        Add rnew to Mnew
   For ∀ r1 ∈ M1
     If r1 hasn't been merged or added to Mnew
        Add r1 to Mnew
   For ∀ r2 ∈ M2
     If r2 hasn't been merged or added to Mnew
        Add r2 to Mnew
   Return Mnew
```

Figure 5.14: Merge Two Mediated Schema Algorithm

Compute GLAV Mapping for Each Local Peer

Next, having created the mediated schema, we need to be able to create the GLAV mapping from the mediated schema to the local peer schema in order to allow for queries to be translated. With the mediated schema and the corresponding MappingTables, computing the GLAV mapping is easy. Figure 5.15 shows an algorithm of computing the GLAV mapping for local

schema E. Example 13 gives an example of computing GLAV Mapping for local peer.

```
Algorithm ComputeGLAVMapping(M, MT, E)
/* M is the mediated schema, MT is the corresponding MappingTable for M,
   E is the local schema */
   LV_M = ∅
   GV_M = ∅
   For each m ∈ M
      If m does not have a corresponding mt ∈ MT
         If ∃ e ∈ E s.t. name(e) = name(m)
            Let lv_m = q(attributes(e)) :- m(attributes(m))
            Let gv_m = q(attributes(e)) :- e(attributes(e))
            Add lv_m to LV_M
            Add gv_m to GV_M
      Else //m has a corresponding mt ∈ MT
         Let body be the subgoal set for local peer, body = ∅
         For each row i of mt
            If ∃ r ∈ E s.t. name(r) = mt(i, 0)
               Construct subgoal sg s.t. name(sg) = name(r), using m's
                  attribute value as the corresponding attribute value for sg
               Add sg to body
               If ∄ s ∈ E s.t. name(s) = mt(i + 1, 0)
                  Let lv_m = q(var(body)) :- m(attributes(m))
                  Let gv_m = q(var(body)) :- body
                  Add lv_m to LV_M
                  Add gv_m to GV_M
                  body = ∅
   Return LV_M, GV_M
```

Figure 5.15: Compute GLAV Mapping Algorithm

Example 13 Given schema B, mediated schema M and MappingTable MT, the GLAV mapping from M to B can be computed.

Schema B:

B_flight(date, company, flightNo, service)

B_schedule(date, flightNo, depart, arrival, numLeft)

...

Schema M:

flight(date, flightNo, company, service, class, ID,
discount, depart, arrival, numLeft)

...

MappingTable: flight

M.flight	1	2	3	4	5	6	7	8	9	10
B_flight	1	3	2	4						
B_schedule	1	2						3	4	5
C_schedule	2	3				1				
C_price					2	1	3			

The GLAV mapping for Map_{M_B} can be obtained.

Mapping Map_{M_B}:

LAV:

Q1(date, company, flightNo, service):-
M.flight(date, flightNo, company, service, class,
ID, discount, depart, arrival, numLeft)

GAV:

Q1(date, company, flightNo, service):-
B.B_flight(date, company, flightNo, service),
B_schedule(date, flightNo, depart, arrival, numLeft)

□

Query Reformulation Algorithm

With the GLAV mapping computed for each local peer E, any query posed over the mediated schema M can be easily reformulated to that over E and vice versa. This enables a fast translation for queries between any local schema and M, which can be used for query processing in the PDMS as discussed in Section 5.2. Figure 5.16 gives the algorithm of query reformulation. Example 14 shows an example of query reformulation using this algorithm.

Example 14 Reformulate Query Q to that over local schema B using the GLAV mapping computed in Example 13

```
Algorithm QueryReformulation(q, Map_{M_E}), M, E
/* q is the query posed by the user; Map_{M_E} is the GLAV mapping between M
  and E; M is the mediated schema; E is the local schema */
    Let reformQ be the reformulated query of q
    Let head(reformQ) = head(q)
    If any subgoal sg of q ∈ M
      For each subgoal sg of q
        Find in LAV a view lv s.t. sg ∈ body(lv)
        For each variable v in lv
          Let pos be the position of v in body(lv)
          Change v to the variable v' in sg s.t. pos is the position of v' in sg
        Find in GAV a view gv s.t. head(gv) = head(lv)
        For each variable v in head(gv)
          Change v to v' with the same position in the head of lv
        Append body(gv) to the body of reformQ
    Else // sg ∈ E
      For each subgoal sg of q
        Find in GAV a view gv s.t. sg ∈ body(gv)
        For each variable v in gv
          Let pos be the position of v in body(gv)
          Change v to the variable v' in sg s.t. pos is the position of v' in sg
        Find in LAV a view lv s.t. head(lv) = head(gv)
        For each variable v in head(lv)
          Change v to v' with the same position in the head of gv
        Append body(lv) to the body of reformQ
    Return reformQ
```

Figure 5.16: Query Reformulation Algorithm

Q = q(d, f) :- M.flight(d, f, price, company, service, class, ID, discount, depart, arrival, numLeft)

Using the algorithm in Figure 5.16, Q is reformulated to Q' over B

Q' = q(d, f) :- B.B_flight(d, company, f, service), B.B_schedule(d, f, depart, arrival, numLeft)

□

5.5.2 Section Summary

In this section, we discussed the system setup phase - creating a mediated schema for the PDMS from the beginning. We presented PSM algorithm which includes schema mediation, computing GLAV mapping and query reformulation algorithms. For each step, we give examples to illustrate our idea. MePSys provides the first prototype to automatically create a mediated schema in PDMSs. With PSM algorithm, queries can be easily processed among different peers in a PDMS.

Chapter 6

Updating the Mediated Schema

Some of the peers might build up a database application using the mediated schema after the system set-up phase. For those peers having applications over the mediated schema, if the mediated schema changes, the already-built applications might need to be re-built, which takes quite a lot of redundant work. On the other hand, if a peer leaves, that peer schema information needs to be dropped from the mediated schema. However, no existing algorithm has been proposed to deal with such a case. In this Chapter, we discuss how to update the mediated schema in the steady state - i.e., after the system setup phase. In particular, we determine both how to update the mediated schema and mappings if a new peer joins the PDMS network (Section 6.1) or an old one leaves the network (Section 6.2), and how the mediated schema and associated mappings change if some peer's local database schema evolves (Section 6.3).

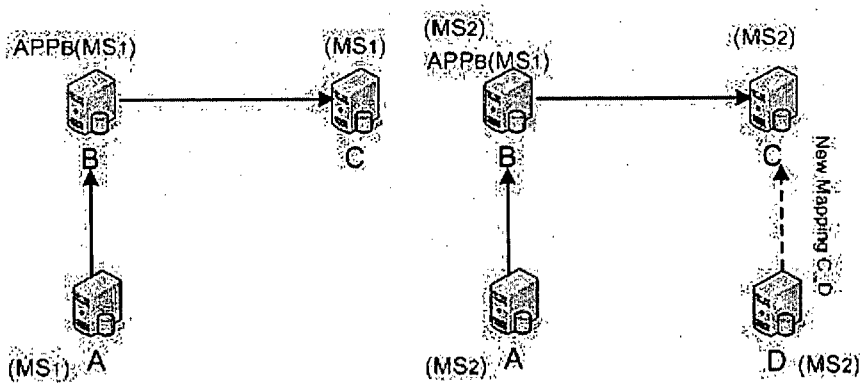
6.1 Adding a New Peer to the System

After the system setup phase, every peer maintains an up-to-date mediated schema M. If a new peer P decides to enter the network, the mediated schema M needs to be updated to a new mediated schema M' which also include P's schema information. However, since after the system setup phase, some peer has already built up an application over M. A naive approach is to compute a new mediated schema M', and rebuild all those already-built applications, which takes quite a lot of redundant work. Since the mediated schema can change periodically during the life cycle of such a PDMS, it is not realistic to ask the user of each peer to build a new

database application every time the mediated schema is updated. A good strategy is to let the peer maintain its already-built application $APP_P(M)$ and maintain a mapping from M' to M. Since the mapping from M' to P can easily be calculated by using the MappingTable that corresponds to M', any query over M passed from other peer can be reformulated to that over P. P's user can also use its local schema or the mediated schema M on which APP_P is built. Below is an concrete example of such a scenario.

Example 15 An example of a new peer joining the network
There are three peers in the current system: A, B and C. Mappings are created between A and B, and B and C as Map_{A_B} and Map_{B_C} respectively. We only consider relations that can represent a *Concept* (in the form of a mapping) for the mediated schema because they are the core part of our discussion.

Consider the situation shown in Figure 6.1(a): after the setup phase, a mediated schema MS_1 has been built and maintained at each peer. The mappings from MS_1 to A, B and C and the MappingTables also have been built.



(a) initial system (b) after peer D joins

Figure 6.1: Adding a New Peer to the System

After building the mediated schema MS_1 and the mappings to local sources, user at peer B built a database application APP_B using the medi-

ated schema MS_1. So the information stored at each peer can be summarized as follows.

- Peer A: schema A, Map_{A_B}, MS_1, $Map_{MS_1\_A}$, MT_1.
- Peer B: schema B, Map_{A_B}, Map_{B_C}, MS_1, $Map_{MS_1\_B}$, MT_1, $APP_B(MS_1)$.
- Peer C: schema C, Map_{B_C}, MS_1, $Map_{MS_1\_C}$, MT_1.

Sometime later, a new peer D decides to join in the network. It creates a mapping Map_{C_D} to peer C, as shown in Figure 6.1(b).

Using the algorithms described in Chapter 5, a new mediated schema MS_2, MappingTable MT_2 and GLAV mappings can be computed at peer C.

C passed MS_2, MT_2 to D. D computes $Map_{MS_2\_D}$. C further computes the GLAV mapping from MS_2 to C using $Map_{MS_2\_MS_1}$ and $Map_{MS_1\_C}$ and get the new GLAV mapping $Map_{MS_2\_C}$ for itself.

C also passes MS_2, MT_2, $Map_{MS_2\_MS_1}$ to its acquaintance B. B computes Map_{MS2_B} using $Map_{MS_2\_MS_1}$ and Map_{MS1_B}. B also passes MS_2, MT_2, $Map_{MS_2\_MS_1}$ to its acquaintance A. Using $Map_{MS_2\_MS_1}$ and $Map_{MS_1\_A}$, a mapping from mediated schema MS_2 to local schema A can be computed.

At the end, all the peers get the most updated information about MS_2. Each peer also knows how to map its local schema to MS_2. Information stored at each peer can be summarized as follows.

- Peer A: schema A, Map_{A_B}, MS_2, $Map_{MS_2\_A}$, MT_2.
- Peer B: schema B, Map_{A_B}, Map_{B_C}, MS_1, $Map_{MS_1\_B}$, $APP_B(MS_1)$, $Map_{MS_2\_MS_1}$, $Map_{MS_2\_B}$, MS_2, MT_2.
- Peer C: schema C, Map_{B_C}, Map_{C_D}, MS_2, $Map_{MS_2\_C}$, MT_2.
- Peer D: schema D, Map_{C_D}, MS_2, $Map_{MS_2\_D}$, MT_2.

For peer B, MS_1 and $Map_{MS_1\_B}$ are kept in its database application $APP_B(MS_1)$. $Map_{MS_2\_MS_1}$ is kept because query over the new mediated

schema MS_2 can be translated to MS_1 and further be used by application $APP_B(MS_1)$.

Thus, every peer can keep the updated mediated schema and maintain a mapping from the mediated schema to local schema. Compared to the naive approach, no burden has been added to the local peer users. Additionally, all the previously created database applications over previous versions of mediated schema can still be used. New applications can also be created over the new versions of the mediated schema.

Details of this example is shown in the Appendix A. □

6.1.1 Algorithm

Figure 6.2 shows the algorithm of updating the mediated information for the case of adding a peer. In this algorithm, we consider how to get a new mediated schema, and how to maintain the mapping from the new mediated schema to the old one on which an application might have been built.

Algorithm **Update Mediated Information (Adding a Peer E)**
Let M be the original mediated schema
Let MT be the corresponding MappingTable set
1. E creates a mapping to F which is already in the network
2. E creates a mediated schema M_{EF} using the Procedure in Figure 5.1
3. F creates a new mediated schema M' using M_{EF} and M using the Procedure in Figure 5.14
   F computes GLAV mapping $Map_{M'\_M}$ and $Map_{M'\_F}$
   F updates its maintained information: $M \rightarrow M'$, $MT \rightarrow MT'$ and $Map_{M\_F} \rightarrow Map_{M\_F}$
   F send M', corresponding MT' to E
   F broadcast M', corresponding MT' and $Map_{M'\_M}$ to all other acquaintances
   F computes $Map_{M'\_F}$ for itself
4. E maintains MT', M', and computes $Map_{M'\_E}$ using MT'
5. For any other peer G that received the message originally from F
   If there are applications built over M
   G maintains the following information: M', MT', $Map_{M'\_M}$, M, $Map_{M'\_M}$, $Map_{M'\_G}$
   Else G simply updates $M \rightarrow M'$, $MT \rightarrow MT'$ and $Map_{M\_G} \rightarrow Map_{M'\_G}$
   G further broadcasts all mediated schema information to its acquaintances

Figure 6.2: Update mediated information when new peer joins the network

6.2 Dropping a Peer from the System

In this section, we explore four possible approaches to update the mediated schema after dropping one peer schema. We compare the advantages and disadvantages of each strategy. Comparably, Strategy One and Two are the naive solutions. Strategy Three is better than One and Two. However, in the worst case, it faces the same problem as Strategy One and Two. Strategy Four, though poses additional requirement for the leaving peer, can keep the system much more stable and does not require more time. We think this is the best approach among all.

Strategy One: Once a peer decides to leave the network, the peer needs to notify any other node in the network, which triggers the schema mediation process from the very beginning. Every node in the network is regarded as a new peer.
Advantages: The setup for such a system is very easy. No additional algorithm for removal operation needs to be designed. Only schema mediation algorithm will be involved.
Disadvantage: Basically, this strategy is not realistic. First, the schema mediation process will be too frequent. If peers leave the network frequently in a short period, there will be too much system work assigned for schema mediation only. Resources can not be used wisely. Second, the previously-created mediated schema cannot be fully used of in the process of creating the new one. The new mediated schema might not change dramatically from the old one but this strategy requires that the new version of the mediated schema should be created from the beginning.

In conclusion, this strategy is not at all satisfactory.

Strategy Two: Re-do the schema mediation once every assigned period. If in one period, one peer is leaving the network, the system needs to do the schema mediation again from the beginning. There are two ways to know whether a peer X is leaving the network. (1) Peer X notifies any other node before its departure. (2) Other peer, usually X's acquaintance, PINs

or communicates with peer X. If it gets no response, X can be assumed to have left the network. X's information needs to be removed from the mediated schema and triggers a new schema mediation from the beginning.
Advantages: Better than Strategy one because the frequency of the schema mediation process is decreased. Algorithms involved are still simple.
Disadvantages: The previously-created mediated schema cannot be fully used of in the process of creating the new one.

Strategy Three: The leaving peer X will not notify other peers when it leaves the network. X's acquaintance Y will recognize X's leaving by PINing X or communicating with X but getting no response. Y once realizes X's absence, it will try to compute the new mediated schema without knowing the schema of X. However, this requires that peer Y be able to recognize which relation in the MappingTables comes from X.
Advantages: (1) Updating of the mediated schema is timely. (2) Peer can leave the network at any time without notifying any other. (3) The calculation is based on the previous mediated schema, so the process will not be time-consuming. Old versions of the mediated schema can be fully used of.
Disadvantages: (1) Every peer would be able to know other peers schema from the MappingTables, which is not ideal for the sake of safety and privacy reason. (2) For peers that lost connection because other peers' leave, such peers need to re-join the network, which costs additional schema mediation work after the deletion work. We use Example 16 to explain this idea.

Example 16 Consider the example proposed in Example 15 (details in Appendix A), and assume the status of the network is when MS_2 has been created and all mediated information has been stored at every peer, as is shown in Figure 6.1(b).

Case 1: B recognizes that A has left the network.

Once B recognizes that A has left the network, it checks A's information in the MappingTable MT_2. A is only involved in the mapping with B and has no other acquaintances, so the deletion will be easy:

1. remove from MappingTable set MT_2 all rows originally from schema A and remove from MT_2 all columns only from schema $A \rightarrow MT_3$
2. for every relation a in MS_2, if a is originally from schema A, remove a from MS_2 and update each relation a in MS_2 corresponding to MT_3;
3. delete Map_{A_B} from B's information set;
4. create the a mapping $Map_{MS_3\_MS_2}$;
5. broadcast MS_3, MT_3, and $Map_{MS_3\_MS_2}$ to the other peers.

Case 2: B Recognizes that C has left the network. This is a more general case when the leaving peer has more than one acquaintance and acts as a bridge connecting other peers.

B first checks MT_2. From the two intermediate mediated schemas IM_{BC} and IM_{CD}, B knows that C originally had connection with B and D. So B will first update the mediated schema and the MappingTable. Update the mediated information:

1. Delete from the mediated schema all relations that originally come from schema B. Following the details in Appendix A, delete "B_flight(date, company, flightNo, service)" and "B_price(flightNo, class, price)" from MS_2.
2. Delete from the MappingTable rows that represent relations from schema B and all intermediate relations which involves schema B. In this example, delete from MT_2 the rows "B_flight", "B_price", "IM_{BC}.flight", "IM_{CD}.flight" and "MS_1.flight".
3. Check all columns of the MappingTable whether there are columns not useful any more. Delete all such columns. Delete "ID" and "discount" from MT_2 in this example.
4. Update the relation in the mediated schema that comes from a mapping. In this example, "flight(date, flightNo, price, company, service, class, ID, discount, depart, arrival, numLeft)" is now updated to "flight(date, flightNo, price, company, service, class, depart, arrival, numLeft)" .

5. Since D is lost connection with any other node in the network, B can assume that D needs to re-join the network. B do the same deletion for D.

6. Get new mediated schema MS_3 and corresponding MappingTables MT_3. Create a mapping $Map_{MS_3\_MS_2}$.

7. Broadcast MS_3, MT_3 and $Map_{MS_3\_MS_2}$ to all the other peers.

All peers that lost connection need to re-enter the network. For the worst case, e.g. the topology shown in Figure 6.3, all nodes need to re-join the network if B leaves.

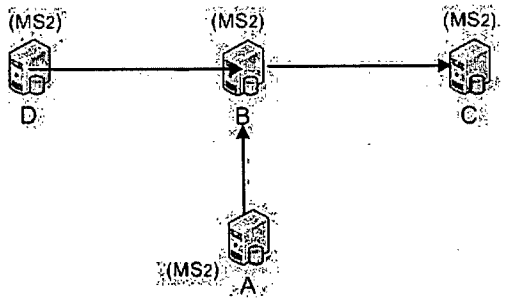


Figure 6.3: Peer Leaving (Bad Topology)

□

Strategy Four: A peer X cannot leave until X calculates the new mediated schema. The idea is that given the original mediated schema M and local schema X that is to be removed, compute the remaining part. In order to do this, a "removal" operator needs to be designed to remove part of the mediated schema using X's local schema and mappings to get the updated mediated schema. The removal operation can remove some relations, or some attributes in the relations. Further, X helps each of its acquaintances find another acquaintance in X's acquaintance list to make sure that when X leaves, the network is still connected. X does not need to build a mapping for them since the mapping can be inferred from the existing MappingTables.

Advantage: This is a better decision than the previous two strategies. First, it is not necessary to re-build the mediated schema from the very beginning every time. The original mediated schema can be fully used when building the new one. Second, the calculation time for removing part of the mediated schema is far less than re-building the whole mediated schema.
Disadvantages: The only disadvantage is that the leaving peer needs to do additional work and notify other nodes before it leaves the network.

In this strategy, peers cannot leave without notifying the other nodes. However, some existing P2P database systems also requires that some peer needs to do extra tasks before leaving the network. For example, [16] requires that only leaf nodes in their network structure can voluntarily leave the network while other nodes must find a replacement to store their information first before leaving. So this is acceptable if we require that a node need to trigger a removal operation before it leaves the network.

The removal algorithm is shown in Figure 6.4

```
Procedure RemoveLocalSchema (MS, MT, X, Map_MS_X)
/* MS is the mediated schema, MT is the corresponding MappingTable set,
   X is the local peer schema, Map_MS_X is the mapping from MS to X */
    For every mapping map_MS_X ∈ Map_MS_X
      Let R be body(LAV(map_MS_X))
      Let mt ∈ MT be the MappingTable with R as the first relation
      If mt doesn't exist
      /* R is a direct copy of local peer relation */
        Mark R in MS for later deletion
      Else
        Let r be body(GAV(map_MS_X))
        Let h be head(map_MS_X)
        For each attribute attr in h
          Let SG be the subgoals in r containing attr
          If in mt, no other relations except those in SG have corresponding
            attributes for attr
            Mark attr in MS for later deletion
    Delete all attributes and relations with deletion marks from MS to MS'
    Update MT to MT'
    Create a mapping Map_MS'_MS from MS' to MS
    Return MS', MT', Map_MS'_MS
```

Figure 6.4: Update mediated information when a peer leaves the network

The results MS', MT' and $Map_{MS'\_MS}$ need to be broadcasted to all other peers in the network.

Example 17 Consider the example proposed in Example 15 (details in Appendix A), and assume that mediated schema for A, B, C and D has been created. Peer D would like to leave the network, so D does the calculation of the new mediated schema. Figure 6.5, 6.6 and 6.7 show the network status before D's leaving.

Mediated schema MS_2 = {
flight(date, flightNo, price, company, service, class, ID, discount, depart, arrival, numLeft)
...
}

Figure 6.5: Example 17 Input Information (a)

MappingTable flight:

MS_1.flight	1	2	3	4	5	6	7	8	9	10	11
MS_1.flight	1	2	3	4	5	6	7	8			
IM_{AB}.flight	1	2	3	4	5	6					
A_flight	1	2	3								
B_flight	1	3		2	4						
B_price		1	3			2					
IM_{BC}.flight	1	2	3	4	5	6	7	8			
B_flight	1	3		2	4						
B_price		1	3			2					
C_schedule	2	3					1				
C_price						2	1	3			
IM_{CD}.flight	1	2					3	4	5	6	7
C_schedule	2	3					1				
C_price						2	1	3			
D_schedule	1	2							3	4	5

Figure 6.6: Example 17 Input Information (b)

According to the algorithm in Figure 6.4, D checks every conjunctive mapping in Map_{MS2_D}, if the LAV part is directly from a relation $r \in$

$Map_{MS_2\_D}$:
  $LV = \{$
    $MS_2\_D_1$(date, flightNo, depart, arrival, numLeft) :-
      MS_2.flight(date, flightNo, price, company, service, class, ID, discount, depart, arrival, numLeft)
    ...
  }
  $GV = \{$
    $MS_2\_D_1$(date, flightNo, depart, arrival, numLeft) :-
      D.D_schedule(date, flightNo, depart, arrival, numLeft)
    ...
  }

Figure 6.7: Example 17 Input Information (c)

D, delete r from the mediated schema MS_2 and the MappingTable MT_2. In this example, there's no such mapping. If the conjunctive mapping is obtained from a mapping between a pair of peers, let the LAV part be *lav* and let the GAV part be *gav*. Delete from the MT_2 all relations in *gav* and the intermediate relation related with *gav*. So in this example, delete "D_schedule" and "IM_{CD}.flight" from MT_2. Delete from MT_2 all columns that don't contain any values. In this example, delete the columns "depart", "arrival" and "numLeft". Update the relation in the mediated schema that represents *lav* according to the updated MappingTable. In this example, "flight(date, flightNo, price, company, service, class, ID, discount, depart, arrival, numLeft)" is updated to "flight(date, flightNo, price, company, service, class, ID, discount)".

So a new MappingTable MT_3, shown in Figure 6.8, can be obtained. A new mediated schema MS_3 can also be obtained, shown in Figure 6.9. D also creates a mapping from MS_3 to MS_2, shown in Figure 6.10.

D passes MS_3, MT_3 and $Map_{MS_3\_MS_2}$ to all the other peers in the network and leaves the network.

□

MappingTable flight:

MS_1.flight	1	2	3	4	5	6	7	8
MS_1.flight	1	2	3	4	5	6	7	8
IM_{AB}.flight	1	2	3	4	5	6		
A_flight	1	2	3					
B_flight	1	3		2	4			
B_price		1	3			2		
IM_{BC}.flight	1	2	3	4	5	6	7	8
B_flight	1	3		2	4			
B_price		1	3			2		
C_schedule	2	3					1	
C_price						2	1	3

Figure 6.8: Example 17 Output (a)

Mediated schema MS_3 = {
    flight(date, flightNo, price, company, service, class, ID, discount)
    ...
}

Figure 6.9: Example 17 Output (b)

6.2.1 Section Summary

In this section, we have explored four possible ways of updating the mediated schema when a peer leaves the network. We have also illustrated the advantages and disadvantages of each approach with examples. We finally presented an algorithm to remove a peer schema's information from the mediated schema based on the fourth approach. The fourth approach outperforms all other approaches because with the support of the removal operation, all previously constructed applications can still be available, all the peers are still connected, and no redundant work will be resulted.

6.3 Evolution of a Peer Local Schema

If peer E's local database schema evolves from S_E to S'_E after the system setup phase, the mediated schema M needs to be changed accordingly. A

$Map_{MS_3\_MS_2}$:
  $LV = \{$
    $MS_3\_MS_{21}$(date, flightNo, price, company, service, class, ID, discount) :-
      MS_3.flight(date, flightNo, price, company, service, class, ID, discount)
    ...
  }
  $GV = \{$
    $MS_3\_MS_{21}$(date, flightNo, price, company, service, class, ID, discount) :-
      MS_2.flight(date, flightNo, price, company, service, class, ID,
        discount, depart, arrival, numLeft)
    ...
  }

Figure 6.10: Example 17 Output (c)

simple, however effective, solution to this case is to first treat E as leaving the network (removing S_E from M), then joining the network with S'_E.

Chapter 7

A Study of Mapping Composition

We have presented our algorithms of creating a mediated schema and mappings to local sources in a PDMS in Chapter 5. We further discussed how to update the mediated schema in Chapter 6. In a PDMS dealing with query processing, mapping composition is always a main issue to consider. How well the mapping Map_{A_C} keeps the information of mappings Map_{A_B} and Map_{B_C} largely decides how well the PDMS can transfer information among different peers. Thus, we make a study of mapping composition in this chapter.

Note that our focus in this thesis is to understand the fundamentals of mapping composition in this context. As such, our algorithm is not designed to handle *all* possible patterns, but rather to focus on those that are the most common. In particular, we only consider input mappings to be mappings with the same *Concept* (Definition 3), ignoring such complicated factors as self-join and self-restrictive component (Definition 6). On the other hand, using MePSys is actually transferring the problem of mapping composition into another: using the mediated schema to relate different schemas, which is more comprehensible. However, we make a general comparison here to see how well each algorithm works in different circumstances.

Section 7.1 presents four examples with the intuition of where the complexity comes from. Section 7.2 shows the analysis results.

7.1 Complexity: Where Difficulties Come From

In section 2.2, we have presented the definition of mapping composition and briefly introduced the complexity of mapping composition in first order logic

language. In this section, we present several examples to show where difficulties come from for the problem of mapping composition. These examples are well-studied in [20] and [11]. We quote them here using their original mapping language. The translation to conjunctive mappings are trivial.

Madhavan et al's work [20] introduced the complexity of mapping composition by analyzing the number of composed mappings. Take a look at the following two examples.

Example 18 Assume the following mappings from [20]:

$$M_{A-B} = \{a(x,y) \subseteq b(x,x_1), b(x_1,y)\}$$
$$M_{B-C} = \{b(x,x_1), b(x_1,x_2), b(x_2,y) \subseteq c(x,y)\}$$

The final composed mapping set M_{A-C} involves more than one formulas:

$$a(x,x_1), a(x_1,x_2) \subseteq c(x,y_1) \quad (18.a)$$
$$a(x_1,x_2), a(x_2,x) \subseteq c(y_1,x) \quad (18.b)$$
$$a(x,x_1), a(x_1,x_2), a(x_2,y) \subseteq c(x,y_1), c(y_1,y) \quad (18.c)$$

□

Example 18 shows that the number of composed mappings does not depend on the number of the input mappings.

To analyze Example 18, [20] illustrates relation a, b and c using path length. Suppose b encodes all the edges of a graph G. Relation $b(x,y)$ means there's a path from x to y whose path length is one. Relation $a(x,y)$ means there's a path from x to y whose path length is two. Similarly, relation $c(x,y)$ means there's a path from x to y whose path length is three. So the mapping in M_{A-B} means a is a subset of the node pairs with paths of length two in G, and mapping in M_{B-C} means all node pairs in G with path of length three are a subset of c.

(18.a) describes the fact that if there's a path of length four starting from x, then there is a path of length three starting from x. (18.b) shows that if there's a path of length four ending at x, then there's a path of length

three ending at x. (18.c) tells us if there's a path of length three in a, then there is a path of length two in c. All of the above three mappings can be inferred from M_{A-B} and M_{B-C} as the mapping from A to C.

Example 19 Assume the following mappings from [20]:

$$\begin{aligned} M_{A-B} = \{&a_{rg}(x,y) \subseteq b_r(x,x_1), b_g(x_1,y) \\ &a_{gg}(x,y) \subseteq b_g(x,x_1), b_g(x_1,y)\ \} \\ M_{B-C} = \{&b_r(x,x_1), b_g(x_1,x_2), b_g(x_2,y) \subseteq c_{rgg}(x,y) \\ &b_g(x,x_1), b_g(x_1,y) \subseteq c_{gg}(x,y)\} \end{aligned}$$

This example is also originally presented in [20]. The final composed mapping is an infinite set of mappings:

$$a_{gg}(x,y) \subseteq c_{gg}(x,y) \quad (19.a)$$

$$a_{rg}(x,x_1), a_{gg}(x_1,x_2) \subseteq c_{rgg}(x,y_1) \quad (19.b)$$

$$a_{rg}(x,x_1), a_{gg}(x_1,x_2), ..., a_{gg}(x_n,x_{n+1}) \subseteq c_{rgg}(x,y_1), c_{gg}(y_1,y_2), ..., c_{gg}(y_{n-1},y_n) \quad (19.n)$$

□

Example 19 tells us that the composition of finite mappings may result in infinite set of composed mappings.

To analyze Example 18, [20] uses colored edges to illustrate this example. Informally, each above equation captures the fact that if there's a path starting from a red edge followed by $2n+1$ green edges, then there must be a path starting from a red edge followed by $2n$ green edges. Each mapping only encodes finite steps in a path. Each time, a red edge can be added to the beginning of several green edges, which will cause a new mapping. All of these mappings cannot be expressed by others. So an infinite mapping set will become the result of mapping composition in Example 19.

In Fagin et al's work [11], they analyzed mapping composition problem by proving that it is NP-complete if the mapping is in a first-order logic language. They further proved that using Second-Order Logic mapping language, all composed mapping can be expressed.

Example 20 Assume the following mappings from [11]:
Mapping from A to B:

$$\forall e\ (Emp(e) \rightarrow \exists\ m\ Mgr1(e,m)) \tag{20.1}$$

Mapping from B to C:

$$\forall e \forall m\ (Mgr_1(e,m) \rightarrow Mgr(e,m)) \tag{20.2}$$
$$\forall e\ (Mgr_1(e,e) \rightarrow SelfMgr(e)) \tag{20.3}$$

Fagin et al. proved that the composition of Mapping A-B and B-C is not definable by any finite set of source-to-target tuple generated dependencies, is not first-order-definable and is not definable in Datalog [11]. However, this can be expressed using second-order logic.
Mapping from A to C:

$$\exists\ f\ (\forall\ e(Emp(e) \rightarrow Mgr(e,f(e)) \wedge \forall\ e(Emp(e)(e = f(e)) \rightarrow SelfMgr(e))).$$

□

Example 20 shows that the composed mapping of two mappings in first-order logic might not be expressed by first-order logic. Intuitively, the difficulty in this example comes from the second attribute of $Mgr1(e,m)$. In mapping (20.1), m is an existential variable. It is not bound by any of the variables appearing in the left-hand side. However, in (20.3), the right-hand side is actually bound by the second attribute of $Mgr1(e,m)$. m acts as a selector which decides when $Mgr1$ can be mapped to Mgr and when to $SelfMgr$. As there is no information about m in Emp, such a selector is lost in schema A. Thus mapping composition for this example will fail if it is expressed in first-order logic.

7.2 Exploring Possible Patterns in Mapping Composition

Previous work [11, 20] defined when mappings would be difficult or impossible in several ways. For example, Fagin et al.'s proved and analyzed when

the definability and computational complexity of the composition of two schema mappings can be determined. They showed that "the composition of a finite set of full source-to-target tuple-generating dependencies (source-to-target tgds) with a finite set of source-to-target tgds is always definable by a finite set of source-to-target tgds, but the composition of a finite set of source-to-target tgds with a finite set of full source-to-target tgds may not be definable by any (finite or infinite) of source-to-target tgds; furthermore, it may not be definable by any formula of least fixed-point logic, and the associated composition query may be NP-complete" [11]. For clarity, Kolaitis summarizes their results in Table 7.1.

Table 7.1: A Summary of Mapping Composition Complexity from [11, 18]

Map_{A_B}	Map_{B_C}	Map_{A_C}	Composition Query
finite set of full s-t tgds $\varphi(x) \rightarrow \psi(x)$	finite set of s-t tgds $\varphi(x) \rightarrow \exists\, y\ \psi(x,y)$	finite set of s-t tgds $\varphi(x) \rightarrow \exists\, y\ \psi(x,y)$	in P
finite set of s-t tgds $\varphi(x) \rightarrow \exists\, y\ \psi(x,y)$	finite set of (full) s-t tgds $\varphi(x) \rightarrow \psi(x)$	may not be definable: by any set of s-t tgds; in FO-logic; in Datalog	in NP; can be NP-complete

In their analysis, they divided the mapping composition patterns in to two main categories based on whether the first mapping is a finite set of full s-t tgds or not. If the first mapping is a finite set of full s-t tgds, the mapping composition complexity falls into P space. If not, the complexity is likely to fall into NP.

Our work requires composition from a different angle: relating different source schemas not by the composed mappings but by the mediated schema. This causes us to want to explore when our algorithm is possible.

In this section, we make a study of the mapping composition problem based on [11] and [20]. Rather than only focus on these two categories that Fagin et al. adopted, we try to explore different factors that can cause the complexity in mapping composition and come up with 36 patterns which can

be included into the above two main categories, but in smaller granularity.

7.2.1 Different Patterns of Mapping Composition

We make a comparison with Piazza mapping composition algorithm and Fagin et al's Second-Order Logic mapping composition algorithm in Table 7.2 based on six criteria that form 36 mapping composition patterns. The main idea of this study is to see for all possible mapping composition patterns, how well our approach of relating the local schemas using the mediated schema can perform and whether the complexity that are raised in previous work would also make our approach difficult and impossible in certain circumstances.

The six criteria for the mapping composition comparison study are listed as follows. It is possible that there exist other criteria for the problem of mapping composition. The criteria below are those that we think essential and clear.

- Whether each mapping is a 1-subgoal-to-1-subgoal mapping or a m-subgoal-to-n-subgoal mapping.
- Whether the first mapping Map_{A_B} is a full set of tgds or not.
- Whether the second mapping Map_{B_C} is a full set of tgds or not.

  This criteria with the previous one are used to judge whether mapping composition is decidable for each specific case, which is proposed by Fagin et al. in deciding the complexity of mapping composition problem.

- Whether the existential attributes in the second schema B map to the third schema C.

  This criteria is complementary to the previous two, and is very important to decide whether the mapping composition will fall into NP. Just as we show in Table 7.2, there are a large set of mapping composition that can be easily processed without any complexity even though the first mapping is not a full set of tgds. A simple explanation for this is

that the existential attribute in the second schema is the one that hinders the first mapping to be a full set of tgds, however, this attribute do not act in the mapping composition. Thus such attributes will not cause any difficulty in computing a composed mapping.

- Whether there exists self-restrictive component for schema B. *Self-restrictive component* is defined in Definition 6. Self-restrictive component is a factor that causes conflict in *Concept* for two mappings. On the other hand, self-restrictive component is a cause of complexity in the mapping composition problem as is shown in Example 20.

Definition 6 (Self-Restrictive Component) We say that the component C_1 is a self-restrictive component if: (1) C_1 satisfies the following condition:

- C_1 is a component from CM_1;

  C_2 is a component from CM_2;

  C_1 and C_2 are components over the same schema, and: $\text{IDB}(C_1) = \text{IDB}(C_2)$

and (2) Q_1 and Q_2 constructed below satisfy $Q_1 \subseteq Q_2$ & $Q_1 \not\supseteq Q_2$.

- Let $name(sg)$ be the relation name of the subgoal sg;

  Let $sg\_names(Q)$ be the names of all of the relations in query Q;

  Let $overlap\_names = \{name(sg) \mid name(sg) \in sg\_names(C_1)$ and $name(sg) \in sg\_names(C_2)\}$;

  Let $C_{1overlap} = \{sg \mid name(sg) \in overlap\_names$ and $sg \in body(C_1)\}$;

  $C_{2overlap} = \{sg \mid name(sg) \in overlap\_names$ and $sg \in body(C_2)\}$.

- We now create new queries Q_1 and Q_2 that describe the overlapping parts of C_1 and C_2 respectively:

  Let $IDB(Q_1)$ and $IDB(Q_2)$ be $IDB(CM_1)$ (which, by the above requirement, is also equal to $IDB(CM_2)$;

  Let $subgoals(Q_1) = C_{1overlap}$, and $subgoals(Q_2) = C_{2overlap}$;

Let all variables in Q_1 and Q_2 be distinguished. That is let $vars(head(Q_1))$ $= vars(subgoals(Q_1))$ and let $vars(head(Q_2)) = vars(subgoals(Q_2))$.

□

- Whether there are any composed non-identical self-join in the mappings.

  Composed Non-identical Self-Join Components is defined in Definition 7. Composed non-identical self-join components are the main cause for the complexity in mapping composition problem for mappings with self-join. For mappings with composed non-identical components without self-join, there is no such complexity. This is because, every subgoal in one component can either relate to exactly one subgoal in the other component or cannot find a subgoal in the other at all. But when mappings with composed non-identical self-join components, multiple choices will occur when relating subgoals from one component to the other.

Definition 7 (Composed Non-identical Self-Join Components) Assume there exists mappings A_B and B_C. Let $a\_b \in A\_B$ and $b\_c \in B\_C$ be intersections. Let $a\_b(b)$ be the b component of the intersection a_b. Let $b\_c(b)$ be the b component of the intersection b_c. $a\_b(b)$ and $b\_c(b)$ are composed non-identical self-join Components if 1) $a\_b(b) \neq b\_c(b)$ but $\exists$ a relation r s.t. $r \in body(a\_b(b))$ and $r \in body(b\_c(b))$, and 2) r appears at least twice in at least one of $a\_b(b)$ and $b\_c(b)$. □

Table 7.2: Different Patterns for Mapping Composition

Pattern num	1-to-1 or m-to-n	Map_{A_B} full	Map_{B_C} full	existential attr in B map to C	self-restrictive	composed non-identical self-join	Piazza	MePSys	Second-Order Logic algo
1	1:1	✓	✓	/	X	/	✓	✓	✓
2	1:1	✓	✓	/	✓	/	✓	X	✓
3	1:1	✓	X	/	X	/	✓	✓	✓
4	1:1	✓	X	/	✓	/	✓	X	✓
5	1:1	X	✓	X	X	/	✓	✓	✓
6	1:1	X	✓	X	✓	/	✓	X	✓
7	1:1	X	✓	✓	X	/	*note*1	✓	✓
8	1:1	X	✓	✓	✓	/	*note*1	X	✓
9	1:1	X	X	X	X	/	✓	✓	✓
10	1:1	X	X	X	✓	/	✓	X	✓
11	1:1	X	X	✓	X	/	*note*1	✓	✓
12	1:1	X	X	✓	✓	/	*note*1	X	✓
13	m:n	✓	✓	/	X	X	✓	✓	✓
14	m:n	✓	✓	/	X	✓	✓	X	*note*2
15	m:n	✓	✓	/	✓	X	✓	X	✓
16	m:n	✓	✓	/	✓	✓	✓	X	*note*2
17	m:n	✓	X	/	X	X	✓	✓	✓
18	m:n	✓	X	/	X	✓	✓	X	*note*2

Pattern num	1-to-1 or m-to-n	Map_{A_B} full	Map_{B_C} full	existential attr in B map to C	self-restrictive	composed non-identical self-join	Piazza	MePSys	Second-Order Logic algo
19	m:n	✓	X	/	✓	X	✓	X	✓
20	m:n	✓	X	/	✓	✓	✓	X	*note*2
21	m:n	X	✓	X	X	X	✓	✓	✓
22	m:n	X	✓	X	X	✓	✓	X	*note*2
23	m:n	X	✓	X	✓	X	✓	X	✓
24	m:n	X	✓	X	✓	✓	✓	X	*note*2
25	m:n	X	✓	✓	X	X	*note*1	✓	✓
26	m:n	X	✓	✓	X	✓	*note*1	X	*note*2
27	m:n	X	✓	✓	✓	X	*note*1	X	✓
28	m:n	X	✓	✓	✓	✓	*note*1	X	*note*2
29	m:n	X	X	X	X	X	✓	✓	✓
30	m:n	X	X	X	X	✓	✓	X	*note*2
31	m:n	X	X	X	✓	X	✓	X	✓
32	m:n	X	X	X	✓	✓	✓	X	*note*2
33	m:n	X	X	✓	X	X	*note*1	✓	✓
34	m:n	X	X	✓	X	✓	*note*1	X	*note*2
35	m:n	X	X	✓	✓	X	*note*1	X	✓
36	m:n	X	X	✓	✓	✓	*note*1	X	*note*2

1. *note*1: Logically speaking, the Piazza method is not rigorous enough to handle such cases. For example, just as what we have explained in Example 2, constraint on y_1 is lost in the composed mapping. We call the case that constraint specified in the original mappings but lost in the composed mapping a *constraint loss* case.

2. *note*2: Results are logically correct, but are very complicated, not meaningful and easy to understand. Take Example 21 as an example, which is the same as Example 2 but all the variables are renamed.

Example 21 Use the Second-Order Logic Mapping Composition Algorithm to process the following mappings.

M_{A_B}:

$$\forall x\ \forall y(a(x,y) \rightarrow \exists x_1 b(x,x_1), b(x_1,y))$$

M_{B_C}:

$$\forall x\ \forall x_1\ \forall x_2\ \forall y(b(x,x_1), b(x_1,x_2), b(x_2,y) \rightarrow c(x,y))$$

Using the Second-Order Logic Mapping Composition Algorithm, the result is a set of eight very complicated logic expression. Details and results of this example are shown in Appendix B. Compared to the results given by Piazza in Example 18, this result is much too complicated and hard to read. □

7.2.2 An Analysis of Mapping Composition Patterns

From Table 7.2, the following rules can be concluded.

1. Whether Piazza method is expressive or not depends entirely on whether existential attributes in the second schema are mapped to the third schema.

2. The Second-Order Logic Mapping Composition algorithm can handle cases with composed non-identical self-join components. However, the results do not contain any semantic information, only logically correct. The representation is complicated and hard to understand.

3. For 1:1 mappings, whether MePSys can handle the pattern depends on whether self-restrictive component exists. If self-restrictive components exist, conflicts for the same concept will exist in the mappings so that no relation in the mediated schema for a common concept can be created.

4. For m:n mappings, whether MePSys can handle the pattern depends on whether composed non-identical self-join components exist in the pattern. For now, MePSys has yet to realize the mediation of schemas if mappings contain composed non-identical self-join components. We put this into our future work.

5. For the cases when Map_{A_B} is a full set of tgds, mapping composition can succeed using any of the three algorithm.

6. For the cases when Map_{A_B} is a not a full set of tgds, it is not always the case that mapping composition will be undecidable. It will depend on whether existential attributes in schema B can be mapped to schema C.

Analyze the three approaches, especially MePSys, using the results of Table 7.2. The two mapping composition algorithms, Piazza approach [20] and the Second-Order Logic algorithm [11], can handle all of the patterns, though in some cases, their composed mappings are not rigorous enough or not comprehensible enough. MePSys will fail to handle mapping composition whenever self-restrictive components or composed non-identical self-join components exist. However, self-restrictive components will cause a conflict in the mappings to have the same *Concept*, and MePSys will only merge schemas when mappings express the same *Concept*. Thus, mappings with self-restrictive component are out of the consideration of MePSys.

Aside from these two special groups of mapping composition, the approach of using PSM algorithm to build a mediated schema and using the mediated schema to relate different sources is decidable.

Comparatively, whenever Second-Order Logic algorithm has trouble in presenting the results, our approach also has trouble of that pattern. This

is because all such patterns contain self-joins. In our current approach, self-join components are not considered for the input mappings. We have yet to study and analyze when mappings with self-join components can represent the same *Concept* and when cannot. This will be our future work. On the other hand, when Piazza approach is weak in presentation, it is not always the case that the MePSys will have trouble. This is because, Piazza approach is weak in presentation when existential attributes in the second schema map to the third one while our approach transfers the mapping composition problem to the one that how each local schema relates to the mediated schema.

Chapter 8

System Implementation and Experiment

In previous chapters, we discussed MePSys, a PDMS supporting a mediated schema to help translate queries. We also presented our PSM algorithm of building a mediated schema, creating mappings to local sources and translating queries among different peers. In this chapter, we study the performance of MePSys in the case of creating a mediated schema and mappings, updating the mediated schema and mappings, and query reformulation.

8.1 System Implementation and Setup

We choose FreePastry [2] as a provider of network layer to our MePSys. FreePastry is a generic, scalable substrate for P2P applications. It uses efficient routing strategy, and each node maintains a routing table which keeps track of its immediate neighbors. FreePastry also provides the functionality of notifying each peer application of message arrival, neighboring node failures, etc. The expected number of forwarding steps in FreePastry overlay network is $O(logN)$. We use FreePastry version 1.4.4 for the network layer. All implementation is written in Java, which makes it cross-platform.

We setup our experiment in Emulab [1], a network emulation testbed, to get the realistic P2P environment for our system. Emulab provides the ability to access a set of different machines to emulate nodes in a real network. Network bands and message delays can also be set. Different numbers of peer nodes (12, 16, 20 and 24) have been used for the performance of schema mediation and query reformulation in MePSys. In total, we get 40 machines from Emulab, including 15 used for network control and 1 for experiment control, leaving us with a maximum of 24 different peers. Each Emulab

machine that we get provides 900M memory with 2992.787 MHz processor. The quality of such machines is similar to that of a peer in a P2P network since each peer needs to act as both a server and a client. We choose 70ms delay for each message and 50M bandwidth[1] to simulate a real network environment. Although Emulab was only able to provide 24 peers for our experiment, which is less than the hundreds or thousands of peers that may exist in a P2P network, we believe this to be a reasonable number of peers to test for a PDMS. In a PDMS, the system is relying on the translation of semantics, and while our system does improve the semantics of query translation through the introduction of Concept (Chapter 4), attempting to semantically compose query rewrites across hundreds of peers with heterogeneous schemas will not result in a semantically meaningful result.

8.2 Input Schemas and Mappings

We based the relational schemas at each peer on the schemas in TPC-H [4], where each schema contains 8 relations, each with an average of 8 attributes. We use this information to automatically generate a relational schema as a local database schema for each peer in MePSys, making our peer schema as realistic as possible. We further generate mappings between pairs of acquainted nodes; as we will see, the mappings varied from experiment to experiment, but we controlled (1) the average number of acquaintances per peer (2) the average number of relations per peer schema (3) the average number of attributes in a relation.

8.3 Experiment 1: Schema Mediation

The main purpose of this experiment is to see the performance of the algorithms described in Chapter 5 and to compare the performance when different numbers of peers are involved in the system setup phase.

[1]The bandwidth is not an important factor in MePSys. We ran the same experiment using 1M bandwidth, and no significant difference in time occurred in the result.

In Theorem 1, we gave an upper bound of time for system setup phase. The time of schema mediation in the PDMS is decided by the number of hops when the last node in the network received all node information: $\max_E\{\max_F\{shortestPath(A,F) + shortestPath(F,E)\}\}$, where A is the mediation starting node, E and F represent all nodes in the network. Figure 8.1 shows the result of mediating 12, 16, 20 and 24 peers' information respectively. When peer number is fixed, the time of completing a mediated schema is linear and proportional to the number of hops. The time of each hop is decided by two factors: delay time for each message and message transferring time. In our scenario, the time of transferring a mediated schema cannot be ignored. The time for transferring a mediated schema is highly related to the size of the mediated schema. When the number of peers gets larger, the mediated schema also increases, resulting the time of each hop to increase as well. Thus the time of each hop for mediating 12, 16, 20 and 24 peers are different. On the other hand, when the number of peers in a network is fixed, the time for each hop is similar, because the messages the nodes are passing each time are of similar size. Considering the case of 24 peers with 20 hops, the total time of mediating the 24 peer schemas only costs 31 seconds, which is quite a satisfactory time cost for the system setup phase.

8.4 Experiment 2: Query Reformulation

In this experiment, we test the performance of our query reformulation algorithm and compare the local computation time with the whole query reformulation time in the network. Since the query reformulation algorithm is very fast, for each query on each peer, the query is reformulated 10 times, and this query reformulation is reported as local computing time. This way, though the time was artificially inflated, it would not be subject to timing errors caused by having to measure very small numbers.

Time of query reformulation and broadcasting in the whole network is proportional to the topology depth when each query message size is similar.

We define *Topology Depth* as the maximum of the shortest path for all

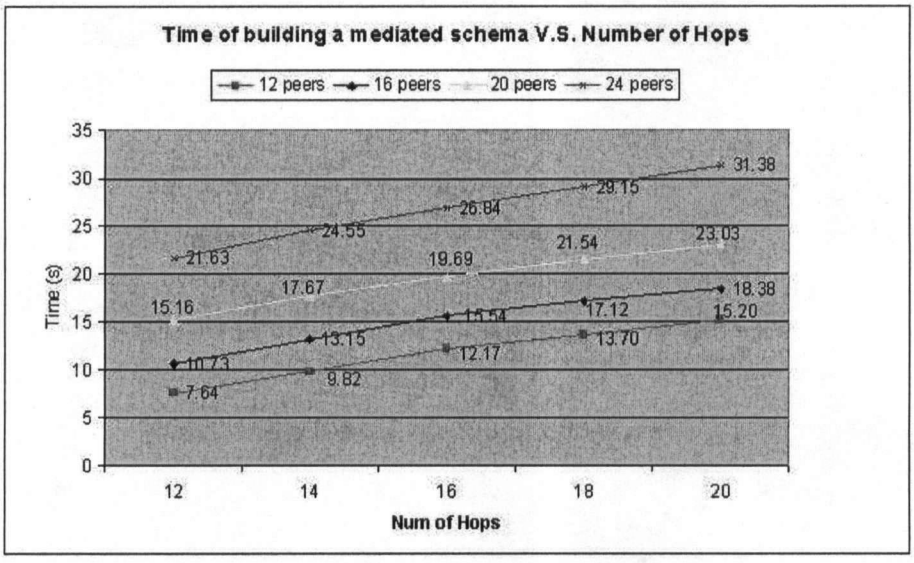


Figure 8.1: Time of building a mediated schema V.S. Number of Hops

nodes from the node where query is initially posed.

$$TopologyDepth = \max\{shortestPath(A, E)\}$$

where A is the node where query is posed, E is all other nodes in the network.

Topology depth takes an important role in deciding the total time for query reformulation in the network. This is because queries will be sent to all other acquaintances except the message source, and each node will not process the same query twice. What's more, in most cases, query messages are small. Figure 8.2 shows the time results of completing the query reformulation and broadcasting in the network. We posed a query with 2 subgoals on one peer schema. For each topology depth from 4 to 9, we run experiment on 12, 16, 20 and 24 peers respectively. Results show that for queries (with relatively small size under 1k) sent to topologies of the same topology depth, there is almost no time difference to reformulate the query among the four different sets. On the other hand, results in Figure 8.2 show that the time of query reformulation and broadcasting is exactly proportional to

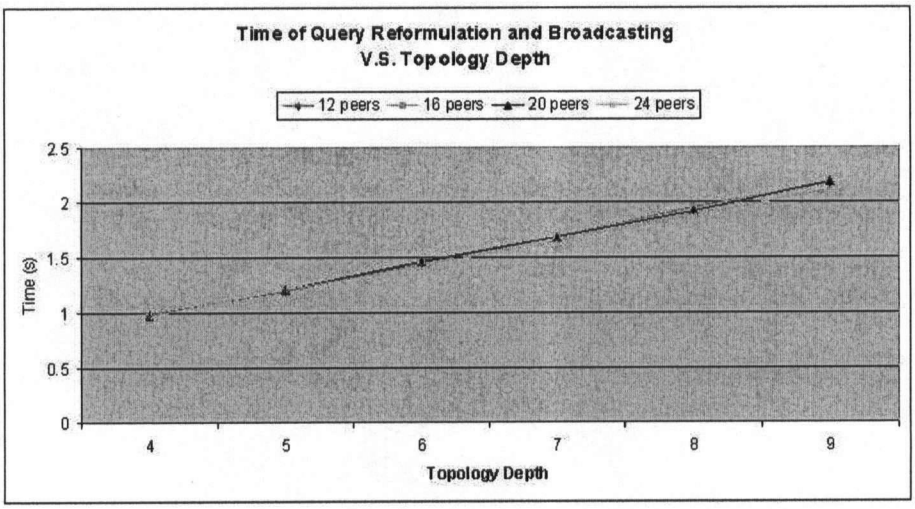


Figure 8.2: Time of Query Reformulation and Broadcasting V.S. Depth of Topology

the topology depth of the network.

We further compared the quality of the query reformulation algorithm when different queries are posed. We fixed the network topology and posed ten different queries to a source peer. Each query contain subgoals ranging from 1 to 10. For some peer when queries cannot be reformulated, the local reformulation time will be very little. Thus for each query, we compare the total reformulation and broadcasting time with the maximum local peer computing time for 10 times query reformulation.

Figure 8.3 shows that as the number of subgoal grows, the time spent on local reformulation time also increases. This is as anticipated since our algorithm reformulates queries based on each subgoal. Another finding from Figure 8.3 is that when the subgoal number increases, the time for completing the query in the whole network also increases. This is because when the subgoal number increases, the size of the message containing that query also increases, which causes the time of each hop to increase as well. However, when processing queries with no more than 3 subgoals, the reformulation and broadcasting such queries is still close to constant. From Figure 8.3, we

can conclude that even for 10 times of query reformulation, maximum local reformulation time is always less than 3% of the total network time.

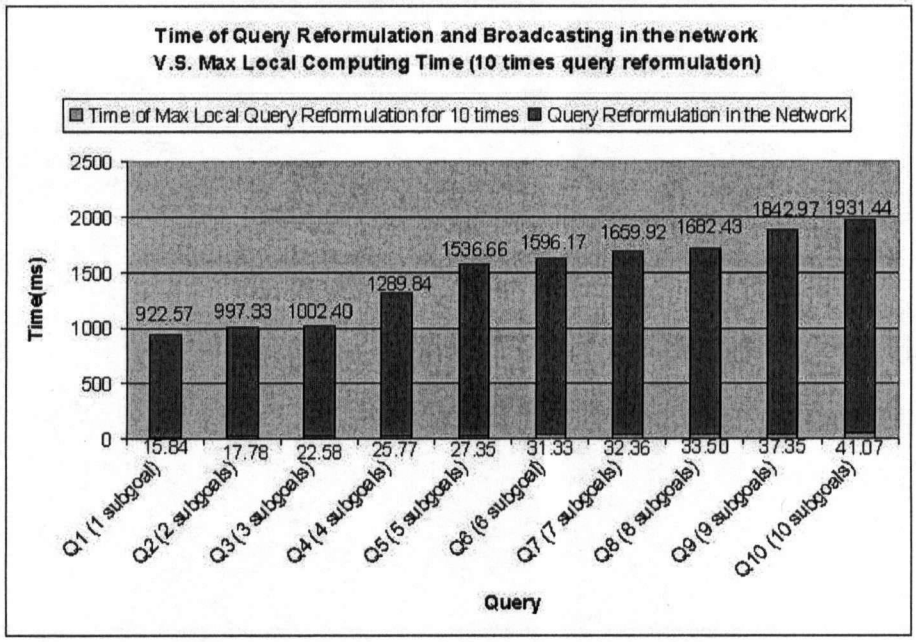


Figure 8.3: Time of Query Reformulation in the network V.S. Max Local Computing Time for 10 times Query Reformulation

8.5 Experiment 3: Updating the Mediated Schema

Next, we tested the performance of updating the mediated schema in the stable state, i.e., after the system setup period. This experiment is a follow-up experiment of the first experiment, Schema Mediation (Section 8.3), and is designed as follows: Assume all peers maintain the most updated mediated schema and GLAV mappings to local schema. A new peer Z joins the network and creates a mapping to one of the peer, say A, which triggers a new schema mediation process. After that, A broadcasts the new mediated schema to all other peers, and then all peers, upon receiving an

updating message, update their locally maintained mediated schema and other information.

Ten different experiments, representing ten different topologies, are executed in order to get the following time for each setting.

1. t_1 = time of initial system setup phase
2. t_2 = time of computing the new M at peer A
3. t_3 = average time of updating M at each peer except A
4. t_4 = total network time for updating M in the whole network after Z joins the network

Time t_1 is directly obtained from the statistics in Experiment 1. Thus, it is possible to compare the time of updating the mediated schema with the initial system setup time for a fixed topology.

"Topology Depth", which is defined in Experiment 2, is used to note down the topology depth for each experiment. Since after updating the mediated schema at A, the same updating message will only be processed at each peer once, the time for the whole updating process to complete depends on the topology depth.

Table 8.1: Updating the Mediated Schema for Adding a New Peer

Experiment	Topology Depth	t_1(ms)	t_2(ms)	t_3(ms)	t_4(ms)	t2/t4
16 peers 12 hops	4	10733.86	71.33	0.0099	5868.13	1.21%
16 peers 14 hops	5	13147.71	71.10	0.0105	7325.73	0.97%
16 peers 16 hops	7	15542.58	69.16	0.0099	11477.08	0.46%
16 peers 18 hops	8	17117.51	71.49	0.0105	12766.28	0.56%
16 peers 20 hops	9	18376.80	71.38	0.0106	13075.01	0.54%
20 peers 12 hops	4	15160.07	91.50	0.01	5297.89	1.73%
20 peers 14 hops	5	17672.95	90.57	0.0103	6616.83	1.37%
20 peers 16 hops	6	19693.76	91.01	0.0105	7921.49	1.15%
20 peers 18 hops	8	21542.09	87.99	0.009	9591.46	0.92%
20 peers 20 hops	7	23028.56	90.27	0.009	7940.53	1.14%

Table 8.1 shows the results of different time w.r.t. the ten topologies. The computation of a new mediated schema at peer A, t_1, is always around

70 ms for 16 peers, and 90 ms for 20 peers, less than 2% of t_4, the total time for updating a mediated schema in the network. The time of updating local mediated information in all other peers, t_3, is always about 0.01 ms, which can be ignored compared to the time spent for the network.

Chapter 9

Conclusion and Future Work

In this thesis, we presented MePSys, a relational peer data management system. Our key contributions are:

- We presented a mechanism to support a mediated schema in PDMS which allows easier access to more information.
- We defined what a concept means in the case of conjunctive mapping and how it impacts the understandability of the mapping and mediated schema.
- We introduced a construct MappingTable which helps us to transform unstructured mapping information to structured forms and which is easier to construct mappings from the mediated schema to local sources.
- We also provided an approach to create mediated schema in PDMS and get mappings to local sources.
- We further explored how to update the mediated schema in the steady state.
- We finally implemented the system and showed the experimental results.

In future work, we will extend this work to a more generic method. First, reconsidering the definition of *Concept*, we want to figure out how to distinguish mappings with the same semantic information while using different IDB names. We also would like to explore the semantic issues when a broader range of mappings are considered, e.g., mappings with self-joins. Second, more optimization issues can be considered in the future system since in

current MePSys, each mediated schema message passed among peers is still large. There are many possible ways to decrease the size of the messages by sharing information between different constructs. Good optimization can significantly decrease the schema mediation time in the system setup phase. Third, we would like to explore some better approaches for updating the mediated schema when local schema evolves.

Bibliography

[1] Emulab. https://www.emulab.net/.

[2] Freepastry. http://www.freepastry.org/.

[3] Napster. http://www.napster.com/.

[4] Tpc-h. http://www.tpc.org/tpch/.

[5] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R.J. Miller, and J. Mylopoulos. The hyperion project: From data integration to data coordination. *SIGMOD Record*, 2003.

[6] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18:323–364, 1986.

[7] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. In *WebDB*, 2002.

[8] A. Bonifati, E. Q. Chang, T. Ho, L. V.S. Lakshmanan, and R. Pottinger. Heptox: Marrying xml and heterogeneity in your p2p databases (system demostration). In *VLDB*, 2005.

[9] Elaine Q. Chang. Schema mapping and query translation in heterogeneous peer-to-peer xml databases. Technical report, The University of British Columbia, 2005. Master thesis.

[10] R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, 2003.

[11] R. Fagin, P.G. Kolaitis, L. Popa, and W. Tan. Composing schema mappings: Second-order dependencies to the rescue. In *PODS*, 2004.

[12] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *the 16th Nat. Conf. on Artificial Intelligence (AAAI99)*, pages 67 – 73, 1999.

[13] Steven Gribble, Alon Halevy, Zachary Ives, Maya Rodrig, and Dan Suciu. What can databases do for peer-to-peer. In *WebDB*, 2001.

[14] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *ICDE*, 2003.

[15] Alon Y. Halevy, Zachary G. Ives, Peter Mork, and Igor Tatarinov. Piazza: Data management infrastructure for semantic web applications. In *WWW*, 2003.

[16] H.V. Jagadish, Beng Chin Ooi, and Quang Hieu Vu. Baton: A balanced tree structure for peer-to-peer networks. In *VLDB*, 2005.

[17] A. Kementsietsidis, M. Arenas, and R. J. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *SIGMOD*, 2003.

[18] Phokion G. Kolaitis. Data exchange and composition of schema mappings, 2004. Presentation slides given at Carleton University. http://www.scs.carleton.ca/~ diis/arise/Presentations/PhokionKolaitis .ppt.

[19] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

[20] J. Madhavan and A.Y. Halevy. Composing mappings among data sources. In *VLDB*, 2003.

[21] W. S. Ng, B. C. Ooi, K. Tan, and A. Zhou. Peerdb: A p2p-based system for distributed data sharing. In *ICDE*, 2003.

[22] R. Pottinger. *Processing Queries and Merging Schemas in Support of Data Integration.* PhD thesis, 2004. PhD thesis.

[23] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, Dec 2001.

[24] P. Rodriguez-Gianolli, M. Garzetti, L. Jiang, A. Kementsietsidis, I. Kiringa, M. Masud, R. Miller, and J. Mylopoulos. Data sharing in the hyperion peer database system. In *ICDE*, 2005.

[25] S. P. Shah, Y. Huang, T. Xu, M. MS Yuen, J. Ling, and B. F. Ouellette. Atlas - a data warehouse for integrative bioinformatics. In *BMC Bioinformatics*, 2005.

[26] Igor Tatarinov and Alon Halevy. Efficient query reformulation in peer-data management systems. In *SIGMOD*, 2004.

[27] Jeffrey D. Ullman. *Principle of Database and Knowledge-Base Systems.* 1989.

[28] Jeffrey D. Ullman. Information integration using logical views. In *ICDT*, pages 19–40, 1997.

Appendix A

Details for Example 15

Example 15: An example of a new peer joining the network
There are three peers in the current system: A, B and C. Mappings Map_{A_B} and Map_{B_C} are created as follows. We only consider the relations that are created by mappings for the mediated schema and omit relations that do not participate in the mappings because relations that represent a *Concept* among different peers are the core part of our discussion.

```
Mapping Map_A_B:
        flight(date, flightNo, price) :- A_flight(date, flightNo, price)
        flight(date, flightNo, price) :-
                B_flight(date, company,flightNo,service),
                B_price(flightNo, class, price)
Mapping Map_B_C:
        flight(date, flightNo, class, price) :-
                B_flight(date, company, flightNo, service),
                B_price(flightNo, class, price)
        flight(date, flightNo, class, price) :-
                C_schedule(ID, date, flightNo),
                C_price(ID, class, price, discount)
```

Shown in in Figure 6.1(a), after the setup phase, a mediated schema MS_1 has been built and maintained at each peer. The mappings from MS_1 to A, B and C and the MappingTable will also be built.

```
Mediated schema MS_1 = {
    flight(date, flightNo, price, company, service, class, ID, discount)
    ...
}
```

MappingTable flight:

MS_1.flight	1	2	3	4	5	6	7	8
IM_{AB}.flight	1	2	3	4	5	6		
A_flight	1	2	3					
B_flight	1	3		2	4			
B_price		1	3			2		
IM_{BC}.flight	1	2	3	4	5	6	7	8
B_flight	1	3		2	4			
B_price		1	3			2		
C_schedule	2	3					1	
C_price						2	1	3

Mapping $Map_{MS_1\_A}$:
$LV = \{$
$MS_1\_A_1$(date, flightNo, price) :-
MS_1.flight(date, flightNo, price, company, service, class, ID, discount)
...
$\}$
$GV = \{$
$MS_1\_A_1$(date, flightNo, price) :- A.A_flight(date, flightNo, price)
...
$\}$

Mapping $Map_{MS_1\_B}$:
$LV = \{$
$MS_1\_B_1$(date, company, flightNo, service, class, price) :-
MS_1.flight(date, flightNo, price, company, service, class, ID, discount)
...
$\}$
$GV = \{$
$MS_1\_B_1$(date, company, flightNo, service, class, price) :-
B.B_flight(date, company, flightNo, service),
B.B_price(flightNo, class, price)
...
$\}$

Mapping $Map_{MS_1\_C}$:
$LV = \{$
$MS_1\_C_1$(ID, date, flightNo, class, price, discount) :-
MS_1.flight(date, flightNo, price, company, service, class, ID, discount)
...
}
$GV = \{$
$MS_1\_C_1$(ID, date, flightNo, class, price, discount) :-
C.C_schedule(ID, date, flightNo),
C.C_price(ID, class, price, discount)
...
}

After building the mediated schema MS_1 and the mappings to local sources, peer B also built a database application APP_B over MS_1. So the information stored at each peer is as follows.

- Peer A: schema A, Map_{A_B}, MS_1, $Map_{MS_1\_A}$, MT_1.
- Peer B: schema B, Map_{A_B}, Map_{B_C}, MS_1, $Map_{MS_1\_B}$, MT_1, $APP_B(MS_1)$.
- Peer C: schema C, Map_{B_C}, MS_1, $Map_{MS_1\_C}$, MT_1.

A new peer D joins in the network. It creates a mapping to peer C, shown in Figure 6.1(b).

Mapping Map_{C_D}:
flight(date, flightNo) :- C_schedule(ID, date, flightNo),
C_price(ID, class, price, discount)
flight(date, flightNo) :-
D_schedule(date, flightNo, depart, arrival, numLeft)

Using algorithms described in Chapter 5, a new mediated schema MS_2, MappingTable MT_2 and GLAV mappings can be computed.

Mediated schema $MS_2 = \{$
flight(date, flightNo, price, company, service, class, ID, discount, depart, arrival, numLeft)
...
}

MappingTable flight:

MS_1.flight	1	2	3	4	5	6	7	8	9	10	11
MS_1.flight	1	2	3	4	5	6	7	8			
IM_{AB}.flight	1	2	3	4	5	6					
A_flight	1	2	3								
B_flight	1	3		2	4						
B_price		1	3			2					
IM_{BC}.flight	1	2	3	4	5	6	7	8			
B_flight	1	3		2	4						
B_price		1	3			2					
C_schedule	2	3					1				
C_price						2	1	3			
IM_{CD}.flight	1	2					3	4	5	6	7
C_schedule	2	3					1				
C_price						2	1	3			
D_schedule	1	2							3	4	5

Two mappings will be created first.

Mapping $Map_{MS_2\_MS_1}$:
  $LV = \{$
    Q_1(date, flightNo, price, company, service, class, ID, discount) :-
      MS_2.flight(date, flightNo, price, company, service, class, ID, discount, depart, arrival, numLeft)
    ...
  }
  $GV = \{$
    Q_1(date, flightNo, price, company, service, class, ID, discount) :-
      MS_1.flight(date, flightNo, price, company, service, class, ID, discount)
    ...
  }

Mapping $Map_{MS_2\_D}$:
$LV = \{$
$MS_2\_D_1$(date, flightNo, depart, arrival, numLeft) :-
MS_2.flight(date, flightNo, price, company, service, class, ID, discount, depart, arrival, numLeft)
...
$\}$
$GV = \{$
$MS_2\_D_1$(date, flightNo, depart, arrival, numLeft) :-
D.D_schedule(date, flightNo, depart, arrival, numLeft)
...
$\}$

C passed MS_2, MT_2 and $Map_{MS_2\_D}$ to D. So the information that peer D stores is: MS_2, MT_2, $Map_{MS_2\_D}$, schema D, Map_{C_D}.

C further computes the GLAV mapping from MS_2 to C using $Map_{MS_2\_MS_1}$ and $Map_{MS_1\_C}$.

Mapping $Map_{MS_2\_C}$:
$LV = \{$
$MS_2\_C_1$(ID, date, flightNo,class, price, discount) :-
MS_2.flight(date, flightNo, price, company, service, class, ID, discount, depart, arrival, numLeft)
...
$\}$
$GV = \{$
$MS_2\_C_1$(ID, date, flightNo,class, price, discount) :-
C.C_schedule(ID, date, flightNo),
C.C_price(ID, class, price, discount)
...
$\}$

C will keep the following information: MS_2, MT_2, $Map_{MS_2\_C}$, schema C, Map_{C_D}, Map_{B_C}.

C passes MS_2, MT_2, $Map_{MS_2\_MS_1}$ to B. B computes Map_{MS2_B} using $Map_{MS_2\_MS_1}$ and Map_{MS1_B}.

Mapping $Map_{MS_2\_B}$:
$LV = \{$
$MS_2\_B_1$(date, company, flightNo, service, class, price) :-
MS_2.flight(date, flightNo, price, company, service, class, ID,
discount, depart, arrival, numLeft)
...
}
$GV = \{$
$MS_2\_B_1$(date, company, flightNo, service, class, price) :-
B.B_flight(date, company, flightNo, service),
B.B_price(flightNo, class, price)
...
}

Since B has application APP_B still using MS_1, MS_1 and the mapping MS_2 to MS_1 need to be kept as well. So the information that B stored is MS_1, MS_2, $Map_{MS_2\_MS_1}$, $Map_{MS_2\_B}$, MT_2, schema B, Map_{B_C}, Map_{A_B}.

B will pass MS_2, MT_2, $Map_{MS_2\_MS_1}$ to A.

Using $Map_{MS_2\_MS_1}$ and $Map_{MS_1\_A}$, a mapping from mediated schema MS_2 to local schema A can be computed.

Mapping $Map_{MS_2\_A}$:
$LV = \{$
$MS_2\_A_1$(date, flightNo, price) :-
MS_2.flight(date, flightNo, price, company, service, class, ID,
discount, depart, arrival, numLeft)
...
}
$GV = \{$
$MS_2\_A_1$(date, flightNo, price) :-
A.A_flight(date, flightNo, price)
...
}

A will keep the following information: MS_2, MT_2, Map_{MS2_A} and Map_{A_B}.

Thus, every peer can keep the updated mediated schema and maintain a mapping from the mediated schema to local schema. Additionally, all the previously created database applications over previous versions of the mediated schema can still be used.

Appendix B

Details for Example 21

Use the Second-Order Logic Mapping Composition Algorithm to process the following mappings.

M_{A_B}:

$$\forall x\ \forall y(a(x,y) \rightarrow \exists x_1 b(x,x_1), b(x_1,y))$$

M_{B_C}:

$$\forall x\ \forall x_1\ \forall x_2\ \forall y(b(x,x_1), b(x_1,x_2), b(x_2,y) \rightarrow c(x,y))$$

A detailed process is as follows using the Second-Order Logic Mapping Composition Algorithm.

Step zero (change variable names):

M_{A_B}:

$$\forall x\ \forall y(a(x,y) \rightarrow \exists x_1 b(x,x_1), b(x_1,y))$$

M_{B_C}:

$$\forall s\ \forall s_1\ \forall s_2\ \forall t(b(s,s_1), b(s_1,s_2), b(s_2,t) \rightarrow c(s,t))$$

Step one (split):

M_{A_B}:

$$\exists f(\forall x\ \forall y(a(x,y) \rightarrow b(x,f(x,y)), b(f(x,y),y))$$
$$\Rightarrow \{\ a(x,y) \rightarrow b(x,f(x,y)),\ a(x,y) \rightarrow b(f(x,y),y)\ \}$$

Step two (compose):

M_{B_C}:

$(b(s,s_1), b(s_1,s_2), b(s_2,t) \rightarrow c(s,t))$

$\Rightarrow \{$
$(a(x,y), b(s_1,s_2), b(s_2,t) \wedge (x = s) \wedge (f(x,y) = s_1) \rightarrow c(s,t));$
$(a(x,y), b(s_1,s_2), b(s_2,t) \wedge (f(x,y) = s) \wedge (y = s1) \rightarrow c(s,t))$
$\}$

$\Rightarrow \{$
$(a(x,y), a(x',y'), b(s_2,t) \wedge (x = s) \wedge (f(x,y) = s_1) \wedge (x' = s_1)$
$\wedge (f(x',y') = s_2) \rightarrow c(s,t));$
$(a(x,y), a(x',y'), b(s_2,t) \wedge (x = s) \wedge (f(x,y) = s_1) \wedge (f(x',y') =$
$s_1) \wedge (y' = s_2) \rightarrow c(s,t));$
$(a(x,y), a(x',y'), b(s_2,t) \wedge (f(x,y) = s) \wedge (y = s_1) \wedge (x' = s_1)$
$\wedge (f(x',y') = s2) \rightarrow c(s,t));$
$(a(x,y), a(x',y'), b(s_2,t) \wedge (f(x,y) = s) \wedge (y = s_1) \wedge (x' = s_1) \wedge$
$(f(x',y') = s_2) \rightarrow c(s,t));$
$\}$

$\Rightarrow \{$
$(a(x,y), a(x',y'), a(x'',y'') \wedge (x = s) \wedge (f(x,y) = s_1) \wedge (x' = s_1)$
$\wedge (f(x',y') = s_2) \wedge (x'' = s_2) \wedge (f(x'',y'') = t) \rightarrow c(s,t));$
$(a(x,y), a(x',y'), a(x'',y'') \wedge (x = s) \wedge (f(x,y) = s_1) \wedge (f(x',y') =$
$s_1) \wedge (y' = s_2) \wedge (x'' = s_2)\ (f(x'',y'') = t) \rightarrow c(s,t));$
$(a(x,y), a(x',y'), a(x'',y'') \wedge (f(x,y) = s) \wedge (y = s_1) \wedge (x' = s_1)$
$\wedge (f(x',y') = s_2) \wedge (x'' = s_2) \wedge (f(x'',y'') = t) \rightarrow c(s,t));$
$(a(x,y), a(x',y'), a(x'',y'') \wedge (f(x,y) = s) \wedge (y = s_1) \wedge (x' = s_1)$
$\wedge (f(x',y') = s_2)\ (x'' = s_2) \wedge (f(x'',y'') = t) \rightarrow c(s,t));$
$(a(x,y), a(x',y'), a(x'',y'') \wedge (x = s) \wedge (f(x,y) = s_1) \wedge (x' = s_1)$
$\wedge (f(x',y') = s_2) \wedge (f(x'',y'') = s_2) \wedge (y'' = t) \rightarrow c(s,t));$
$(a(x,y), a(x',y'), a(x'',y'') \wedge (x = s) \wedge (f(x,y) = s_1) \wedge (f(x',y') =$
$s_1) \wedge (y' = s_2) \wedge (f(x'',y'') = s_2) \wedge (y'' = t) \rightarrow c(s,t));$
$(a(x,y), a(x',y'), a(x'',y'') \wedge (f(x,y) = s) \wedge (y = s_1) \wedge (x' = s_1)$
$\wedge (f(x',y') = s_2) \wedge (f(x'',y'') = s_2) \wedge (y'' = t) \wedge c(s,t));$
$(a(x,y), a(x',y'), a(x'',y'') \wedge (f(x,y) = s) \wedge (y = s_1) \wedge (x' = s_1)$
$\wedge (f(x',y') = s_2) \wedge (f(x'',y'') = s_2)\ (y'' = t) \wedge c(s,t))$

}

Step three (add quantifiers):

$$
\begin{aligned}
&\Rightarrow \{\\
&\exists f \forall x \forall y \forall x' \forall y' \forall x'' \forall y'' \forall s \forall s_1 \forall s_2 \forall t\, (a(x,y), a(x',y'), a(x'',y'')\\
&\wedge (x = s) \wedge (f(x,y) = s_1) \wedge (x' = s_1) \wedge (f(x',y') = s_2) \wedge\\
&(x'' = s_2) \wedge (f(x'',y'') = t) \rightarrow c(s,t));\\
&\exists f \forall x \forall y \forall x' \forall y' \forall x'' \forall y'' \forall s \forall s_1 \forall s_2 \forall t\, (a(x,y), a(x',y'), a(x'',y'')\\
&\wedge (x = s) \wedge (f(x,y) = s_1) \wedge (f(x',y') = s_1) \wedge (y' = s_2) \wedge\\
&(x'' = s_2) \wedge (f(x'',y'') = t) \rightarrow c(s,t));\\
&\exists f \forall x \forall y \forall x' \forall y' \forall x'' \forall y'' \forall s \forall s_1 \forall s_2 \forall t\, (a(x,y), a(x',y'), a(x'',y'')\\
&\wedge (f(x,y) = s) \wedge (y = s_1) \wedge (x' = s_1) \wedge (f(x',y') = s_2) \wedge\\
&(x'' = s_2) \wedge (f(x'',y'') = t) \rightarrow c(s,t));\\
&\exists f \forall x \forall y \forall x' \forall y' \forall x'' \forall y'' \forall s \forall s_1 \forall s_2 \forall t\, (a(x,y), a(x',y'), a(x'',y'')\\
&\wedge (f(x,y) = s) \wedge (y = s_1) \wedge (x' = s_1) \wedge (f(x',y') = s_2) \wedge\\
&(x'' = s_2) \wedge (f(x'',y'') = t) \rightarrow c(s,t));\\
&\exists f \forall x \forall y \forall x' \forall y' \forall x'' \forall y'' \forall s \forall s_1 \forall s_2 \forall t\, (a(x,y), a(x',y'), a(x'',y'')\\
&\wedge (x = s) \wedge (f(x,y) = s_1) \wedge (x' = s_1) \wedge (f(x',y') = s_2)\\
&(f(x'',y'') = s_2) \wedge (y'' = t) \rightarrow c(s,t));\\
&\exists f \forall x \forall y \forall x' \forall y' \forall x'' \forall y'' \forall s \forall s_1 \forall s_2 \forall t\, (a(x,y), a(x',y'), a(x'',y'')\\
&\wedge (x = s) \wedge (f(x,y) = s_1) \wedge (f(x',y') = s_1) \wedge (y' = s_2) \wedge\\
&(f(x'',y'') = s_2) \wedge (y'' = t) \rightarrow c(s,t));\\
&\exists f \forall x \forall y \forall x' \forall y' \forall x'' \forall y'' \forall s \forall s_1 \forall s_2 \forall t\, (a(x,y), a(x',y'), a(x'',y'')\\
&\wedge (f(x,y) = s) \wedge (y = s_1) \wedge (x' = s_1) \wedge (f(x',y') = s_2) \wedge\\
&(f(x'',y'') = s_2) \wedge (y'' = t) \rightarrow c(s,t));\\
&\exists f \forall x \forall y \forall x' \forall y' \forall x'' \forall y'' \forall s \forall s_1 \forall s_2 \forall t\, (a(x,y), a(x',y'), a(x'',y'')\\
&\wedge (f(x,y) = s) \wedge (y = s_1) \wedge (x' = s_1) \wedge (f(x',y') = s_2) \wedge\\
&(f(x'',y'') = s_2) \wedge (y'' = t) \rightarrow c(s,t));\\
&\}
\end{aligned}
$$

The whole set in step three is the result of the composed mapping Map_{A_C} of Map_{A_B} and Map_{B_C}. Compared to the results given by Piazza in Example 18, this result is much too complicated and hard to read.