uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Md. Mehedi Masud

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Master of Computer Science

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Expressive Query Translation in Peer Databases

TITRE DE LA THÈSE / TITLE OF THESIS

Iluju Kiringa

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

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

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

A. El Saddik

M. Liu

Gary W. Slater

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

Expressive Query Translation in Peer Databases

by

Md. Mehedi Masud

Supervised by- Dr. Iluju Kiringa

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

Ottawa-Carleton Institute for Computer Science
School of Information Technology and Engineering
University of Ottawa

Your file Votre référence
ISBN: 0-494-11345-6
Our file Notre référence
ISBN: 0-494-11345-6

Expressive Query Translation in Peer Databases

Master of Computer Science Thesis
Ottawa-Carleton Institute of Computer Science
University of Ottawa

by Md. Mehedi Masud
May
2005

Abstract

Query processing in peer database systems has moved towards a new direction beyond mere file sharing and keyword-based information retrieval processes used in the first peer-to-peer systems to embrace data sharing and coordination across heterogeneous frontiers. The strong dynamics of peer-to-peer networks, coupled with the diversity of peer vocabularies, makes query processing in peer database systems a very challenging task. In this thesis, we propose a framework for translating expressive relational queries across heterogeneous peer databases. Our framework avoids an integrated global schema or centralized structures common to the involved peers. The cornerstone of the approach is the use of both syntax and instance level mappings that each peer constructs and shares with other peers. Based on this user provided mapping information, the algorithm applies generic translation rules to translate SQL queries. Our approach supports both query translation and propagation among the peers preserving the autonomy of individual peers. It combines two previous proposals for data sharing via query mapping across heterogeneous boundaries that were based on syntax and instance level mappings separately into a more general framework. We have developed a prototype as a query service layer wrapped around a basic service providing heterogeneity management. The prototype has been evaluated on a small peer-to-peer network to demonstrate the viability of the approach.

Acknowledgements

First of all, I glorify the greatness and bounty of almighty Allah who has bestowed on me the strength and ability without which it would not have been possible to carry out the thesis.

I am grateful to my supervisor Dr. Iluju Kiringa who has given me the opportunity to do research in the exiting and evolving field of databases and guided me to the way of innovation and novelty. His invaluable ideas and profound research experience kept me enthusiastic and optimistic all the way to the completion of this thesis. I thank him for his many hours of patience for listening to my problems. Also I am grateful to my Co-Supervisor Dr. Hasan Ural who has inspired me time to time from a different corner. Their assistance, comments, constructive criticism and positive attitude helped me proceed towards completing each step of the thesis.

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

I am also grateful to my all friends specially those in my lab who inspired and gave me suggestions.

Lastly, I thank my wife Arshi whose understanding, inspiration and constant support provided me with optimism in all situations.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and Objectives

In the past few years, Peer-to-Peer (P2P) applications have emerged as a popular way of sharing data in decentralized and distributed environments. In such environments, involved data sources, called peers, act autonomously in their sharing of data and services. A peer database system (PDBS) consists of a peer or node of a P2P network which has been augmented both with a conventional data base management system and an interoperability layer that enables data sharing across (usually) heterogeneous boundaries. The local databases on each peers are called *peer databases*. Each PDBS is independent of others and maintains its own peer databases. In addition to the latter, each PDBS needs to establish value correspondences between its local data and data on remote PDBSs for the purpose of data sharing. Such value correspondences constitute logical links that we call *acquaintances*. For example, we may consider a network of peer database systems of family doctors, hospitals, medical laboratories, and pharmacists that are willing to share information about treatments, medications, and test results of their patients. Consider a

situation where a patient has an accident in a city where he is currently visiting. He then visits a walk-in clinic. The doctor in the clinic needs to know the patient's previous medications and treatments from the patient's family physician. The associated doctor in the walk-in clinic establishes acquaintances with the patient's family doctor and pharmacist for sharing information about the patient. The doctor in the walk-in clinic then retrieves information from the acquainted peers, for instance family physicians and medical laboratories. In this perspective, we need particularly fine-grained data coordination, sharing and query processing strategies. Coordination rules need to be dynamic because peers join and leave the system freely. In such a dynamic environment, the existence of a global schema for the entire databases is not feasible [8]. We can consider another interesting domain from biological databases. In such systems, the two sources might store related, yet inherently different, information. One source may be storing information about genes, while another stores information about proteins. Genes and proteins are related since genes produce proteins, yet there is no way to map the schema for genes to that for proteins without altering one or the other. The authors in [25] propose an alternating way of mapping called *mapping tables*. Mapping tables are data-level mappings which list pairs of corresponding values between two sources. Our query translation mechanism uses such mappings to translate queries between peer databases.

In a peer-to-peer paradigm, mainly two basic services are offered. First, it offers to its peers the ability to share data with each other. Second, it offers the ability for peers to query each other's contents. Our objective is to investigate a query mechanism that can be used to share data through query translation. We can find similar issues in the context of traditional distributed [34], federated and multidatabase systems [35]. Although the P2P systems resemble to the distributed and multidatabase system in

terms of decentralization fashion, but solution offered there can not be directly applied to peer-to-peer data management systems [24]. This is mainly due to the following three features of the new paradigm: the lack of centralized control; the transience of the inter-peer connections; the limited cooperation among the peers.

Several previous approaches to the problem of query translation across heterogeneous sources exist, e.g. [21, 15, 6, 24, 5]. Our approach generalizes two of these previous approaches based on a restricted form of rule-based schema level mappings [15], and data instance level mappings [24].

The approach described in [15] introduced the idea of using human-specified rules to specify mappings of selection conditions of SQL queries; i.e. to specify how two selection conditions are related. For example, such a rule may specify that a condition $Airport = McDonald$ is to be mapped to the condition $City = Ottawa$. This codifies a portion of the domain semantics. The rule also gives a user-provided function for transforming the airport A to a city C. The whole rule is as follows:

$[Airport = A] \mapsto C = Location(A); emit : [City = C]$

This rule says that the condition $Airport = A$ maps to the condition $City = C$, whereby the value C is obtained by applying the user-provided function $Location(A)$ which finds the city where a given airport is located. Using these rules, the task in [15] is to design an algorithm for translating complex queries whose selection conditions may match one or more rules of the sort given above.

On the other hand, the approach presented in [24] relies on collections of correspondences between data values called *mapping tables* (More on these in Section 3.3 and in [25]). Here, query processing is viewed in the general context of data sharing; that is, a user on a peer poses a query which, from the user's point of view, is strictly local, but

the system propagates that query through translation to acquainted peers. The translation algorithm uses the mapping tables to map the posed query to the data values (also referred to as "vocabulary") of acquainted peers. Unlike the approach in [15], where the user is asked to "mind" her vocabulary, the whole approach in [24] is end-user-oriented so that the user does not mind her vocabulary, since the system takes care of bridging the heterogeneity gap at the level of the data instance.

1.2 Contributions

In this thesis, we make the following contributions.

- First, we present a query translation mechanism that does not use a restrictive global/mediated schema and that considers the heterogeneity and autonomy of peer databases. The algorithm translates arbitrary Select-Project-Join SQL queries in two phases. The first phase is performed in the local peer where the query is originated. The second phase is completed at the remote peer when the query originator can not fully translate the query.

- Second, We present a mapping stack consisting of four layers of mappings that each peer uses partially or fully in order to create semantic relationships with other peers. We present the semantic relationships formally in terms of correspondence assertions. We also introduce some generic query translation rules based on the mappings and show how these rules are used to translate queries on-the-fly.

- Third, we present a two phase algorithm that translates queries according to a collection of user-generated rules that combine a restricted form of syntactic schema

mappings with the data instance level mappings. We also offer the criteria of a sound translation of queries based on our solutions

- Finally, we implemented our query translation algorithm on top of the increasingly reliable P2P JXTA framework [7]. We ran the implementation on a small prototype P2P network of JXTA-based peers. We show measurements that indicate that the algorithm is efficient.

1.3 Outline

The remainder of the thesis is organized as follows. Chapter 2 offers an overview of the background and motivation of this thesis. We start with the basic concepts of peer-to-peer databases. In this chapter we also present related work. In Chapter 3, we discuss a usage scenario and address some concepts that are used in our query translation framework, for instance, mapping table semantics, peer-to-peer mapping semantics, correspondence assertions, and data and P2P network models. Chapter 4 introduces query translation rules and peer-to-peer query semantics. We also describe the query translation algorithm elaborately in this Chapter. Then Chapter 5 presents the architecture and experimental results to evaluate the efficiency of the framework. Finally, we conclude with Chapter 6, which offers a summary of the thesis and a discussion of future work.

Chapter 2

Background

Over the past few years P2P networks are becoming increasingly popular. In P2P environments, the most significant work has been focused on the area of file-sharing systems like Gnutella [41], Napster [18], Freenet [4], and Kazaa [12]. Recently, the database community has begun to exploit P2P paradigms for database applications. The paper [11] first discusses data management issues in P2P environments from a database research perspective. The paper addresses some challenging issues in P2P systems and proposes some initial ideas for data placement and query answering. Until now a lot of progress has been achieved on peer database systems issues such as data integration, placement, mapping, and query formulation [14, 20, 22, 24, 13]. Still there exists no complete peer database system. In this thesis, we are interested in query translation for data sharing in a peer database network. In the first and second parts of this chapter we briefly discuss how the peer-to-peer systems differ from the traditional de-centralized systems, for example, multidatabases and distributed systems. Then we discuss research issues regarding peer database systems. In the third part, we describe in detail existing peer database systems motivating the work presented in this thesis.

2.1 Multidatabase and Distributed Data Management Systems

In multidatabase systems, the de-centralized nature of the system is not transparent to the user: each user independently must be concerned about discovering sources, understanding schemas, integrating the sources, and creating a global schema. Meanwhile in peer-to-peer systems the integration of sources is performed in a peer-to-peer fashion. That is, sources are integrated in a pairwise mappings. Also in multidatabase system a rigid global schema construction is required, along with a priori knowledge of all the sources to be integrated. Therefore, it does not allow a graceful joining, or leaving, of sources.

In distributed database systems, there is a global schema that describes the information in the underlying sources, which are homogeneous. From the design perspective, this global schema is generally designed in a top-down fashion. That is we first design the global schema, and given a set of sources, we describe how the information is to be allocated among them. In such a system, it is assumed that sources are to be stable and unchanging throughout the system's lifetime.

Query processing techniques in peer database networks can be compared to similar techniques used in the traditional distributed and multi database systems which use global or mediated schema to allow viewing of all involved databases as one single database. In such traditional systems, which integrate the participating data sources, a user poses a global query to a mediator that retrieves data from the underlying sources to answer the query. There are three main steps in processing a global query [9]. Firstly, the global query is decomposed into a number of sub-queries. Secondly, each sub-query

is translated to a query for the corresponding local database system that manages a given local source and sent there for execution. Finally, the results returned from the local DBMSs are combined into the answer. On the contrary, in general, peer database systems have no such global schema and permit true autonomy to peers. Each node is responsible for the management of its own database and has no control over the other nodes in the system. Also the nodes can join and leave the system freely. In addition, queries are posed on local database and results are retrieved from the local database as well as from the peer database network. Queries are recursively propagated over the network. Nodes are connected with links in arbitrary fashion. So the topology of the P2P network is dynamic.

2.2 Research in Peer Database Systems

In this section we briefly state some of the progresses that have been achieved so far in peer database systems such as data coordination, mapping, and data sharing.

In [13], Arenas et al. present an architecture for peer data management systems. They propose a mechanism for coordinating peer databases using Event-Condition-Action (ECA) rules. The ECA rules are also known as triggers or alerters. ECA rules are used to monitor when an event occur in a peer. For instance, whenever an update occurs then specified actions are propagated and executed at remote peers. The ECA rules have condition that determine when the actions should be executed.

The paper [14] introduces a data model called the Local Relational Model (LRM) and it is designed to allow P2P data management systems to describe relationships between two peer databases. The acquaintances between two peers are based upon the definition of coordination formulas and domain relations within the system. The main goals of the

data model are to support semantic interoperability in the absence of a global schema.

Kantere et al. [22] describe techniques related to establishing and abolishing acquaintances of peers in P2P systems. They also present an approach for specifying data exchange policies on-the-fly based on constraints. When a peer joins the network, it needs to register to certain interest groups. The system assumes that a standard schema exists for each interest group and the schema is known to all members of the group. So the standard schema becomes the bottleneck of the system. The complexity to maintain the standard schema is high. Moreover, such a standard schema restricts changes to local database schema of peers.

An approach for data coordination which avoids the assumptions of a global schema is introduced in [19]. The authors of [19] introduce the notion of a group. They define it as a set of nodes, which are able to answer queries about a certain topic. Each group has a node called Group Manager (GM), which are in charge of the management of the metadata needed to run the group [19]. According to their proposal, each query must pass through the group manager. The paper does not mention how a node is chosen as a group manager.

2.3 Peer Database Systems

In this section we briefly discuss about some peer database systems and their query processing strategy which serve as the basis of our work.

2.3.1 PIAZZA

The Piazza system [30, 20] provides a framework that replaces the single logical schema of data integration systems with a small subsets set of mediators' schema that are interlinked to define semantic mappings between the peer schemas. Piazza proposes a language called Peer Programming Language (PPL) to specify schema mappings between peer databases; PPL combines the two dominant data integration formalisms found in the literature, namely local-as-view (LAV) and global-as-view (GAV). GAV is used to define relations of the mediator's schema over the relations in the sources. To define relations in the sources over the mediated schema PPL uses the LAV approach. All these are schema (as opposed to our instance) level mappings.

There are two types of mappings in Piazza. They are storage descriptions and peer mappings. The storage descriptions describe the mappings between peer relations and its stored relations. Peer mappings provide semantic relationships between the schema of different peers. In Piazza, a query is posed on peer schemas and reformulated in terms of the stored relations. The query reformulation algorithm takes as input a set of peer mappings, storage descriptions, and a query Q. The algorithm assumes that all the peer mappings are available at a single location, and hence all the reformulation is done in a single place. The algorithm first constructs a simple rule prolog-like goal tree, where goal nodes are labeled with atoms of the peer relations, and rule nodes are labeled with peer mappings. The query reformulation starts by expanding each query sub-goal according to the relevant definitional peer mappings in the peer data management system. When none of the leaves of the tree can be expanded any further, storage descriptions are used to reformulate the query in terms of stored relations

Peer	Names	Keywords
P1	Kinases	Protein, Human
	SeqID	Key,Identifier,ID
	length	Length
	ProteinSeq	Sequence, protein Sequence
P2	Protein	Protein, annexin, zebrafish
	SeqNo	Number, identifier
	Len	Length
	Sequence	sequence

Figure 2.1: Keywords for peer relation and attribute names

2.3.2 Peer DB

Peer DB [27] consists of a number of autonomous peers, each of which consists of a relational DBMS, an agent system DBAgent, and a cache manager. PeerDB uses an information retrieval approach to establish schema mappings where each relation and attributes of relations in the peers databases are tagged with descriptive keywords. The main feature of PeerDB is the absence of a predetermined and uniform schema that nodes share. Query processing in PeerDB is performed in two phases using agents. In the first phase, agents find relevant peers matching keywords; in the second phase, they perform query answering. Consider an example taken from [27] in Figure 2.1 where two peers $P1$ and $P2$ have the following relations

Kinases(SeqID, length, proteinSeq)

Protein(SeqNo, len, sequence).

Figure 2.1 shows the keywords defined for these relations. Assume that the user at peer P1 issues the following query to look for kinases sequences that are longer than 30 base pairs:

select Seqid, proteinSeq
from Kinases
where length > 30

Using the common keywords between the P1 and P2 schema elements, PeerDB translates query Q to the following query over the schema of peer P2

select SeqNo, sequence
from Protein
where len ≥ 30

The keyword-matching strategy in PeerDB may give irrelevant query reformulation because a keyword of a relation may match syntactically with keywords of attributes or relations of other peers. The user must decide which queries are to be executed. So, in PeerDB, continuous user involvements are required before the user fetches required data from peers. In our approach, on the contrary, once acquaintances are in place, the user need not worry about them at query time.

2.3.3 Hyperion

Hyperion addresses heterogeneity by means of mapping expressions and mapping tables [25]. A typical Hyperion peer is shown in Figure 2.2. Mapping tables are used when the data vocabulary of acquainted peers are different. A mapping table contains a set of data associations between values in two peer databases. Consider an example from the domain of airline-ticket reservation systems taken from [24] to illustrate the Hyperion

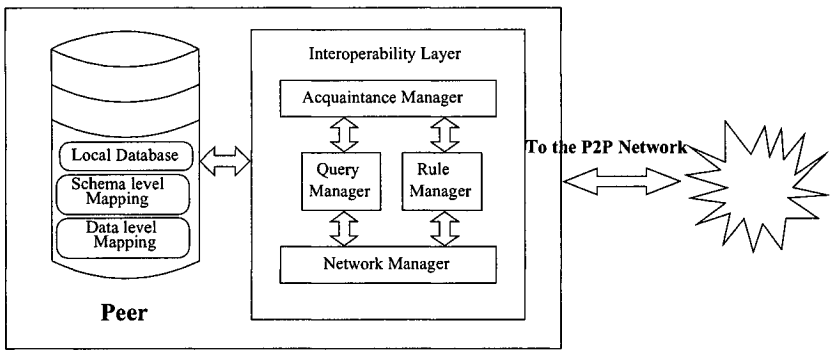


Figure 2.2: Hyperion architecture

Fno	Date	Time	Dest
LH401	11/05	10:00	S.F.
LH541	11/15	00:30	N.Y.
LH690	11/21	12:00	L.A.
LH691	11/22	12:00	L.A.

(a) Table LH

Flight	Dt	Tm	To
UA120	11/15	10:30	JFK
UA134	11/15	11:00	LAX
UA135	11/21	12:00	ONT
UA141	11/22	07:00	ORD

(b) Table UA

Fno		Flight
LH541		UA120
LH690		UA135

(c) Mapping table fno2flight

Date		Dt
X		X

(d) Mapping table date2dt

Dest		To
N.Y.		JFK
N.Y.		LGA
N.Y.		EWR
L.A.		LAX
L.A.		ONT

(e) Mapping table dest2to

Figure 2.3: Database instances and mapping tables

peer to peer database management. Suppose that two airlines Lufthansa and United have the following schemas.

LH(fno, date, time, dest)

UA(flight, dt, tm, to)

Figure 2.3 shows database instances and mapping tables. From Figure 2.3, we notice that the flight information and related airport data are represented differently in two

peers. Mapping table $fno2flight$ associates flight number of LH to flight number UA. Also the mapping table $dest2to$ associates city names in the LH relation to airport codes in the UA relation. Assume that a user poses a query to retrieve information for the LH flights to Los Angels. Then, a query such as the following one may be used:

*select **
from LH
where dest = "L.A."

The above query can be translated for UA flights using mapping tables of Figure 2.3 as follows:

*select **
from UA
where (to ="LAX" OR to = "ONT")

To translate a query, the algorithm represents the query as a $T-query$. A $T-query$ is a tabular representation of the query. The paper [24] also presents an algorithm that is used to test sound translation of queries. We now briefly describe the query translation algorithm used in [24] with respect to a mapping table m.

Algorithm

Let P and P' be two peers that expose attributes U and U', q is a query over peer P, and m is a mapping table between the set of attributes U and U'. Then, convert q to q'

with respect to the mapping table m as follows:

Step1: Convert query q to its disjunctive normal form q_{DNF}

Step2. Convert query q_{DNF} to its corresponding $T - query\ q_T = T$

Step3. Compute $T - query\ q'_T = \Pi_{U'}(T \bowtie m)$

Step4. Convert $T - query\ q'_T$ to q'

Step5. Output the query q_c.

The algorithm supports select-project-join queries, where the selection formula is of the simple form $A = x$. Where A is an attribute and x is a constant value in a predicate term. The algorithm first transforms the query into disjunctive normal form (DNF). Therefore the transformed query may contain repeating occurrences of the same constraint in many disjuncts. The algorithm does not consider the inter-dependencies between constraints. Therefore, the DNF might contain number of repeating terms in the query. It does not describe how to translate queries where the mapping table does not fit like range queries, queries with negation and group by queries. The algorithm uses mapping tables to deal with data level heterogeneity between peer databases during query translation for the acquainted peers. Our algorithm extends the above algorithm by using mapping tables and rules to deal with heterogeneity both at the data level and at the schema level. We also extend the semantics of sound translation of query based on multiple mapping tables.

2.4 Summary

This chapter offered an overview of the work related to this thesis. We first described the traditional distributed and multidatabase systems and how they differ from peer

database systems. After that we described some research issues in peer database systems. We also described some work related to query processing systems and highlighted their similarities and differences with our approach. In our solution we address an expressive query translation framework considering the heterogeneity of peers both at the schema level and the instance level. Indeed our queries are arbitrary SPJ queries

Chapter 3

Motivating Example, Preliminaries and Definitions

In this chapter we describe a motivating example and introduce some notions and concepts that will be used throughout this thesis.

3.1 Motivating Example

We use a motivating example from the Health Care domain where hospitals, family doctors and pharmacies, all share information about patients. This information includes health histories, medications, and exams. Figure 3.1 depicts the schemas used for this domain. The Family Doctor peer has a unique Ontario Health Insurance Patient (OHIP) number assigned to each patient and record is kept of name, illness, date of visit and medication of patients. The relation *MedicalExam* stores the information about the patients' medical examinations. The Hospital peer has relations *Patients* and *Labtest*. Table 3.2 shows partial instances of both peer databases.

Family Doctor Database
MedicalExam(OHIP, TestName, Result)
Patient (OHIP, Lname, Fname, Illness)
Hospital Database
Patients (PATID, Primary_Desc, Name)
Labtest (PATID, Test, TestResult, Cost)

Figure 3.1: Database schemas

OHIP	Lname	Fname	Illness	Date
233GA	Lucas	Andrew	Headache	Jan/04
501NE	Davidson	Ley	Allergy	Jan/04

(a) Patient table instance

OHIP	TestName	Result
233GA	whitebloodcount	9755 c/mcL
501NE	homoglobin	14.6 g/dL

(b) MedicalExam table instance

PATID	Name	Primary_Illness
243	Davidson,ley	StomachPain
359	Lucas, Andrew	Heart Problem

(c) Patients

TestId	Test	Result	PATID
4520	C0427512	633 c/mcL	359
4521	C0518015	12.5 g/dL	243

(d) LabTest table instances of hospital database

Figure 3.2: Database instances

Assume that a patient has been admitted to a hospital. The doctor in the hospital needs the medical examination that the patient has gone through on admission as well as the patient's recent test reports from the patient's family doctor. To get the medical examination, the doctor in the hospital may pose the following query against the local Hospital peer database:

Q1:

*select **

from LabTest

where PATID="243" AND Test="C0518015"

To get the test reports, the very same doctor needs to have the following query posed against the remote peer database of the patient's family doctor:

Q2:

*select **

from MedicalExam

where OHIP="501NE" AND TestName="homoglobin"

Normally, due to the autonomy of peer databases, the doctor in the hospital should be able to pose the later query in terms of the schema and instance data values of his local peer database. However, Figure 3.2 shows that there are differences in the way patients' information is represented both at the schema and at the data instance level in the two peer databases. This representational discrepancy raises the need for some way of mapping one representation to the other, both at the schema and at the data instance levels. Such a mapping will permit interpretability between these heterogeneous scenarios for query translation. We will use mapping tables [25] to resolve heterogeneity at the data level and syntactic mappings of schema element for schema level heterogeneity during query translation. Based on these mappings, we develop several generic query translation rules that translate a query q posed on a peer P to a set of queries $Q' = \{q_1, \cdots, q_n\}$, where each one of the q_i's is the original query q translated to the vocabulary of a given acquainted peer P_i.

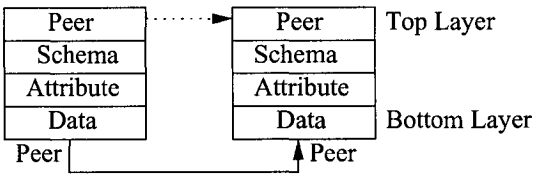


Figure 3.3: Peer to peer mapping stack

3.2 Peer-to-Peer Mappings

In what follows, we assume that sources are fully autonomous and may partially or fully share information at different levels. We consider pairwise mappings between peers with their shared schema elements. A mapping consists of four layers. Figure 3.3 shows the four layers of mappings. The top layer is the peer-to-peer mapping, which defines the acquaintance between two peers. Two peers have an acquaintance (or, equivalently, are acquainted) iff they are linked by a set of mapping tables. Schema elements are exchanged between peers as part of the acquaintance protocol. The second layer is the schema level mapping which helps to overcome schema level heterogeneity. The next layer is the attribute level mapping that is used to overcome heterogeneity at the attribute level. The last one is called instance/data level mapping. We use this layer if there are any differences in data vocabularies between attributes of two peer relations. We use the concept of mapping tables to create the data layer mapping.

There may be four different types of mappings between schema elements. They are as follows:

1) One to One: In this mapping type, a schema element such as an attribute/relation of one peer is mapped to exactly one attribute/relation of another peer. For example the attribute $OHIP$ of relation *Patient* in the Family Doctor peer database is mapped to attribute $PATID$ of relation *Patients* in the Hospital peer database.

2) One to Many: Here, a schema element such as an attribute/relation of one peer is mapped to more than one attribute/relation of another peer. For example the attribute Name of relation *Patients* is mapped to attributes *Lname* and *Fname* of relation *Patient*. Moreover, a set of attributes of one relation in one peer may be mapped to more than one relation in another peer. To see this, consider the following scenario.

Peer Hospital:

$Patient(OHIP, Lname, Fname, Illness, Address, Telno)$

Peer Labtest:

$Patients(PATID, Name, PrimaryIllness)$,

$PatientInfo(PATID, Street, City, Pcode, Telephone)$

From the above scenario, whenever we pose a query against the Hospital peer database to retrieve a patient's name and address given a particular patient ID, we need only one relation. However, to retrieve information from the Labtest peer, we need to join two relations.

3) Many to One: This mapping is the reverse of one to many. For example, *Lname* and *Fname* of the Family Doctor peer is mapped to attribute *Name* in the Hospital peer. Similar mappings between relations may exist.

4)Many to Many: This is established when more than one attributes/relations in one peer correspondent to more than one attributes/relations in a remote peer.

In this thesis we mainly focus on the query translation process. We assume that such mappings are generated using some existing schema mapping techniques. Here, we address query translation rules to be obtained from the above syntactic and data level mappings. We also present a strategy for finding relations in the translated query if no mapping exists between relation names. We shall elaborate on this issue in Section 4.5.

OHIP	PATID
501NE	243
233GA	388

(a) Mapping table OHIP2PATID

TestName	Test
homoglobin	C0518015
whitebloodcount	C0427512

(b) Mapping table TestName2Test

Figure 3.4: Example of mapping tables

3.3 Mapping Tables

We need to focus on data values and value mappings when there is little or no agreement on the logical design of data and data vocabularies are different in two different sources. Through value mappings, we can still share and query specific data of interests. Authors in [25] introduce a semantic called *mapping table* to represent such value mappings that store the correspondence between values. They use mapping table for data integration and exchange between heterogeneous data sources. Mapping tables are data-level mappings that list pairs of corresponding values between two sources. Simply we can say that mapping tables are binary tables containing pairs of corresponding identifiers from two different sources. Intuitively, a mapping table is a relation over the attributes $X \cup Y$, where X and Y are non-empty sets of attributes from two peers. For example, Figure 3.4 shows a mapping table representing a mapping from the set of attributes $X = \{PAITID\}$ to the set of attributes $Y = \{OHIP\}$. The same table shows another mapping table that relates $MedicalExam.TestName$ and $LabTest.Test$. In general, we may need to map values containing multiple attributes called $n - array$ mapping tables. For example, one peer may store geographic locations using postal code and by pairs of area code and city name in another peer.

Mapping tables represent expert knowledge and are typically created by domain specialists. Currently the creation of mapping tables is a time-consuming and manual process

performed by a set of expert curators. Still there is no complete automated tool to facilitate the creation, maintenance and management of these tables [25]. However, the paper [25] introduces two mechanisms to maintain mapping tables that reduce considerable effort of curators.

Infer new mapping tables : This problem concerns to find the set of all mapping tables that are valid and available over a network of peers. For this purpose we must combine the knowledge from these mapping tables in order to infer additional mappings-mappings that are not explicitly represented in any peer. The paper proposes such an algorithm. There are two assumptions in the algorithm: the path must be linear and the length of path must be small. The algorithm also has not considered the dynamic settings of P2P systems.

Determine consistency of mapping tables : Due to evolving architecture of P2P network, it is natural that curators need to delete, edit, copy, or merge mapping tables. So updating mapping table is a crucial task and it needs to ensure that the changes of mappings of one table do not invalidate those expressed by another. The paper also proposes a static algorithm to help curators determine whether, or not, a set of mapping tables is consistent.

However, both of these mechanisms play an important role in helping a curator understand and correctly specify the semantics of a set of mapping tables.

We can treat mapping tables as constraints in exchanging information between peer databases [25]. In the thesis, we use mapping tables in this sense. We assume a *closed world* semantics for mapping tables. A mapping table may contain variables. Therefore it requires a valuation function to map between data in a mapping table.

Definition 1 *[25] A* valuation ρ *over a mapping table* m *is a function that maps each*

keyword		kw
OPH		APH
OPH		AARE
NGF receptor		p75 ICD

(a) keyword2kw

year		py
X		X

(b) year2py

Figure 3.5: Mapping tables

constant in m to itself and each variable v of m to a value in the intersection of the domains of the attributes where v appears. Furthermore, if v appears in an expression of the form v -S, then $\rho(v) \notin S$.

Consider a mapping table m from X to Y, whose mappings might include variables, and a value $x \in dom(X)$. To determine the set of Y-values with which the value x can be associated, the following definition uses the notion of valuation to help us determine this set.

Definition 2 *[25] Let X and Y be nonempty disjoints sets of attributes and let m be a mapping table from X to Y. We define* $Y_m(x)$, *where* $x \in dom(X)$, *as follows:* $Y_m(x) = \{y | \exists\ t \in m$ *and there exists valuation* ρ *over m such that* $\rho(t[X]) = x$ *and* $\rho(t[Y]) = y\}$.

Example 1 *Consider the domain of biological databases, namely MedLine and PubMed. let us assume that we have mapping tables as shown in Figure 3.5 taken from [25]. We can see that* $kw_{keyword2kw}(OPH) = (APH, AARE)$. *Furthermore, if we consider the mapping table year2py from Figure 3.5, then notice that* $py_{year2py}(2003) = 2003$.

3.4 Data and P2P Network Model

A *database schema* is any nonempty, finite set $DB[W] = \{R_1[U_1], \ldots, R_n[U_n]\}$ of relations, where U_i is a subset W, the set all available attributes, and R_i is a relation name;

$R_i[U_i]$ denotes a relation R_i over a set U_i of attributes. Given a relation R, and a query q, we will use the notation $att(R)$ and $att(q)$ to denote the set of attributes mentioned in R and q respectively. *Instances* of a relation schema $R_i[U_i]$ and a relational database $DB[W]$ are defined in the usual way.

Now we formally introduce the notion of a network of PDBSs.

Definition 3 (Network of Peer Database Systems) *A network of PDBSs is a pair $(\mathcal{N}, \mathcal{M})$. Here, $\mathcal{N} = \{(\mathcal{P}, \mathcal{L})\}$ is an undirected graph, where $\mathcal{P} = \{P_1, \cdots, P_n\}$ is a set of peers, $\mathcal{L} = \{(P_i, P_j) | P_i, P_j \in \mathcal{P}\}$ is a set of acquaintances. Each peer P_i is associated with an instantiated relational database – a peer database – with schema $DB_i[W_i]$, and each acquaintance (i, j) is associated with a set $\mathcal{M}_{ij} \in \mathcal{M}$ of mapping tables.*

We will interchangeably call networks of PDBSs "P2P networks". To motivate our thesis, we used an example by assuming that, though the peer schemas are heterogeneous, each peer makes all its schema visible to acquainted peers. However, this assumption is not realistic and we relax it in this section. We now assume that peers make only a subset of its schema visible to its peers. That is, we assume that each peer P_i exports a (possibly empty) subset $V_i \subseteq DB_i[W_i]$ of schema elements (i.e. attributes or relations) called *export schema* of P_i. For simplicity, we assume that $V_i \cap V_j = \emptyset$, for all $i \neq j$.

3.5 Correspondence Assertion (CA)

We need to formally characterize the relationship between attributes/relations of acquainted peer schemas. We capture this relationship in the notion *correspondence assertion* (CA). Informally, a CA states that the data in one peer is represented in another peer differently with respect to the used vocabulary, format or structure. The CAs are

generated from mapping tables in an obvious way: whenever there is a mapping table m that maps values of attribute A_1 to values of attribute A_2, then we establish a CA between A_1 and A_2. We say that A_1 and A_2 are related with respect to m. This generation occurs at acquaintance time.

Definition 4 *Suppose that peers $P1$ and $P2$ have exported schemas V_{P1} and V'_{P2}, respectively. Let $\mathcal{M}$ be the set of mapping tables between $P1$ and $P2$. Moreover suppose that $\epsilon_1 \subset V$ and $\epsilon_2 \subset V'$. Then there is a correspondence assertion $\mathcal{CA}$ between ε_1 and ε_2 if there is a mapping table $m \in \mathcal{M}$ such that ϵ_1 and ϵ_2 are related with respect to m. We use the following syntax to represent correspondence assertions:*

$\mathcal{CA} : \epsilon_1 \longrightarrow \epsilon_2 : m$

Example 2 *Consider the instances in Figure 3.2 and the mapping tables in Figure 3.4. Suppose the Family Doctor peer has the following export schema:*

$V = \{OHIP, Lname, Fnane, TestName, Result\}$

Also suppose the Hospital peer has the following export schema:

$V = \{patid, name, test, result\}$

Therefore we create the following correspondence assertions.

CA1: OHIP⟶PATID

CA2: TestName⟶Test

3.6 Summary

In this Chapter we gave a motivating example that will be used throughout the thesis to illustrate our query translation framework. We also briefly described the mapping table

semantics and found that by using mapping tables we are able to associate seemingly unconnected databases. Moreover we address a peer-to-peer mapping scenario and formally characterize the mappings in terms of correspondence assertions. Later we show how to create query translation rules from correspondence assertions which serve as the basis of the query translation framework.

Chapter 4

Querying Peer-to-Peer Databases

4.1 Introduction

In the previous chapter, we introduced some basic concepts that are the basis of our query translation algorithm. In this chapter we introduce our generic translation rules and algorithm. The translation rules are based on mapping tables and correspondence assertions.

In our framework the input to the algorithm is an SPJ SQL query whose selection formula may contain arbitrary conjunction and disjunction of literals. That is, queries may mention the negation operator in their selection formula. The query translation process is completed in two phases. The first phase is completed in the local peer where the query is originated and the second phase is performed at the remote acquainted peers. The second phase is completed when the received query is partially translated. We shall describe the translation issues elaborately in Section 4.5. Before describing the framework we first discuss the query syntax and semantics.

4.2 Query Syntax and Semantics

We start by specifying the syntax of the queries that are supported in our framework. Next we describe the query execution semantics.

4.2.1 Query Language

We already mentioned our framework supports Selection-Projection-Join (SPJ) queries whose selection formula may contain arbitrary conjunction and disjunction of literals. That is, queries may mention the negation operator in their selection condition. The atoms of the selection formula are of the form (A *op* B) and (A *op* a), where A and B are attribute names, a is a constant, and *op* is a comparison operator $<, >, =, \leq, \geq$, or $\neq$. Without loss of generality, we assume that a query q, written in our algebra, will be of the following form $q = \Pi_{A_1,\cdots,A_k}(\sigma_F(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n))$ where $R_i[U_i](1 \leq i \leq n)$ are relation names, each $A_j \in U_1 \cup U_2 \cup \cdots \cup U_n (1 \leq j \leq k)$, and F is a selection formula. In this thesis we use SQL language to represent the user queries.

4.2.2 Query Execution Semantics

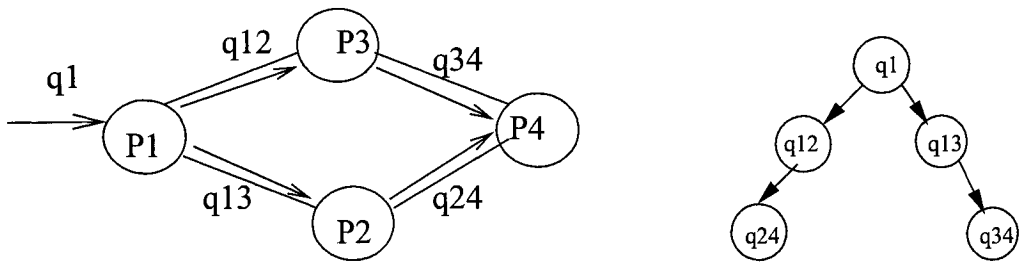
There are two types of queries in PDBs, namely local and global queries [13]. A local query is defined in terms of a local peer database schema, while a global query is defined over the peer database schema of the P2P network that are directly or indirectly acquainted with the peer where the global query is initiated. Semantically, a global query is a set of queries translated from the local query, all destined to acquainted peers. A global query is generated when a user wants results from all reachable (directly or indirectly) acquainted peers in the P2P network.

This informal semantics of queries can be formalized as follows (slightly modifying the semantics given in [24]). Suppose a network $\mathfrak{N} = (\mathcal{N}, \mathcal{M})$ of PDBSs, where $\mathcal{N} = (\mathcal{P}, \mathcal{L})$ and $\mathcal{P} = \{P_1, \cdots, P_n\}$. A local query q in a peer $P_i \in \mathcal{P}$ is defined over (a subset of) the schema $DB_i[W_i]$ of P. The answer to q is a relation over the instance db_i of $DB_i[W_i]$. A global query $q_{\mathfrak{N}}$ over the P2P network $\mathfrak{N}$ is a set $\{q_1, \cdots, q_k\}$ $(1 \leq k \leq n)$, where each query q_i $(1 \leq i \leq k)$ is a *component query* defined over the schema of a reachable peer. The intuition behind this definition is the following: each component query q_i is a user defined query that has been forwarded to all reachable peers via translation through translation rules. The answer to the global query $q_{\mathfrak{N}}$ is the set $\{q_1(db_{i_1}), \cdots, q_k(db_{i_k})\}$, where $q_j(db_{i_j})$ $(1 \leq j \leq k)$ is a relation over the instance db_{i_j} of peer P_j.

We can depict the relationship between the component queries of a global query using a *query dependency graph*. The nodes in this graph represents the component queries and an edge from query q_i to q_j represents the translation and propagation of query q_i to query q_j from peer P_i to P_j.

Example 3 *Consider the P2P network of figure 4.1(a). The network consists of four peers* $P = (P1, P2, P3, P4)$. *The undirected edges between peers represents the acquaintances between them and the directed edges depict query propagations. We assume that the acquaintances are bi-directional. Let us assume that a query* q_1 *is originated in peer* $P1$. *After translation the peer* $P1$ *sends the translated queries* q_{12} *and* q_{13} *to peer* $P1$ *and* $P2$ *respectively. Peers P2 and P3 then translate and forward the query to peer P4. So the global query* $q_{\mathfrak{N}} = \{q_1, q_{12}, q_{13}, q_{24}, q_{34}\}$. *The corresponding query dependency graph is shown in Figure 4.1(b)*

For query propagation we use a $translation-and-forward$ mechanism. When a user poses a query on a peer then the peer first translates the query for all acquainted peers



(a) A P2P network (b) Query dependency graph

Figure 4.1: A P2P network and corresponding query dependency graph.

and sends the translated queries to its acquaintances. The translation may be partial or complete which, depends on how the mapping tables and correspondence assertions are established or defined between acquainted peers. We say a query is partially translated when the peer does not find a data association in a mapping table or correspondence association to translate a query predicate term. Before sending the translated query the peer first adds a tag, we say global identification (GID) of the query as well as the peer identification (PID). When a remote peer receives the query, the peer either translates and/or forwards the query adding its PID with the query. This $translation - and - forward$ process continues until no further propagation is possible. The global identification (GID) is used to avoid duplicate translation of a query in a peer because a peer may receive queries in different form but with same GID from multiple acquaintances. Therefore, if a peer receives a query with the same GID as a query already seen, then the new query is rejected. The path tag makes this possible and thus helps avoid cycles. So the path tag is used to trace from which peer the query has been originated and the list of peers the query has been visited. Therefore, looking at the path tag, a peer knows whom to forward the query. We also propose another strategy that reduces the number of global queries in the P2P network. We will come back to this issue of the reduction of the

number of generated queries in Section 5.3.

4.3 Query Translation Rules

Query translation is the problem of transforming a query q over the schema of a local peer P to a query q' over the schema of an acquainted peer P'. This transformation is based on a set $\mathcal{R}$ of translation rules. Translation rules $\mathcal{R}$, to be introduced below, are statements as to how the vocabulary of a query over a source peer is to relate to the vocabulary of an acquainted target peer. The query vocabulary refers to three types of information, namely the set of *attributes* that represent the answer or result of the query, the *data values* mentioned in query search conditions, and the *relation names* mentioned in the query.

To compute an answer of a global query, there are two main tasks to be carried out. The first main task is to translate the given query, initiated at a particular peer, to a subset of the acquainted peers according to the translation rules and mapping tables. The second main task is to propagate the queries according to the acquaintance graph.

Definition 5 (Query Translation System) *A query translation system T is a triple $(q, Q, \mathcal{R})$, where q is the original query, Q is a set of target queries, and $\mathcal{R}$ is set of translation rules that translates the query q to the set of target queries Q. Each rule $r \in \mathcal{R}$ is of the form*

$$\sigma_c : \sigma_{c'} : te,$$

where σ_c and $\sigma_{c'}$ denotes the predicate atoms over the schema of the local and the target peer, respectively, and te is a translation expression stating how σ_c and $\sigma_{c'}$ are related.

In the sequel, we address rules to resolve the problem of heterogeneity among peers

during query translation, both at the data and at the schema level. We define four classes of translation rules, namely Merge (M), Separation (S), Data Association (DA), and Data Conversion (DC). Each one of the following sections describes each one of these query translation rules and shows how these rules resolve heterogeneity on the fly during query translation among peers.

4.3.1 Merge Rules (M)

This class of rule resolves differences in format for the same data value. Therefore, we need a mechanism to translate the data format in the selection formula of the source query to the format of the formula that represents the target selection formula. We represent a merge rule as follows:

$$\sigma_{\bigwedge A_i=x_i} : \sigma_{B=y} : y = \Pi_B(R' \bowtie T_{\bigwedge A_i=x_i}), \tag{4.1}$$

where $R' = f_M(R(A_1, \cdots, A_n), B)$; $\sigma_{\wedge A_i=x_i}$ is a pattern of a selection formula in the source query and $\sigma_{B_i=y}$ is the translated selection formula for the target query. The value of y for attribute B is determined by the formula defined in the translation expression of the rule. The semantics of formula above is as follows:

- $f_M(R(A_1, \cdots, A_n), B)$ is a function that creates a temporary relation R' from relation R with attributes $A_1, \cdots, A_n$, and B; $A_1, \cdots, A_n$ are mentioned in $\sigma_{\wedge A_i=x_i}$ and B is the target attribute. Values of attribute B are generated with a user-provided function f that is applied on attributes $A_1, A_2, \cdots, A_n$.
- $T_{\bigwedge A_i = x_i}$ is a tabular representation of the term $\bigwedge A_i = x_i$.

Example 4 *Consider the following query that retrieves all the information about a patient named "Lucas Andrew" from the Family Doctor peer:*

Q3:

*select **

from patient

where lname= "Lucas" AND Fname="Andrew"

The corresponding merge rule is:

$$\sigma_{AND(Lname="Lucas",Fname="Andrew")} : \sigma_{Name="Lucas,Andrew"} :$$

$$"Lucas, Andrew" = \Pi_{Name}(Patient' \bowtie T_{AND(Lname="Lucas",Fname="Andrew")}),$$

where $Patient'$ is $f_M(Patient(Lname, Fname), Name)$, with f_M defined as $Lname+','+Fname$ (The '+' denotes the concatenation operation).

The complete translation, with respect to the merge rule above is:

Q4:

*select **

from patients

where name = "Lucas, Andrew"

There are few observations to make here. Our goal is to find the value y for the attribute *Name* according to the merge rule. To obtain the value y, we first create an extended relation *Patient'* from relation *Patient* using the formula M given in the merge rule. An instance of this extended relation is shown below:

Lname	Fname	Name
Lucas	Andrew	Lucas,Andrew
Davidson	Ley	Davidson,Ley

This instance is obtained with the function f defined as *(Lname+','+Fname)*. We then represent the predicate *name* = "*Lucas, Andrew*" into tabular form as follows:

Lname	Fname
Lucas	Andrew
Davidson	Ley

Then, after joining the above two tables and performing the projection over *Name*, we obtain the table below:

Name
Lucas,Andrew

Finally, we convert the above tabular representation into the predicate *Name= "Lucas, Andrew"*, which is the semantic translation of the predicate *Lname="Lucas" AND Fname="Andrew"*.

4.3.2 Separation Rule (S)

The separation rule is the reverse of the merge rule. The formal representation of the separation rule is:

$$\sigma_{A=x} : \sigma_{\bigwedge B_i=y_i} : y_i = \Pi_{B_i}(R' \bowtie T_{A=x}), \tag{4.2}$$

where $R' = f_S(R(A), (B_1, \cdots, B_n))$; $\sigma_{A=x}$ is a pattern of a selection formula and $\sigma_{\bigwedge B_i=y_i}$ is the translated selection formula. The value of $y_i's$ for attribute $B_i's$ are determined by the formula defined in the translation expression of the rule. The semantics of formula

above is as follows:

• $f_S(R(A), (B_1, \cdots, B_n))$ is a function that creates a temporary relation R' from relation R with attributes A *and* $B_1, \cdots, B_n$; A is mentioned in $\sigma_{\wedge A = x}$ and $B_i's$ are the target attributes for the translated query. Values of attribute $B_i's$ are generated with a user-provided function f that is applied on attributes A.

• $T_{A=x}$ is a tabular representation of the term $A = x$.

Example 5 *Consider a query that retrieves all the information of a patient named "Lucas Andrew" from the relation Patients.*

Q5:

*select **

from patients

where name = "Lucas, Andrew"

The corresponding separation rule to translate above query is:

$\sigma_{Name=("LucasAndrew")} : \sigma_{AND(Lname="Lucas",Fname="Andrew")} :$

$"Lucas" = \Pi_{Fname}(Patients' \bowtie T_{(Name="LucasAndrew")})$,

$"Andrews" = \Pi_{Lname}(Patients' \bowtie= T_{(Name="LucasAndrew")})$,

where $Patients'$ *is* $f_S(Patients(Name), (Lname, Fname))$ *is a split operation that will break down the attribute* $Name$ *into first and last name.*

After applying the rule 4.3.2 *we get the following translated query:*

Q6:

*select **

from patient

where lname= "Lucas" AND Fname="Andrew"

4.3.3 Data Association Rule (DA)

Our third query translation rule translates queries based on mapping tables. This rule deals with data level heterogeneity. When a query mentions a data value that differs from data values used in an acquainted peer, then we need to translate the predicate term in such a way that other peers are able to decipher it. We use mapping tables to translate this type of predicate term. The formal representation of the rule is:

$$\sigma_{\bigwedge A_i=x_i} : \sigma_{\bigwedge \bigvee B_j=y_j} : y_j = \Pi_B(m(A,B) \bowtie T_{\wedge A_i=x_i}), \tag{4.3}$$

where $\sigma_{\wedge A_i=x_i}$ is a pattern of a selection formula in a local query q and $\sigma_{\vee B_j=y_j}$ is the translated selection formula for target queries, and A and B are sets of attributes. The value of y_j for attribute B_j is determined by the formula $\Pi_B(m(A,B) \bowtie T_{\wedge A_i=x_i})$. Here, attributes A_i are those mentioned in $\sigma_{\bigwedge A_i=x_i}$.

Example 6 *Consider a query that retrieves information from the MedicalExam relation of a patient with OHIP = "5017NE"*

Q5:

*select **

from MedicalExam

where ohip="501NE"

Suppose we have the following mapping table in place between the Family Doctor and the Hospital peer databases:

OHIP		*PATID*
501NE		*243*
233GA		*388*

Then query Q5 is rewritten for the Hospital database schema as follows:

Q6:

*select **

from LabTest

where PATID="243"

The translation rule for the above query is:

$$\sigma_{OHIP=501NE} : \sigma_{\vee PATID="243"} : PATID = \Pi_{PATID}(Ohip2Patid(OHIP, PATID) \bowtie T_{OHIP="501NE"})$$

Here $T_{OHIP="501"}$ is the following tabular representation of the predicate OHIP = "501NE":

OHIP
501NE

After joining the above table with the corresponding mapping table and processing the projection over the attribute PATID, we obtain the following table:

PATID
243

From the above table, we find the transformed predicate as PATID ="243"

4.3.4 Data Conversion (DC)

This rule is used for translating data from one domain to another. For example, consider an attribute *height − in − inches* from one database and an attribute *height − in − centimeters* from another. The value correspondence of these two attributes can be

constructed by defining a conversion function f. Formally we represent this rule as follows:

$$\sigma_{A\ op\ x} : \sigma_{\vee B\ op\ y} : y = \Pi_B(R' \bowtie T_{A\ op\ x}), \tag{4.4}$$

where $R' = f_{DC}(R(A), B)$; $\sigma_{A\ op\ x}$ is a pattern of a selection formula in a local query q and $\sigma_{B\ op\ y}$ is the translated selection formula for the target query; f_{DC} is a user-defined function for data conversion; and $T_{A\ op\ x}$ is the tabular form of $A\ op\ x$. If the domain of attribute A and B are same, then function f_{DC} is called an identity function.

4.4 Sound Translation of Queries

In this section we describe the soundness (correctness) of query translation. Soundness ensures that the translated query retrieves correct data from acquainted peers which are relevant to the result of the original query. Consider two peers $P1$ and $P2$ with export schemas $V_1[U_1] \subseteq DB_1[W_1]$ and $V_2 \subseteq DB_2[W_2]$, respectively. Assume that a query translation rule r and a mapping table m exist along with correspondence assertions between the peers. Suppose that a query q_1 is posed over V_1 and that q_2 is the translation of q_1 over V_2. We use the notation $q_1 \longmapsto q_2$ to state that q_2 results from the translation of query q_1. Intuitively, ensuring a correct translation means that the translation should be such that q_2 retrieves from $P2$ only the data that are related to those that could be retrieved from query q_1 in peer $P1$. It is important to establish such a notion of correctness. Here, we extend the definition of correctness of query translation with respect to mapping tables given in [24]. Recall that we treat queries with arbitrary selection formulas. We now define translation soundness with respect to mapping tables.

Definition 6 *[Soundness w.r.t to Mapping Tables] Let q_1, q_2 be queries over peer P_1*

and P_2, respectively; Let $q_1 = \sigma_E(R_1 \bowtie .... \bowtie R_k)$, where E is a selection formula and $R_1, \ldots, R_k$ are relations in P_1. Then q_2 is a sound translation of q_1 with respect to a set $\mathcal{M} = \{m_1(X_1, Y_1), \cdots, m_k(X_k, Y_k)\}$ of mapping tables, denoted by $q_1 \overset{\mathcal{M}}{\longrightarrow} q_2$, where $X_i, i = 1..k$, and $att(q_1)$ are subsets of $\bigcup_{i=1}^{k} X_i$, if for every relation instance r_2 of P_2 and $t_2 \in q_2(r_2)$, there exists a valuation ρ of $\mathcal{M}$, and a tuple $t \in \sigma_E(\rho(\mathcal{M})), i = 1 \ldots k$ such that $\pi_{att(q_2)}(t) = t_2$.

Example 7 *Consider the following query*

Q7:

*select * from MedicalExam*

where OHIP="501NE" AND TestName="homoglobin"

The resulting query after translation is as follows.

Q8:

*select * from LabTest*

where PATID="243" AND Test="C0518015"

The translation algorithm uses mapping tables $OHIP2PATID$ and $TestName2Test$ for translating the query because the query attributes are covered by these two mapping tables. The translation is sound because the instance of relation LabTest of the Hospital peer satisfies the conditions of query Q8. Also tuples retrieved by the query Q8 can be associated with the mapping tables of 3.4. On the other hand the following query is not sound translation of query Q7.

Q9:

*select * from LabTest*

where PATID="243" AND (Test="C0518015" OR Test="C0518016")

In this example, we see that, the relation *LabTest* of Hospital peer satisfies the query $Q9$ but the value "$C0518016$" can not be associated with the mapping table 3.4(b).

In this thesis the translation algorithm incorporates the mapping tables into data association rules. Formally we must extend the definition of soundness to incorporate the translation rules seen in Section 4.3.

Definition 7 (Soundness w.r.t Translation Rules) *Let q_1, q_2 be queries over peer P_1 and P_2, respectively; Let $q_1 = \sigma_E(R_1 \bowtie .... \bowtie R_k)$, where E is a selection formula and $R_1, \ldots, R_k$ are relations in P_1. Then query q_2 is a sound translation of query q_1 with respect to a set of translation rules $\mathcal{R}$ and correspondence assertions $CA = \{ca_1, \cdots, ca_n\}$ between P1 and P2, denoted by $q_1 \overset{\mathcal{R},CA}{\longmapsto} q_2$, if $att(q_1) \subseteq att(CA)$, $att(q_2) \subseteq att(CA)$, and for every relation instance I_2 of P_2 and $t_2 \in q_2(I_2)$, there exists a tuple $t \in q_1(I_1)$, where I_1 is an instance of P_1, such that for all $r \in \mathcal{R}$,*

1. *if r is a merge rule (See rule (4.3.1)), then, for all selection terms $\sigma_{B=y}$ mentioned in q_2 there is a complex selection term $\sigma_{\bigwedge A_i=x_i}$, $1 \leq i \leq n$, mentioned in q_1, such that*

$$y = \Pi_B(f_M(R(A_1, \cdots, A_n), B) \bowtie T_{\bigwedge A_i=x_i}).$$

2. *if r is a data association rule, then use Definition 6.*

The cases of the separation and data conversion rules are treated similarly to the merge rule.

Example 8 *Consider the following query which retrieves all the information about a patient with Lname = "Lucas", Fname = "Andrew " and TestName = "whitebloodcount".*

Q10:

*select * from Patient,MedicalExam*

where Patients.OHIP=MedicalExam.OHIP AND (Lname="Lucas" AND Fname="Andrew")

AND Test="whitebloodcount"

From the above query the query attributes (Fname, Lname, TestName) matches the correspondence assertions of(Lname,Fname⟶Name and TesName⟶Test) which are associated with rule 1 (Merge rule) and 3 (Data Association rule) respectively. The rule 1 uses a function to convert Lname = "Lucas", Fname = "Andrew" to Name = "Lucas, Andrew". Rule 3 uses the mapping tables 3.4(b) to translate the vocabulary for peer Hospital. After applying the rules, the translated query becomes as follows:

Q11:

*select * from Patients, LabTest*

where Patients.PATID=LabTest.PATID AND Name="Lucas, Andrew" AND Test="C0427512"

which is a sound translation of query Q10. To see this, consider the instances of relations Patients and LabTest as well as mapping tables 3.4(a) and 3.4(b). Its tuples satisfy the conditions of the query Q11 and values of attribute Test can be associated with the mapping Table 3.4(b).

4.5 Query Translation Process

Now we describe our query translation strategy. The translation process starts when a user poses a query on its local database and wants a global execution of the query. The translation is completed in two phases. The first phase is completed locally where the query is originated and the second phase is completed in the remote peer. In the first

QueryTranslation (Q)
Input: A query Q from user.
Output: Translated query Q'
begin QC_T = Create AND-OR-NOT tree from the
query conditions;
CM = FindPartition(QC_T);
/*CM: Set of matchings for query conditions */
for each $cm_i \in CM$ **do**
Apply rule r_i associated with cm_i;
Translate the constraint using translation rules;
Translate (cm_i);
end for
end

Figure 4.2: Query translation main procedure

phase, the query is translated based on local information available at the acquainted peers, for instances, correspondence assertions and mapping tables. After translating the query, the peer sends the query to its acquaintances. In the first phase the query translation may be partial. Because we assume that the peers in the peer-to-peer network may not expose all their schema information. For instance a peer may export only a part of its schema elements. So the translated query might miss required relation names to translate the query completely.

4.5.1 Phase I

The algorithm to translate queries in phase I is shown in Figure 4.2. The algorithm mainly translates the selection part of a query. There are two steps for translating a selection of a query. Firstly, the query is analyzed syntactically and the query condition is transformed into a *query condition tree* which is defined as follows.

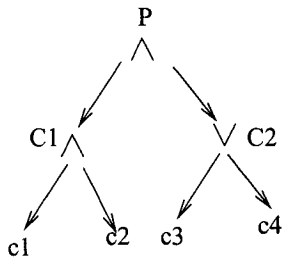


Figure 4.3: Condition tree

Definition 8 *Suppose q is a query. The Query Condition Tree(QCT) of q is a tree representing the selection formula of q, where the inner nodes are boolean operators AND, OR, or NOT, and leaves are atomic conditions of the form A op x, where A is an attribute, op is a comparison operator* $<, >, =, \leq, \geq$, *or* $\neq$, *and x is a constant value.*

Example 9 *Consider a query q with a predicate* $P = (C_1 \wedge C_2)$. *The predicate P is represented by a condition expression* C_i. *A condition expression involves either* $\vee$ *or* $\wedge$, *where each* C_i *consists of atomic conditions* c_i *or condition expressions* C_{ij}. *Therefore, if we prune a query q recursively in this way we find the atomic conditions* c_i *at leaf level. For instance the representation of the query predicate P of a query q is shown in Figure 4.3.*

Secondly, the query condition part is decomposed in terms of the correspondence assertions. The function *FindPartion* performs this task. The function *FindPartition* is shown in Figure 4.4. The algorithm takes as input a query tree and predefined set of correspondence assertions. Its output is a set $P = \{P_1, ...P_n\}$, where each P_i is a triple (T, CA, R); T is a set of predicate terms that are resulted from partitioning the original query predicate terms; CA is the corresponding correspondence assertion that maps the terms in T, and R is the corresponding translation rule that will be used to translate

the predicate terms in T. The function *FindPartition* finds the potential partitions that can be translated independently. That means partitions are disjoint and there is no dependency between the terms in the partitions.

Example 10 *Consider the following complex query:*

Q11:

select lname, fname, result

from Patient,MedicalExam

where Patient.OHIP=MedicalExam.OHIP AND

(((lname="Hall" OR lname="Hull") AND (fname="Andrew")) OR

(OHIP="233GA")) AND (TestName="whitebloodcount" AND Date="Jan/04")

where the predicate P is as follows:

((((lname="Hall" OR lname="Hull") AND (fname="Andrew")) OR (OHIP="233GA"))
AND (TestName="whitebloodcount" AND Date="Jan/04"))

Consider that we have following correspondence assertions:

ca1: $lname \longrightarrow name$

ca2: $lname, fname \longrightarrow name$

ca3: $OHIP \longrightarrow PAITD$

ca4: $TestName \longrightarrow Test$

ca5: $Date \longrightarrow Dt$

Assume the following representation of the predicate.

$A = (lname = \text{“}Hall\text{”}), B = (lname = \text{“}Hull\text{”}), C = (fname = \text{“}Andrew\text{”})$

$D = (OHIP = \text{“}233GA\text{”}), E = (TestName = \text{“}whitebllodcount\text{”}), F = (Date = \text{“}Jan/04\text{”})$

FindPartition(CTree, CA)
Input: A query condition tree CTree and set of Correspondence Assertions CA.
Output: Potential partitions $P = \{P_1, ...P_n\}$ of constraints
begin
1. $CM =$ /* CM is a set of pair (t, ca_i) where t is a atomic term
and ca_i is corresponding correspondence assertion that covers attribute of t

2. **for each** term t at leaf, $t \in T$ /* T is a set of predicate terms in CTree*/
3. $M = \{\}$ /* M is a set of matches found in CA for term t */
4. **for each** correspondence assertion $ca_i \in CA$
5. **if** $attribute(c) \in attribute(ca_i)$ then
6. $M = M \cup ca_i$
7. $CM = CM \cup (t, M)$
8. **end for**
9. **end for**

10. **for each** $m_i \in CM$
11. $m_k = \varnothing$
12. **for each** $m_j \in CM$ *and* $m_i \neq m_j$
13. **if** $(m_j(ca) \cap m_i(ca)) \neq \oslash$ then
14. $m_k(t, ca) = \{\{m_i(t) \cup m_j(t)\}, \{m_i(ca) \cap m_j(ca)\}\}$
15. $P = P \cup m_k(t, ca)$ /*Forming partition*/
16. $CM = CM - m_j$
17. **end for**
18. **if** $(m_k = \varnothing)$
19. $P = P \cup m_i(t, ca)$
20. $CM = CM - m_i(t, ca)$
21. **end for**
22. **end for**

23. **return** P
end

Figure 4.4: Finding partitions

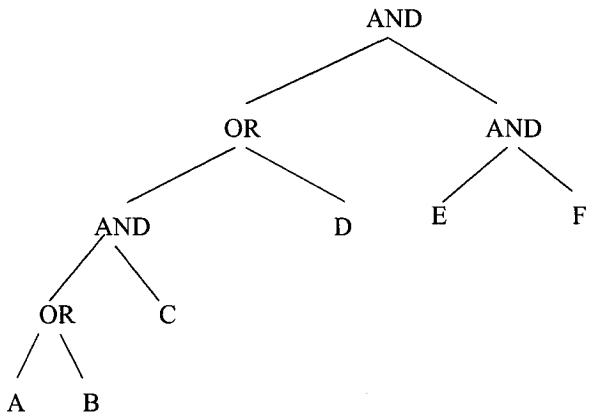


Figure 4.5: The condition tree

So the predicate becomes as follows:

$P = (((A\ OR\ B)\ AND\ C)\ OR\ D)\ AND\ (E\ AND\ F)$

The corresponding *condition tree* is shown in Figure 4.5. Also we assume that correspondence assertions ca1, ca5 are bound to rule 4(Data Conversion), ca2 is bound to rule 1(Merge Rule) and ca3, ca4 are bound to rule 3(Data Association).

Finding potential partitions is important, because the composition of two terms in a query predicate may map to a particular correspondence assertion. Also sometimes, it is possible that a term can not be translated independently but only in combination with another term can the composed term be mapped with a correspondence assertion.

The lines 2-9 of the algorithm *FindPartition* first find the mappings based on correspondence assertions. At this point we find the following initial mappings for our example:

lname = "Hall"

$M = [lname \longrightarrow name, lname, fname \longrightarrow name]$

$CM1 = [lname = \text{“}Hall\text{”}, [lname \longrightarrow name, lname, fname \longrightarrow name], 4]$

lname = “Hull”

$M = [lname \longrightarrow name, lname, fname \longrightarrow name]$

$CM2 = [lname = \text{“}Hull\text{”}, [lname \longrightarrow name, lname, fname \longrightarrow name], 4]$

fname = “Andrew”

$M = [lname, fname \longrightarrow name]$

$CM3 = [fname = \text{“}Andrew\text{”}, [lname \longrightarrow name, lname, fname \longrightarrow name], 1]$

OHIP=“233GA”

$M = [OHIP \longrightarrow PATID]$

$CM4 = [OHIP = \text{“}233GA\text{”}, [OHIP \longrightarrow PATID], 3]$

TestName=“whitebllodcount”

$M = [TestName \longrightarrow Test]$

$CM5 = [TestName = \text{“}whitebloodcount\text{”}, [TestName \longrightarrow Test], 3]$

Date=“Jan/04”

$M = [Date \longrightarrow Dt]$

$CM6 = [Date = \text{“}Jan/04\text{”}, [Date \longrightarrow Dt], 4]$

The lines 10-22 perform the task of finding potential partitions from the above mappings. Therefore we get the following partitions.

$P1 = [lname = \text{“}Hall\text{”}, fname = \text{“}Andrew\text{”}, [lname, fname \longrightarrow name], 1]$

$P2 = [lname = \text{“}Hull\text{”}, fname = \text{“}Andrew\text{”}, [lname, fname \longrightarrow name], 1]$

$P4 = [OHIP = \text{“}233GA\text{”}, [OHIP \longrightarrow PATID], 3]$

$P5 = [TestName = \text{“}whitebloodcount\text{”}, [TestName \longrightarrow Test], 3]$

$P6 = [Date = \text{“}Jan/04\text{”}, [Date \longrightarrow Dt], 4]$

Notice that, we can not simply translate the predicate terms independently without looking for the potential dependency from other terms. In our example in our example, the term fname =“Andrew” does not have any correspondence assertion but by combining it with the term lname=“Hall”, we can translate the *subquery* (lname=“Hall”, fname=“Andrew”). Because the terms can be mapped with the correspondence assertion $lname, fname \longrightarrow name$.

After rewriting the predicate we apply, the corresponding translation rule to each newly generated leaves of the *query condition tree*. The algorithm in Figure 4.6 depicts the translation mechanism.

The query translation algorithm depends on mapping tables and correspondence assertions. The algorithm translates SPJ queries, where selection formula may contain both conjunctive and disjunctive normal form with negation. Sometimes a situation may arise where a query is not translatable. There are three reasons for this case. First, no mapping exists in the mapping table to translate a predicate term. Second, there is no correspondence assertion that maps a predicate term. Third, there is no rule to support the translation of a predicate atom. In all such cases the query is rejected.

In peer database systems, the translation of a given query is not unique [24]. There could be many possible global queries for a particular query, because there is no certain control of query propagation in peer database networks. Also peers are free to join and

leave the system at will. Therefore, in most cases, we do not get *complete answer* (meaning all the matching tuples in the network), but at least we get some answer which can be recognized as complete enough with respect to the set of active peers.

4.5.2 Phase II

At the end of the first phase, a resulting query may still be incomplete, meaning that, e.g., not all the relations to be joined in the translated query have been found. In the second phase of the translation algorithm, the incomplete query received from an acquainted peer will be completed with respect to the locally stored relations. The algorithm finds relations and join operations for the query.

Phase II of the algorithm uses a data structure called *Database Graph* that each peer maintains to represent the relationships between relations in it's schema. Consider a local database schema of a peer as shown bellow.

AC(fno, depdate, deptime, dest)

Passenger(passengerID, fno, Name)

PassgDetails(passengerID, Address, TelPhone)

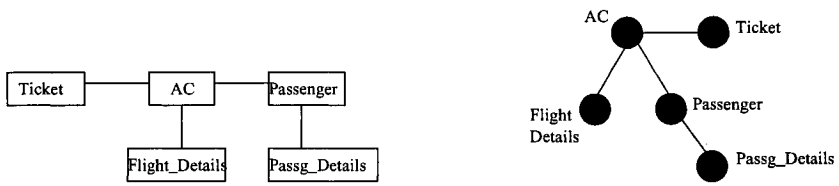
Ticket(fno,Ticketno, price, class)

FlightDetails(fno, meal, reserve)

We represent this database as a connected graph $G = (V, E)$, where the set of vertices V contains the relations of the database, and $(R_i, R_j) \in E$ if R_i and R_j have a common join attribute. The database graph for the example above (See also Figure 4.7(a)) is shown in Figure 4.7(b).

Translate(p_i)
Input: Potential partition p_i
Output: Translated query predicate c'
switch($p_i(r_i)$):
Case 1:
Find relation R if the attributes of $p_i(ca)$ is mentioned $\in R$
Compute $R'(A_1, A_2, ..A_n, B)$ with attributes of $p_i(ca)$
Create table T_c from the constraint of $p_i(c)$
Compute $y = \Pi_B(R' \bowtie T_c)$
Generate constraint c' of the form B *op* y
Return c'
Case 2:
Find relation R if attributes of $p_i(ca) \in R$
Create relation R'$(A, B_1, B_2, ..B_n)$ with attributes of $p_i(ca)$
Populate values of $(B_1, B_2, ..B_n)$ with the function
$f(A, B_1, B_2, \cdots, B_n)$
Create table T_c from the constraint of $p_i(c)$
Compute $\{y_1, y_2, ., y_n\} = \Pi_{B_1,B_2,..B_n}(R' \bowtie T_c)$
Generate new constraint c' with values y of the
form $\wedge(B_i$ *op* $y_i)$
return c'
Case 3:
Find mapping table M if attribute of $p_i(ca) \in M$
Create a table T from the constraint of $p_i(c)$
$Q_T = \Pi_B(M(A, B) \bowtie T_c)$
For each row w of the table Q_T
Generate new constraint c' as $\vee(B = \rho(w))$
Return c'
Case 4:
Find relation R if attribute of $p_i(ca) \in R$
Create relation R'(A,B) with attributes of $p_i(ca)$
Populate values of B with the function *f(A, B)*
Create table T from the constraint of $p_i(c)$
Compute $y = \Pi_B(R' \bowtie T_c)$
Generate new constraint c' with values y of the
form B *op* y
Return c'

Figure 4.6: Translation routine



(a) Schema of Local Database (b) Database Graph

Figure 4.7: Database schema and graph

The algorithm in Figure 4.8 summarizes Phase II.

Example 11 *Consider the following partially translated query received from a neighboring peer where the relation name is unknown. After receiving the query, the algorithm first finds the attributes involved in the query.*

select fno, Y.Name, Ticketno

from X, Y

where X.fno=Y.fno AND fno="LH541" AND (dest="N.Y" OR dest="L.A.")

Attributes found in the projection and selection are as follows:

Projection Attribute : fno, name, Ticketno

Predicate: fno="LH402" AND (dest="N.Y" OR dest="L.A.")

The set of attributes involved in the query is

$QA = \{name, Ticketno, fno, dest\}$

Set of relations in the local database is $V = \{$ *AC, Passenger, PassgDetails, Ticket, FlightDetails*$\}$

Line 3 to 10 find the relations that cover the attribute of QA. Therefore we find the following set

Input: Received partially translated query q, Local Database Graph (LDG)
Output: Set of relation pairs to form join operations in the query q

Begin
1. Let QA = Attributes involved in query q
2. V = Set of vertices of local database graph.
3. V' = Set of vertices that covers attributes of query q
4. J = Denotes the set of relations to be involved in join
5. **for each** $v \in V$
6. A = Attributes of v
7. **if** $QA \subseteq A$
8. $V' = V' \cup \{v\}$
9. break;
10. **else**
11. **if** $(A \cap QA \neq \oslash)$
12. $QA = QA - (A \cap QA)$
13. $V' = V' \cup \{v\}$
14. **end for**

15. Let J denotes the set of relations to be involved in join
16. **for each** $v \in V'$
17. **for each** adjacent v' of $v, v \in V$ and $v' \in V'$
18. **if** $((attribute(v) \cap attribute(v')))$
19. form a pair for join operation between v and v'
20. $J = J \cup \{v, v'\}$
21. $V' = V' - \{v\}$
22. **return** J.

Figure 4.8: Translating incomplete queries

$V' = \{AC, Passenger\}$

Lines 13-17 find the set of pair of relations that form a join in the query. First we take each relation v from V' and find adjacent vertex of v which is also an element of V'. If these two relations have common attributes we form a pair that makes a join in the query. In this way we proceed and form join operations and find relations to form a complete query that is executed in the local database. So the final pair for join operation of the above example is $\{AC,\ Passenger\}$. So the complete query is:

select fno, name
from AC, Passenger
where AC.fno=Passenger.fno AND fno=LH541 AND (dest="N.Y" OR dest="L.A.")

So, in the first phase we can only translate queries partially. When the remote peer receives this partially translated query, it uses its database graph to complete the translation.

Proposition 1 *The algorithm FindPartition generates subqueries from the query q based on correspondence assertions CA, so that the class of translation rules R can be applied to each subquery of the query q.*

It can be shown by induction over the structure of SPJ SQL queries that the translation algorithm outputs only sound queries:

Theorem 1 *The algorithm of Figure 4.2 outputs only sound queries with respect to translation rules.*

Proof Correctness entails that the algorithm *QueryTranslation* generates the correct translation of query q with respect to correspondence assertions and rules. The translation is completed applying rules on each semantic unit of predicates of the query $q\prime$ which is found after applying mappings on the predicates of the original query q using correspondence assertions. We can consider each unit as a subquery. Therefore, if the translation of each subquery of $q\prime$ is a sound then the query $q\prime$ is sound translation of query q.

To begin with, we consider the query condition tree. The query condition tree represents an arbitrary query where inner nodes are AND, OR, and NOT and leaves are predicate atoms. According to the proposition 1, the algorithm *FindPartition* rewrites the tree in terms of correspondence assertions so that the translation rules can be directly apply to each subquery using the algorithm *Translate*.

We start with a simple query q and prove by induction on $|q|$, i.e, the number of atomic predicate terms in q.

Case 1: $|q|$=1, number of atomic terms in q is 1. There are two possibilities in this case.

1. $attr(q) = att(ca)$: It follows that there is a translation rule r associated with ca that translates the term of q.

2. $attr(q) \subset att(ca)$, i.e., there is no exact matching but has partial matching. There are two possibilities here. As our correspondence assertions are complete then there must be a rule that will translate the term t of q. Or, there is another term t' exists in the query q that combines with t and maps with correspondence assertion ca. But this is not possible since q contains only one term.

Case 2:$|q|$=2 with AND node(q is a simple conjunctive form, i.e. $q = (t_1 \wedge t_2)$

There are two possibilities possibilities in this case.

1. $att(q) = att(ca)$: It follows that there is a translation rule r associated ca that translates terms of q, in our case it is rule 1.

2. Otherwise q is decomposable, i.e. there exists decomposition/partitions of terms of q, for instance q=t_1, t_2, such that for each t_i,$att(t_i) = att(ca_i)$. Therefore the translation is obvious

Case 3:$|q|$=k+1 with AND node (q is a simple conjunctive form, i.e. $q = (t_1 \wedge \cdots \wedge t_{k+1})$

1. $att(q) = att(ca)$: It follows that there is a translation rule r associated with ca that translates q, in our case it is rule 1.

2. Otherwise q is decomposable, i.e. there exists some proper subsets $t_1, \cdots, t_{k+1}$ of q, such that for each t_i there is $att(t_i) = att(ca_i)$. Therefore by induction hypothesis we see that for each t_i there is a translation rule r_i.

Case 4:q is in simple disjunctive normal form, i.e., $q = (t_1 \vee \cdots \vee t_k)$. We can prove this in the same way as we did for the conjunctive normal form.

Case 5:if q is an arbitrary predicate with conjunction and disjunction. Let the algorithm $FindPartition$ returns the following partitions $P_1, \cdots, P_k$. Therefore each partition is either disjunctive or conjunctive normal form. So according to case 1 to 4 the translation is obvious. Otherwise there is a partition P_i that can not be translated. In this case, either there is no correspondence assertion ca that cover attribute of P_i. But it is

not possible because according to the proposition 1 the algorithm *FindPartition* finds the potential partitions based on correspondence assertions that can be translated using corresponding translation rule. □

4.6 Summary

This chapter presented our core algorithm. The chapter started with describing the query language that our framework supports and later described the query execution semantics. Next we introduced the query translation rules that are used to translate queries. We also introduced the notion of a sound translation of queries with respect to mapping tables and correspondence assertions. Lastly, we elaborately described and illustrated our query translation algorithm using an example.

Chapter 5

Architecture, Implementation and Experimental Results

In this chapter we describe the architecture of our translation framework. We also discuss the implementation setup to evaluate our algorithms. We evaluated our solutions over a small peer-to-peer network and show the performance results obtained. These performance results will show that the framework is feasible for large network.

5.1 Architecture

The architecture of the query translation framework is depicted in Figure 5.1. The translation process starts when a user poses a query through the user interface and selects the global execution of the query. If the query is local then the query is processed locally. The main component of the architecture is the Query Translation Component which is the implementation of the algorithm in Figure 4.2. It transforms an input SPJ SQL query into a query tree and maps the query using correspondence assertions. Then

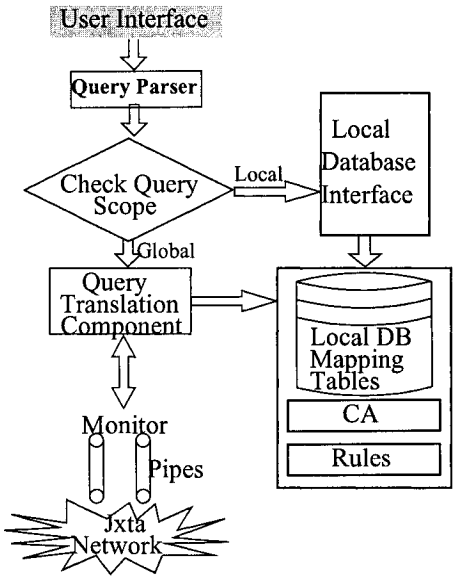


Figure 5.1: Architecture of the framework

it translates each semantic unit of the query using corresponding translation rules and mapping tables and then sends the query to the remote peer. The monitor component looks for incoming queries in the network and forwards them to its acquaintances. The monitor also collect results from acquaintances and forward them to the originator of the query.

5.2 Implementation

We implemented our query translation ideas over a prototype P2P data management system to evaluate the efficiency and performance. The prototype P2P setting is shown in Figure 5.2. We considered an application domain of flight information where the data includes flight information and passenger reservations. We considered six peers

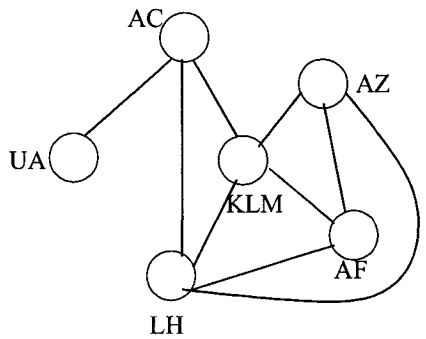


Figure 5.2: The P2P network

AC(fno, date, deptime, dest)
LH(fno, date, deptime, dest, arrtime)
AF(fno, date, deptime, dest, arrtime)
AZ(fno, date, deptime, dest, arrtime)
KLM(fno, date, deptime, dest, arrtime)

Figure 5.3: Schema of peers

represented six airlines, connected in unstructured fashion. The airlines are called Air Canada (AC), Air France (AF), Alitalia Flights(AZ), KLM, United Airline (UA), and Lufthansa (LF). The line between two peers shows the acquaintances between them. The acquaintances are established based on mapping tables. We see that peer UA has only a single acquaintance with AC airline. The peer KLM has four acquaintances. We assume that acquaintances are bidirectional.

We use MySQL to implement the Local database layer of each peer. Schemas and partial instances of each peer are shown in Figure 5.3 and Figure 5.4 respectively. Attributes *fno* denotes flight number. Attributes *date* and *deptime* together denote departure time of flights and attribute *dest* represents destination of flights. Some peers have extra attribute named *arrtime*, represents the arrival time of flights. There are around 50

Fno	Date	deptime	Dest
AC856	10/06/2003	18:15	LONDON
AC866	10/06/2003	19:15	LONDON
AC608	10/05/2003	06:45	HALIFAX
AC872	10/06/2003	17:45	FRANKFURT
AC700	10/05/2003	06:50	NEW YORK

(a) Instances of peer AC

fno	date	deptime	dest	ArrTime
UA928	10/05/2003	18:25	LHR	8:15
UA928	10/06/2003	18:25	LHR	8:15
UA940	10/06/2003	20:55	FRA	12:20
UA672	10/06/2003	8:00	LGA	11:05

(b) Instances of peer UA

Figure 5.4: Database instances of peer AC and UA

dest		dest
HALIFAX		YHZ
LONDON		LHR
NEW YORK		EWR
NEW YORK		JFK
NEW YORK		NYC
NEW YORK		LGA

(a) Instances of peer AC

fno		fno
AC856		UA928
AC872		UA940
AC700		UA670
AC700		UA672
AC700		UA674

(b) Mapping table dest2dest

Figure 5.5: Mapping table fno2fno

mapping tables to map flight numbers, destinations etc between partner airlines. Each mapping table contains average of 150 records. For instance, the instances of mapping tables between peers AC and UA are shown in Figure 5.5.

We choose JXTA for the communication layer because JXTA is an open network computing platform for P2P computing. It provides a common set of protocols and an open source reference implementation for developing P2P applications. Java is used for the programming language and the program length is around 5000 lines. The P2P platform is simulated on IBM computers with windows XP operating system. The CPU

```
****Original Query***
select lname,fname,address from lh,customer where (fno='LH402' OR fno='LH404')
OR (date='10/06/2003' AND deptime='02') AND dest='shanghai'
Parse Time=0.015
Elapsed time of Mapping:0.016
Elapsed time of PredicateTranslation:0.062
Elapsed time of Query Translation:0.093
*****Translated Query:****
select name, city,pcode from ac, passenger where (((fno='AC700') OR (fno='AC702
') OR (fno='AC704') OR (fno='AC706') OR (fno='AC712') OR (fno='AC714') OR (fno='
C716') OR (fno='AC718') OR (fno='AC720') OR (fno='AC724'))) OR (((fno='AC700')
R (fno='AC702') OR (fno='AC704') OR (fno='AC706') OR (fno='AC712') OR (fno='AC7
4') OR (fno='AC716') OR (fno='AC718') OR (fno='AC720') OR (fno='AC724'))) OR ((
date=10/05/2003 AND deptime=20))) AND (dest='shanghai')
```

Figure 5.6: An output of a translated query

is Pentium 4 3.0 GHz and RAM is 760MB.

5.3 Experimental Results

To evaluate our algorithm in its first phase we investigate three studies. We first investigate finding mapping times of predicates of the query because query translation mainly depends on predicate attributes and the number of disjuncts in the query. Also in a predicate there may be terms that may overlap with a common correspondence assertion. So finding mappings is a crucial step in query translation. Figure 5.6 shows a query and it's translated version. Figure 5.7 shows the mapping times and predicate translation times of queries. We ran this experiment with 8 queries where the number of disjuncts gradually ranges from 1 to 8. That is, the first query has one disjunct, the second two disjuncts, and so on. We see from Figure 5.7 that the mapping times increase gradually. The queries are chosen in such a way that there are interdependency between predicate terms in the query. Nicks on the curves are points where there is a change in the amount of interdependency among predicate terms.

We also show the time required for producing the number of disjuncts in the output

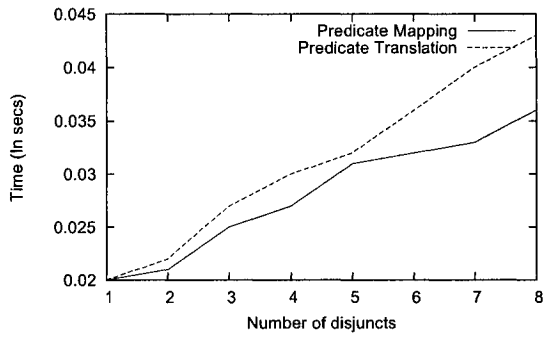


Figure 5.7: Predicate mapping and translation

query. Here, we tried to find the relationship between the time required to perform a query translation and the size, in terms of disjuncts, of the translated (or output) query. Figure 5.8 shows the translation times for queries which gradually produce a number of disjuncts ranging from 1 to 20. The interesting point to notice from Figure 5.8 is that it is not the number of disjuncts in the output query which is a relevant factor for the running times to generate these disjuncts. These times mainly depend on the number of disjuncts in the input query and on the dependency between predicate terms. The same figure also shows that input queries 1 and 10, which both have one single constraint, take almost the same running time, although they produce different numbers of disjuncts in the their translated form.

We also investigate performance of the algorithm for query translation with respect to the number of acquaintances. We experimented with 12 peers. We investigated the times required to translate one query (in a peer) for all the acquainted peers. Figure 5.9 gives a screen shot showing such a translation. We investigate with different input query frequencies. We notice that the translation times increase gradually with the number of peers and number of input queries per second. Figure 5.10 shows the result.

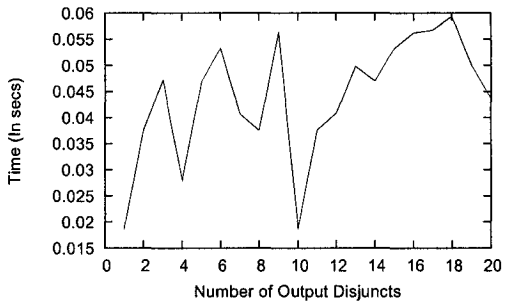


Figure 5.8: Number of disjuncts in the output query

```
***Original Query***
select * from lh where fno='LH9766'
*****Translated Query:*****
For Peer:ac
select fno, date,deptime, dest from ac where (((fno='AC700') OR (fno='AC702')
 (fno='AC704') OR (fno='AC706') OR (fno='AC712') OR (fno='AC714') OR (fno='AC7
') OR (fno='AC718') OR (fno='AC720') OR (fno='AC724')))
For Peer:klm
select fno, date,deptime, dest from klm where (((fno='KL0641') OR (fno='KL0643
'))
For Peer:af
select fno, date,deptime, dest from af where (((fno='AF004') OR (fno='AF006')
 (fno='AF008') OR (fno='AF022') OR (fno='AF6426') OR (fno='AF6498') OR (fno='A
F990') OR (fno='AF8994')))
For Peer:az
select fno, date,deptime, dest from az where (((fno='AZ00610') OR (fno='AZ0761
')))
Time Elapsed:0.375
```

Figure 5.9: Query translation for different peers

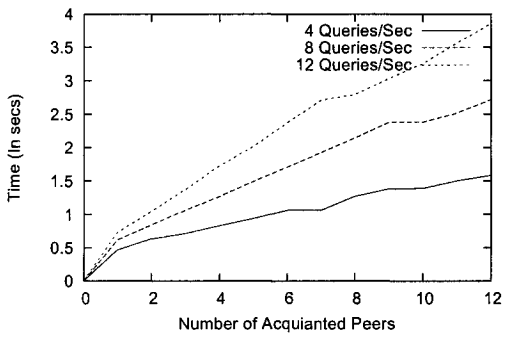


Figure 5.10: Performance(time) with respect to the number of acquaintances

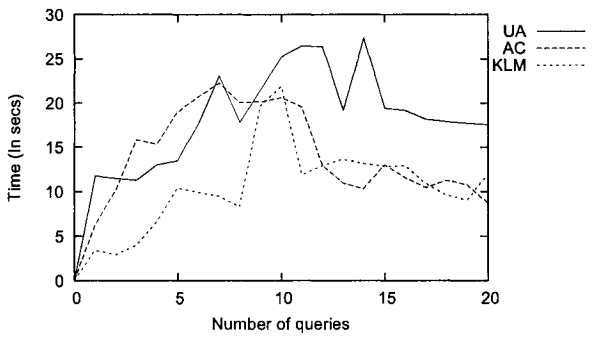


Figure 5.11: Global query completion time for peers UA, AC and KLM (Fully loaded)

We further investigate the execution of global queries generated from different peers in our P2P settings. We flooded the network generating 4 queries/sec from each peer with a total of 20 queries per peer. The total number of queries in the network was 624. The result is shown in Figure 5.11 and 5.12. The figures show that on average the global queries of peers KLM and LH finish first because they have more acquaintances than other peers in the settings. The global queries of the UA peer take the highest times because this peer is linked to the P2P network over one single acquaintance (with peer AC). Figure 5.13(a) and 5.13(b) show the time required to execute global queries

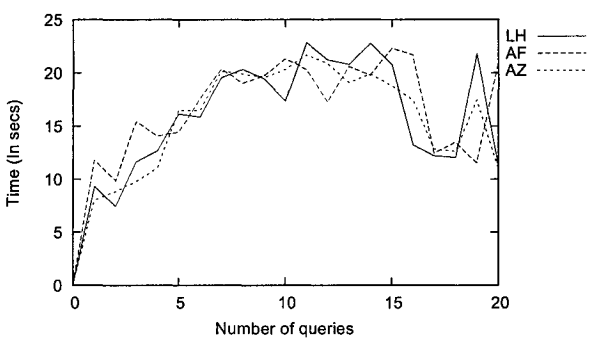
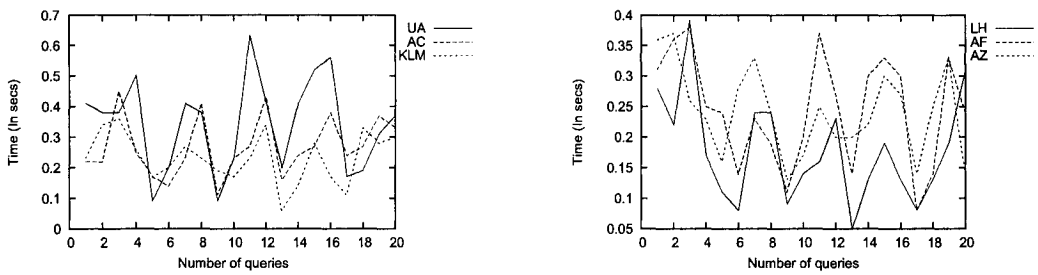


Figure 5.12: Global query completion time for peers LH, AF and AZ (Fully loaded)



(a) Peers UA,AC, and KLM

(b) Peers LH,AF, and AZ

Figure 5.13: Global query completion time (Normal load)

in each peer. Here we executed queries in each peer independently. The motivation of this experiment is to see the relationship between global query execution time and the position of a peer in the P2P settings. We notice from Figure 5.13(a) and 5.13(b) that the global queries generated by peers KLM and AC completed first. Because they have more acquaintances and they act as a hub in the network. Therefore, they can receive and forward queries simultaneously.

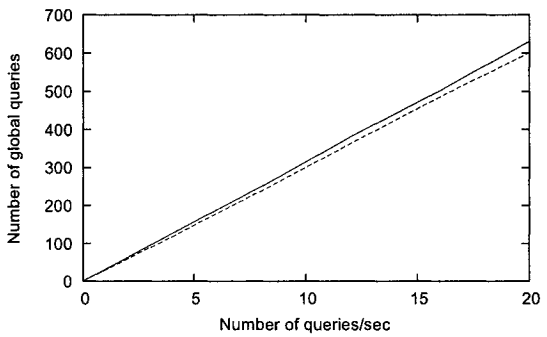


Figure 5.14: Reducing the number of global queries

We now consider an optimization to generate global queries that we subsequently compare with the approach we have persuaded before. We only included the query id

Queries/sec	Without Forward Tag	With Forward Tag
4	126	118
8	249	238
12	381	363
16	501	485
20	630	601

Figure 5.15: Number of global queries

and the peer id (where the query is originated and visited) in the query message. To illustrate our strategy, consider following scenario. Suppose that the peer KLM has generated a query q with an id $KLM1$, which represents a concatenation of the peer id KLM and the query id 1. This peer has four neighbors AC, LH, AF, and AZ; therefore it generates four queries with the same id $KLM1$. After receiving the query $KLM1$, peers AC, LH, AF, and AZ which are acquainted with KLM, now try to translate and forward new queries for their acquaintances. In our setting of Figure 5.2, peers LH, AF, AZ, and AC generate queries for AF and AC, for LH and AZ, for AF and LH, and for AC and LH respectively. We can see that all the queries are redundant for our settings because all the peers have already received the query $KLM1$ before from the peer KLM. In order to reduce the number of queries in the network, we investigated another strategy, adding a tag called *forward tag* in the query. In this strategy, a peer adds three tags before forwarding a query to its acquaintances. They are the query id, the peer id (where the query is originated), and a list of peer identifications (whom the query has been sent or forwarded to). We see from Figure 5.15, which compares both strategies, that the second approach reduces the number of global queries in the network.

5.4 Summary

In this chapter we explained our prototype architecture and depicted the P2P network that we used to perform various experiments. We tested the approach with an airline reservation database domain. We tested the performance of our algorithm from different perspectives, for example, we considered the input query size, the output query size, the number of queries generated per second, and the global query execution time with normal load and fully loaded networks. During our experiments, we mainly focused on time measurements in various cases. The measurements show that the algorithm is feasible for large networks of peer databases. If we consider Figure 5.10, where we gradually increase the number of acquaintances, we see that the translation time increases almost linearly. Our scheme is also scalable, because a peer needs only a pairwise mapping with its acquaintances when it joins the system; this ensures the autonomy of other peers. But we still have to test the approach in real large scale peer database networks to examine how the approach behaves at that scale.

Chapter 6

Conclusion and Future work

6.1 Conclusions

This thesis has offered an expressive query translation mechanism in a peer database network considering the semantic relationships between peer sources. The semantic relationships are formed through correspondence assertions that incorporate instance level and schema level mappings. Our strategy can translate a query if there are appropriate correspondence assertions between the schema elements of acquainted peers. In our translation framework, we consider peers are autonomous and they form pairwise mappings. We avoid centralized or global schema because in peer-to-peer environments the concept of global/centralized schema is not feasible. We also consider databases are heterogeneous in terms of structural, format, and data level conflicts. Moreover, we introduce four generic query translation rules to resolve above mentioned heterogeneity

The thesis also extends the notion of sound translation of queries presented in [25]. We also define the soundness criteria of query translation in terms of a set of rules. Moreover, we performed various experiments to evaluate the performance of the ideas

and demonstrate the feasibility and applicability of the proposed approach. The results show that the algorithm is feasible over large network.

6.2 Future Work

In the future work, we plan to investigate the approach as a query service built on top of the Hyperion [3] peer data management system. We also plan to investigate the dynamic inference of new correspondence assertions from existing ones. Such a dynamic inference mechanism seems necessary to avoid doing manually generate correspondence assertions when peers join/leave the system.

In our framework we avoid the partial translation of queries. For example, a problem might concern whether the present set of mapping tables and correspondence assertions can not translate a query fully. Formally, given a local peer database schema $DB[W] = R_1[U_1, \cdots, R_i[U_i]$, and a set of mapping tables $M = \{m_1, \cdots, m_n\}$ and correspondence assertions $CA = \{ca_1, \cdots, ca_n\}$ do they guarantee that every query vocabulary of q over DB is expressible with M and CA for its acquaintances? Moreover, is it possible to characterize the queries that are not expressible or is it possible to generate queries that would complement M and CA? We have no concrete answers to the above questions.

Bibliography

[1] David S. Johnson and M. R. Garey. Computers and Intractability: A guide to NP-Completeness. W. H. Freeman, 1979.

[2] S. Abiteboul, S. and Hull, R. and V. Vianu. Foundations of Databases. Addison-Wesley, 1995.

[3] Hyperion Project. World Wide Web URL: *http://www.cs.toronto.edu/db/hyperion/*

[4] Freenet. World Wide Web URL: *http://www.freenet.com*

[5] M. Boyd, S. Kittivoravitkul, C. Lazanitis, P. McBrien, N. Rizopoulos. AutoMed: A BAV Data Integration System for Heterogeneous Data Sources. In *CAiSE*, pages 82-97, 2004.

[6] R. Domenig, K.R. Dittrich. Query Exploративeness for Integrated Search in Heterogeneous Data Sources. In *CAiSE*, pages 715-718, 2002.

[7] The JXTA Project. World Wide Web URL: *http://www.jxta.org*

[8] L. Serafini, F. Giunchiglia, J. Molopoulos, and P. Bernstei. Local Relational Model:A Logical Formalization of Database Coordination. In *CONTEXT*, pages 286-299, 2003.

[9] M. Lenzerini. Data Integration: A Theoretical Prespective. In *PODS*, pages 233-246, 2002.

[10] R. J. Miller, L. M. Haas, and M. Hernndez. Schema Mapping as Query Discovery. In *VLDB*, pages 77-88, 2000.

[11] Z. Ives A. Halevy, M. Rodrig, and D. Suciu. What can Databases do for Peer-to Peer? In WebDB. In *WebDB*, pages 31-36, 2001.

[12] Kazaa. World Wide Web URL: *http://www.kazaa.com*

[13] M. Arenas, V. Kantere, A. Kementsietsidis, and I. Kiringa. The Hyperion Project: From Data Integration to Data Coordination. In *ACM SIGMOD RECORD*, 32(3):53-58, 2003.

[14] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, and J. Mylopulos. Data Management for Peer-to-Peer Computing: A vision. In *WebDB*, 2002.

[15] C.-C. K. Chang and H. Garcia-Molina. Mind Your Vocabulary: Query Mapping Across Heterogeneous Information Source. In *SIGMOD*, pages 335-346, 1999.

[16] C.-C. K. Chang and H. Garcia-Molina. Conjunctive Constraint Mapping for Data Translation. In *ACM DL*, pages 49-58, 1998.

[17] C.-C. K. Chang and H. Garcia-Molina. Approximate Query Translation Accross Heterogeneous Information Sources. In *VLDB*, pages 566-577, 2000.

[18] Napster. World Wide Web URL: *http://www.napster.com*

[19] F. Giunchiglia and I. Zaihrayeu. Making Peer Databases Interact-A vision for an Architecture Supporting Data Coordination. In *CIA*, pages 18-35, 2002.

[20] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management System. In *ICDE*, pages 505-516, 2003.

[21] L. Haas, D. Kossmann, E.L. Wimmers, and J. Yang. Optimizing Queries Across Diverse Data Sources. In *VLDB*, pages 276-285, 1997.

[22] V. Kantere, I. Kiringa, and J. Mylopoulos. Coordinating Peer Databases Using ECA Rules. In *P2P DBIS*, pages 108-122, 2003.

[23] V. Kantere, J. Mylopoulos and I. Kiringa. A Distributed Rule Mechanism for Multidatabase Systems. In *CoopIS*, pages 56-73, 2003.

[24] A. Kementsietsidis and M. Arenas. Data Sharing Through Query Translation in Autonomous Sources. In *VLDB*, pages 468-479, 2004.

[25] A. Kementsietsidis, M. Arenas, and R.J. Miller. Data Mapping in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In *SIGMOD*, pages 325-336, 2003.

[26] A. Levy, A. Rajaraman, J.J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB*, pages 251-262, 1996.

[27] W. S. Ng, B. C. Ooi, K. L. Tan, and A. Y. Zhou. PeerDB:A P2P-Based System for Distributed Data Sharing. In *ICDE*, pages 633-644, 2003.

[28] W. S. Ng, B. C. Ooi, and K. L. Tan. BestPeer:A Self-Configurable Peer-to-Peer System. In *ICDE*, pages 272, 2002.

[29] V. Papadimos and D. Maier. Mutant Query Plans. In *Information and Software Technology*, 44(4):197-206, 2002.

[30] I. Tatarinov, Z. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, and X. Dong. The Piazza Peer Data Management Project. In *SIGMOD*, pages 47-52, 2003.

[31] R. Huebsch, J.M. hellerstein, N.L. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *VLDB*, pages 321-332, 2003.

[32] L. Lintao, K.D. Ryu, and K. Lee. Keyword Fusion to Support Efficient Keyword-based Search in Peer-to-Peer File Sharing. In *Proceedings of the Fourth International Workshop on Global and Peer-to-Peer Computing*, 2004.

[33] P. reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. In *Middleware*, pages 21-40, 2003.

[34] M.T. Ozsu and P. Valduriez. Principles of Distributed Database Systems. Prentice Hall, Upper Saddle River, 2nd Edition, 1999.

[35] A. Elmagarmid and M. Rusinkiewicz and A. Sheth. Management of Heterogeneous and Autonomous Database Systems. Morgan Kaufmann Publishers, 1999.

[36] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. In *VLDB*, pages 334-350, 2001.

[37] T. Milo and S. Zohar. Using Schema Matching to Simplify Heterogeneous Data Translation. In *VLDB*, pages 122-133, 1998.

[38] J. Madhavan, A. Rajaraman, and J. J. Ordille. Generic Schema Matching with Cupid. In *VLDB*, pages 49-58, 2001.

[39] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering Queries Using Views. In *PODS*, pages, 95-104, 1995.

[40] A. Y. HaLevy. Answering Aueries Using Views: A Survey. In *VLDB Journal*, 10(4):270-294, 2001.

[41] Gnutella. World Wide Web URL: *http://www.gnutelliums.com*

[42] R. Fagin, P. Kolaitis, R. Miller, and L.Popa. Data Exchange: Semantics and Query Answering. In *ICDT*, pages 207-224, 2003.

[43] E. Franconi, G. Kuper, A. Lopatenko, and I. Zaihrayeu. Queries and Updates in the coDB Peer to Peer Database System. In *VLDB*, pages 1277-1280, 2004.

[44] E. Franconi, G. Kuper, A. Lopatenko, and I. Zaihrayeu. The coDB Robust Peer-to-Peer Database System. In *SEDB*, pages 382-393, 2004.