Data Sharing Through Query Translation in Autonomous Sources

Anastasios Kementsietsidis Marcelo Arenas

Dept. of Computer Science
University of Toronto
{tasos,marenas}@cs.toronto.edu

Abstract

We consider the problem of data sharing between autonomous data sources in an environment where constraints cannot be placed on the shared contents of sources. Our solutions rely on the use of mapping tables which define how data from different sources are associated. In this setting, the answer to a local query, that is, a query posed against the schema of a single source, is augmented by retrieving related data from associated sources. This retrieval of data is achieved by translating, through mapping tables, the local query into a set of queries that are executed against the associated sources. We consider both sound translations (which only retrieve correct answers) and complete translations (which retrieve all correct answers, and no incorrect answers) and we present algorithms to compute such translations. Our solutions are implemented and tested experimentally and we describe here our key findings.

1 Introduction

We consider the problem of data sharing between autonomous structured data sources. Such sources may use different schemas to structure their data. Furthermore, both the data and the schemas of the sources may overlap little, if at all. Still, data residing in the different sources may be closely associated. As an example, consider the domain of biological data sources. Different biological data sources can store inherently different data which range from data for genes or proteins to data for genetic diseases. Nevertheless, these diverse data sets are closely associated since genes *encode for* proteins and are related to genetic diseases.

How can we share data in such a setting? In the domain of biological sources, we can imagine that a biologist who queries her local source for information on a particular protein, say OPH, would like to retrieve, in addition to the local data, related information that is found in any number of networked sources including related genes that encode for protein OPH and genetic diseases related to these genes. To support such sharing of data, we must be able to translate the local query into the vocabulary of the other sources. This involves translating both the structure of the query to use the schema elements of the associated sources, but also the data itself. For example, associated sources may use synonyms of protein OPH, such as APH and AARE, or distinct identifiers for the same gene.

Notice that the related data that each source returns may be very different and it is often not possible to make this data conform to the local schema. For conformance to be possible, the local database must have anticipated the structure of all possible answers to a query. Mappings are needed to fit these structures into the local schema. Such a solution may be undesirable for several reasons. First, we may not wish to change the local schema to accommodate data for which it was not designed. In our example, the query results may be returned to users and not stored locally, so it seems onerous to insist that even virtual local structures be predefined for receiving this data. Second, if any of the networked data sources change their schemas, the local database must, somehow, become aware of this and update its mappings. Otherwise, the translated

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

queries may not be valid on the updated sources. Again, this is an onerous requirement.

Data sharing deals with the exchange of data between heterogeneous sources whose data need not be interdependent and may represent different real world domains. In keeping with the literature, we refer to such autonomous, heterogeneous sources as peers. Data sharing between peers differs from the well-studied problems of data integration [12] and data exchange [6]. The latter two problems use schema-level mappings to express the relationships between heterogeneous schemas. In data integration, these mappings are used, at run time, to conform the data of one source to the schema of another. In data exchange, the mappings are used to populate a target schema with the data of a source schema.

In this work, we consider how to translate queries in the absence of such restrictive schema-level mappings. We make use of a form of data-level mappings called *mapping tables* which we first introduced in [11]. In brief, a mapping table contains a set of data associations between data values in two peer databases. Our previous work showed how to *automate* the management of mapping tables by checking the consistency of the associations and by inferring new associations from existing ones. Our current work focuses on how to use mapping tables during query answering. Our main contributions are:

– We introduce the semantics of query answering in an environment of autonomous peers. The semantics relies on the translation of queries between the peers through the use of mapping tables.

– We introduce the notions of sound translations (which only retrieve correct answers) and complete translations (which retrieve all correct answers, and no incorrect answers) to characterize the relationship between translated queries.

– We extend the definition of mapping tables to store not only associations between data values, but also associations between pairs of translated queries. This common representation of different types of associations permits more systematic and robust solutions for managing the associations.

– We present an algorithm for computing complete translations and an algorithm for testing if a query is a sound translation of another. We use the latter algorithm, and our ability to store past translations in mapping tables, to determine if a query can be translated (partially or in full) by means of the stored translations.

This paper is organized as follows. We motivate our solutions in Section 2, while Section 3 describes the related work. Section 4 presents the semantics of query answering and introduces the notions of sound and complete translations. Section 5 presents the algorithms for computing such translations. Section 6 discusses our implementation while Section 7 presents the experimental results. We conclude in Section 8 with a summary of the work.

2 Motivating example

In what follows, we consider two biological databases, namely MedLine and PubMed [1]. A portion of their schemas and instances can be seen in Figures 1 (a) and (b), respectively. Both databases store similar information about articles, namely, an article identifier, some keywords, which refer to protein names mentioned in the article, and date of publication (PubMed stores the month (*pm*) and year (*py*) of publication, while MedLine stores only the year). In spite of the similarities in their schemas, the databases use different vocabularies to describe articles. For one thing, the two databases use their own local identifiers. Furthermore, they often refer to the same protein by using different names. For instance, OPH in MedLine and APH in PubMed refer to the same protein. We can use mapping tables to represent how values from different vocabularies may correspond [11]. In the same figure, we show examples of such tables. Mapping table *keyword2kw* associates keywords from MedLine to keywords in the PubMed relation. Notice that not all keywords from MedLine are mapped, that is, the tables might be incomplete. Mapping table *id2id* stores the identifiers of articles that are mentioned in both databases. Finally, mapping table *year2yr* uses a variable in its single tuple to represent the identity function, i.e., that each year in the first database is mapped to itself in the second. We note that not all attributes need to be mapped through mapping tables. For example, no table involves attribute *pm* of the PubMed relation.

Example 1 *Assume that a user wants to retrieve all MedLine articles that mention protein OPH. Then a query such as the following may be used:*

$\mathcal{Q}_1$: ***select*** ***
from *MedLine*
where *keyword = "OPH"*

What if this user also wants to retrieve all PubMed articles mentioning the same protein? Given that APH and AARE are synonyms of OPH, the following query may be used:

$\mathcal{Q}_2$: ***select*** ***
from *PubMed*
where *(kw = "APH"* ***OR*** *kw = "AARE")*

Mapping tables might provide us with sufficient information to automate the process of translating a query posed against one database to a query posed

article_id	*keyword*	*year*
20185348	OPH	2000
96281126	OPH	1996
87051725	NGF receptor	1986
99455262	CRAF1	1999
99455262	TNF receptor	1999

(a) MedLine relation instance

paper_id	*kw*	*py*	*pm*
10719179	APH	2000	March
8724851	AARE	1996	February
9915784	p75 ICD	1999	January
10944856	Sialidase 1	2000	July

(b) PubMed relation instance

keyword	*kw*
OPH	APH
OPH	AARE
NGF receptor	p75 ICD
G9 sialidase	Sialidase 1

(c) Table keyword2kw

article_id	*paper_id*
20185348	10719179

(d) Table id2id

year	*py*
$\mathcal{X}$	$\mathcal{X}$

(e) Table year2py

Figure 1: Instances and Mapping tables

against another, where both queries retrieve *related* data. The details of how this is achieved, and under which circumstances such a translation is possible, is the main focus of this work.

Notice that the retrieved data do not conform to the same schema. Even in this simple scenario, where the schemas are rather homogeneous, we cannot *merge* the results due to the difference in vocabularies. In general, we will not know before-hand what data we are going to retrieve and in what format. We expect that even the types of the retrieved data may differ significantly. For example, our biological scenario includes not only information about protein articles but also data about genes and diseases. One of the objectives of this work is to deal with this heterogeneity in the retrieved results.

Example 2 *Continuing with our example, assume now that the user decides to execute query $\mathcal{Q}_3$ which retrieves PubMed articles mentioning protein APH:*

$\mathcal{Q}_3$: ***select*** ***
from *PubMed*
where *kw = "APH"*

Intuitively, query $\mathcal{Q}_3$ satisfies the initial user selection requirements since it returns PubMed articles mentioning protein OPH. However, it does not retrieve all such articles. So, query $\mathcal{Q}_3$ is incomplete, compared to $\mathcal{Q}_2$. Nevertheless, neither query $\mathcal{Q}_2$ nor query $\mathcal{Q}_3$ return any incorrect answers, i.e., articles not mentioning protein OPH.

Notions such as correctness (soundness) and completeness of query translations are formalized in the next sections. Soundness is a property that every translation must satisfy, however, executing queries that are incomplete is often sufficient. For one thing, users are often satisfied with incomplete answers if complete answers are overwhelming. We may also be able to cache the results from sound queries to deliver some answers faster. From a systems point of view, we show that significant savings in computation time can be achieved by reusing sound queries.

Our last remark concerns our representation of queries. Mapping tables allow us to store, as part of our database, the associations of values between different peers [11]. Motivated by the same rational, we offer here a similar representation for associating queries and their translations. This uniform representation allows us to develop a common set of tools to manage both data and query associations.

3 Related work

Our previous work on mapping tables focuses on the management of the tables and how these can be used for value-based lookups [11]. Thus, it does not consider structured queries. In the context of peer-to-peer systems, advanced query mechanisms have been proposed by Harren et al [8] and Huebsch et al [9]. The latter work proposes structured query answering in an architecture that can scale to accommodate a large number of peers. However, peers must agree to support a common schema. Our work does not consider scalability issues but addresses instead issues relating to the heterogeneity of peers.

In Piazza, associations between peers are expressed through either global-as-view (GAV) or local-as-view (LAV) schema-level mappings [7, 14]. Both types of mappings are considered while translating queries between different peers. Our solutions are complimentary to this work since our framework operates in the absence of schema-level mappings and the only mappings used are in the data-level and have the form of mapping tables. The main difference between the two approaches is that while their work assumes that the retrieved data can be made to conform to the schema of the peer where a query is initiated, we make no such assumption.

Ng et al [15] also deal with the translation of queries in a network of peers. Descriptive keywords are used to associate schema elements of different peers. Then, the translation of queries is performed using the associated elements. A limitation of the approach is the underlying assumption that the keywords are used consistently throughout the peer network. Thus, unlike our work, their solutions cannot handle differences in the vocabularies within the data values of the peers.

The work of Chang and Garcia-Molina [5] also deals with the translation of queries between heterogeneous sources. There, syntactic rules are used to map selection predicates from one database to that of another. At first glance, mapping tables look like *materializations* of these syntactic rules. However, the two constructs operate under different assumptions. A syntactic rule that maps MedLine article identifiers to PubMed article identifiers assumes that for any identifier of the former we can *compute* an identifier of the latter. Thus, the query translation process relies on this assumption to translate a query from the former database to one in the latter without having to deal with the intricacies of the mappings at the data level, that is, the fact that mappings are often incomplete. Our work makes no such assumptions and our translation techniques deal with exactly these circumstances. We also use a *uniform* representation both for the rules, i.e., the mappings between data values and for the queries and the mappings between translated queries.

4 Query semantics

We assume that query execution in our peer network uses a *gossiping* mechanism. The process is initiated by the execution of a user-defined query locally in a peer. Then, the user-defined query is forwarded either *as is* or in some translated form to either all or to a selected number of acquaintances of the current peer. Then, the *execute-and-forward* step is repeated in each of the forwarded peers, causing in turn the further propagation of the query. The process terminates after a fixed number of propagations of the initial user-defined query has occurred.

In accordance with the above, we assume hereafter that each query is defined, in terms of syntax, with respect to the schema of a single peer. Our thesis is that for a user to issue a query, she need only be aware of the local schema she is using. Over this schema, we assume that the user poses queries that involve only the operations of selection, projection and join. Still, this is a significant extension of the value-based lookups supported thus far. In terms of execution, queries are classified into two categories. A *local* query, much like a query in a centralized system, is executed using only the data in the local peer. On the other hand, a *global* query uses the peer network to augment locally retrieved data with data that reside in other peers. We now formalize the above notions and explain the query semantics with emphasis on the semantics of global queries.

Consider a set $P = \{P_1, P_2, \ldots, P_n\}$ of n peers. Assume that peer P_i exposes a set of attributes U_i $(1 \leq i \leq n)$ and that $U_i \cap U_j = \emptyset$ $(i \neq j,$ $1 \leq i, j \leq n)$, and let r_i be an instance of P_i. A local query q, hereafter just called a query, over a

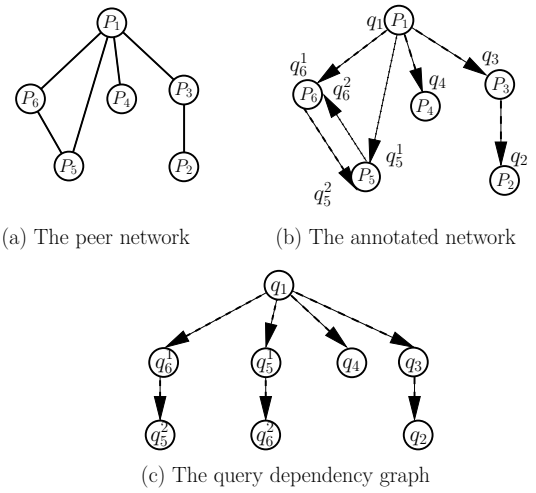

Figure 2: A query dependency graph

peer P_i $(1 \leq i \leq n)$ is defined with respect to the schema of P_i. The result of q is a relation over a set of attributes V_i, where $V_i \subseteq U_i$, whose content is the set of tuples $q(r_i)$. We denote by $att(q)$ the set of attributes V_i. A global query q_P over the peers in P is a set of queries $\{q_1^1, q_1^2, \ldots, q_1^{k_1}, \ldots, q_n^1, q_n^2, \ldots, q_n^{k_n}\}$, where query q_i^j $(1 \leq i \leq n$ and $1 \leq j \leq k_i)$ is over the schema of peer P_i and $att(q_i^j) = att(q_i^l)$, for every $l \in [1, k_i]$. Each query in q_P is called a *component* of q_P and, conversely, q_P is said to be *comprised* of the indicated set. The result of global query q_P is a relation over the schema $R_P[V_1 \cup V_2 \cup \ldots \cup V_n]$, where $V_i = att(q_i^1) = \cdots = att(q_i^{k_i})$ $(1 \leq i \leq n)$, whose content is the set of tuples in the outer-union of the union of the queries in each peer, that is, the outer-union of $\bigcup_{j=1}^{k_1} q_1^j(r_1)$, $\bigcup_{j=1}^{k_2} q_2^j(r_2)$, ..., $\bigcup_{j=1}^{k_n} q_n^j(r_n)$. Hence, we permit peers to return results with different schemas. We also propose not to merge results as merging semantics tend to be application specific. Given these definitions, each local query can be thought of as a trivial global query which is comprised of a single component query.

Example 3 *The set $\{\mathcal{Q}_1, \mathcal{Q}_2\}$ of queries (from Example 1) is a global query over the MedLine and PubMed peers. The execution of this global query is initiated by local query $\mathcal{Q}_1$. The result of this global query is a relation with attributes {article_id, keyword, year, paper_id, kw, py, pm}, whose content is the outer-union of the relations obtained by applying queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$ to the instances shown in Figures 1 (a) and (b), respectively.*

We now examine the relationship between the component queries of a global query. For this, we introduce the notion of query *dependencies*. Intuitively, as queries are propagated in the system, a

(directed) query dependency graph is induced. The nodes in this graph represent queries and there is an edge from query q_j to query q_k if query q_k *depends* on query q_j. A query q_k on peer P_k is said to depend on query q_j over peer P_j, denoted as $dep(q_k) = q_j$, if peers P_j and P_k are acquainted and query q_k has resulted from the propagation of query q_j on peer P_j to peer P_k. Special care is taken to avoid the creation of cycles in the induced graph. This happens if a query propagated by a peer P_k is re-received by the same peer through one of its acquaintances. Tagging queries with the path of peers through which they are propagated allows the detection of such situations. In Figure 2(a) there is an example of a peer network that consists of six peers, i.e., $P = \{P_1, P_2, \ldots, P_6\}$ and there is an edge between two peers if they are acquainted. In Figure 2(b), we use an intermediate representation where each peer in the network is annotated with the component queries of global query $q_P = \{q_1, q_2, q_3, q_4, q_5^1\ q_5^2, q_6^1, q_6^2\}$ that it executes. The edges in this representation show the propagation of queries. Finally, Figure 2(c) shows the dependency graph of q_P. Notice that in some peers more than one query is executed. For example, in peer P_5 we have the execution of two queries, namely q_5^1 and q_5^2. Query q_5^1 results from the propagation of query q_1 from peer P_1 while query q_5^2 results from the propagation of query q_6^1 from peer P_6. We devote the next paragraphs examining how exactly the propagation, and possible translation, of queries is achieved between peers.

4.1 Mapping tables

We offer here an overview of mapping tables since they are the main vehicle used for query translation.

Consider two peers that expose relations with attributes U and V, respectively. A mapping table is a relation over the attributes $X \cup Y$, where $X \subseteq U$ and $Y \subseteq V$ are non-empty sets of attributes from the two peers. For example, a mapping table from a set of attributes $X = \{keyword\}$ to a set of attributes $Y = \{kw\}$ is shown in Figure 1(c). A vertical double line is used to separate the two attributes sets.

To represent different semantics for mapping tables and values within them, the standard convention of using variables is followed. For instance, Figure 1(e) shows a mapping table containing variables. Every valuation of these variables gives a value of *year* that can be mapped to a value of *py*. Since this mapping table contains the same variable in its two columns, every valuation is a tuple of the form (a, a), where a is a constant in the domains of *year* and *py*. Thus, in this case variables offer a compact way of representing the identity mapping.

Mapping tables restrict the way in which information may be exchanged between peers, instead of restricting their contents. Let r_1 and r_2 be instances of peers P_1 and P_2, respectively, and m be a mapping table from X to Y, where X and Y are subsets of the set of attributes exposed by P_1 and P_2, respectively. Given a valuation ρ of the variables of m, a value $x \in \pi_X(\rho(m))$ is associated with the set of values $\pi_Y(\sigma_{X=x}(\rho(m)))$ and, hence, each $t_1 \in r_1$ such that $t_1[X] = x$ can be mapped only to the tuples $t_2 \in r_2$ for which $t_2[Y] \in \pi_Y(\sigma_{X=x}(\rho(m)))$.

4.2 Sound and complete translations

We are interested in translating, through mapping tables, queries that involve the operations of projection, selection and join. In what follows, we consider how this is achieved in the presence of the latter two operators. Then, in Section 4.3 we show how to handle projections. This separation is possible since, as we show, the issues involved are orthogonal.

Consider peers P_1 and P_2 that expose attributes U_1 and U_2, respectively. To begin, we assume that a single mapping table m exists with schema $M[U_1 \cup U_2]$ that associates values of U_1 to values of U_2. We relax this assumption later. Consider two queries q_1 and q_2 over peers P_1 and P_2, respectively, such that $dep(q_2) = q_1$, that is, query q_2 resulted from the propagation and translation of query q_1. We claim that the nature of this translation should be such that q_2 retrieves from peer P_2 only the data that are related with those that could be retrieved from query q_1 in peer P_1. The exact relationship is determined by the set of mapping tables that exists between the two peers.

Definition 4 *Let q_1 and q_2 be queries over peers P_1 and P_2, respectively, such that $q_1 = \sigma_E(R_1 \bowtie \cdots \bowtie R_k)$, where E is a conjunction of equality atoms and $R_1, \ldots, R_k$ are relations in P_1. Then q_2 is a* sound translation *of q_1 with respect to mapping table m, denoted by $q_1 \overset{m}{\mapsto} q_2$, if for every relation instance r_2 of P_2 and $t_2 \in q_2(r_2)$, there exists a valuation ρ of m and a tuple $t \in \sigma_E(\rho(m))$ such that $\pi_{att(q_2)}(t) = t_2$.*

We offer a few remarks on our definition. First, observe that in the definition query q_1 operates on mapping table m, instead of operating on some instance r_1 of P_1, while query q_2 operates on an instance r_2 of P_2. This is to allow for query q_2 to retrieve data from r_2 that could be mapped to some instance r_1 of P_1 through m, but that it is not necessary for the data retrieved from q_1. Second, note that since m contains all the attributes mentioned in $R_1, \ldots, R_k$, in order to evaluate q_1 in the relation $\rho(m)$ we do not need to compute the join of $R_1, \ldots, R_k$, we just have to check the condition E.

Example 5 *Consider the mapping table ML2PM shown in Figure 3. Since all variables in the ta-*

article_id	*keyword*	*year*	*paper_id*	*kw*	*py*	*pm*
$\mathcal{X}_1$	OPH	$\mathcal{Y}_1$	$\mathcal{X}_2$	APH	$\mathcal{Y}_2$	$\mathcal{Z}_2$
$\mathcal{X}_3$	OPH	$\mathcal{Y}_3$	$\mathcal{X}_4$	AARE	$\mathcal{Y}_4$	$\mathcal{Z}_4$

Figure 3: Mapping table ML2PM.

ble are distinct, the table essentially maps protein OPH in MedLine to proteins APH and AARE in PubMed. Now, consider queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$ from our motivating example (see Section 2). Query $\mathcal{Q}_2$ is a sound translation of query $\mathcal{Q}_1$ with respect to the table ML2PM. On the other hand, the following query is not a sound translation of query $\mathcal{Q}_1$.

$\mathcal{Q}_4$: ***select*** ***
from *PubMed*
where *kw = "APH"* ***OR*** *kw = "p75 ICD"*

To see this, consider the PubMed relation instance in Figure 1(b). Its third tuple satisfies $\mathcal{Q}_4$ but it cannot be associated, through ML2PM, to any MedLine article retrieved by $\mathcal{Q}_1$.

Another observation is that the above definition is not symmetric. Also, note that sound translations are not unique.

Example 6 *While our previous example shows that $\mathcal{Q}_4$ is not a sound translation of $\mathcal{Q}_1$, notice that $\mathcal{Q}_1$ is a sound translation of $\mathcal{Q}_4$, with respect to table PM2ML (which is the inverse of the ML2PM). Tuples retrieved by $\mathcal{Q}_1$, from every possible instance of MedLine, correspond to articles mentioning protein OPH which, through PM2ML, can be associated with PubMed articles mentioning protein APH. Concerning the uniqueness of sound translations, remember from our motivating example that both queries $\mathcal{Q}_2$ and $\mathcal{Q}_3$ are sound translations of query $\mathcal{Q}_1$.*

Since one sound translation might retrieve more data than another, we consider next the notion of *completeness*. That is, whether there is a sound translation that retrieves from remote peers all possible sound data.

Definition 7 *Given queries q_1, q_2 over peers P_1 and P_2, respectively, we say that q_2 is a* complete translation *of query q_1 with respect to mapping table m, if $q_1 \overset{m}{\mapsto} q_2$ and for every query q_2' over P_2 such that $q_1 \overset{m}{\mapsto} q_2'$ and every instance r_2 of P_2, $q_2(r_2) \supseteq q_2'(r_2)$.*

Notice that, by definition, if two queries q_2 and q_2' are complete translations of a query q_1, then q_2 and q_2' are equivalent.

Example 8 *Consider query $\mathcal{Q}_1$ from our motivating example and its sound translations, namely, $\mathcal{Q}_2$ and $\mathcal{Q}_3$. We claim, without providing a formal proof, that query $\mathcal{Q}_2$ is a complete translation of $\mathcal{Q}_1$, with respect to mapping table ML2PM.*

We are now in a position to formally characterize the relationship between the component queries of a global query. Specifically, we require that for each pair q_i, q_j of component queries such that $dep(q_j) = q_i$, query q_j is a sound translation of query q_i.

Our definitions assume that a single mapping table maps all the attributes in the relations involved. We relax this assumption in Section 5.5. We also note that so far we have only considered the selection and join operators. We investigate the issues concerning the projection operator next.

4.3 The projection operator

Sound translations guarantee that only correct tuples are retrieved from remote peers. However, not all the attributes from the remote peers are necessarily of interest. The user has the ability, through the projection operator, to express what local attributes are of interest to her and, thus, we provide a similar mechanism for the data retrieved from remote peers.

Our solutions make use of attribute correspondences which associate attributes in different peers. Learning attribute correspondences is a main component of schema matchers [16]. An attribute correspondence for attributes requiring no data translation can be encoded by a simple mapping table with the identity mapping. This is the situation depicted in Figure 1(e). In general, a mapping table $m[X \cup Y]$ encodes, in addition to the set of data associations, an attribute correspondence between the set of attributes X and Y.

Definition 9 *Let P and P' be peers exposing set of attributes U and U', respectively, and $m[X \cup Y]$ be a mapping table such that $X \subseteq U$ and $Y \subseteq U'$. Then, m is* relevant *to a query q over P, if $att(q) \subseteq X$.*

Hence, when translating a query q that includes a projection on attributes $att(q)$ we make use of all the relevant mapping tables $m_1[X_1 \cup Y_1]$, $m_2[X_2 \cup Y_2]$, $\ldots m_k[X_k \cup Y_k]$, and the translated query returns the union of all the Y_i's in these tables.

Example 10 *Consider the query that retrieves from MedLine all the protein names mentioned in articles published in 1998:*

$\mathcal{Q}_5$: ***select*** *keyword*
from *MedLine*
where *year = "1998"*

The complete translation of $\mathcal{Q}_5$, with respect to mapping table year2py shown in Figure 1(e), is:

$\mathcal{Q}_6$: ***select*** *kw*
from *PubMed*
where *py = "1998"*

There are two points to make here. First, due to mapping table year2py, our projection is on the kw attributes of the retrieved PubMed tuples. Second, it is possible that the latter query retrieves PubMed articles that violate mapping table keyword2kw shown in Figure 1(c). However, this is consistent with our query semantics since soundness is defined here only with respect to the year2py mapping table.

5 Algorithms

In general, queries are initially expressed in a query language (e.g., relational algebra, SQL) and are later transformed in some appropriate *internal* representation. Before we discuss the issue of querying, we need to fix these two parameters, i.e., the query language and the representation used.

We focus here on queries that are expressed in $S{+}J$ algebra. An $S{+}J$ query uses the operators of selection and join. The selection formula is *positive*, i.e., it has no negation and it consists of conjunctions and disjunctions of atoms of the form $(A = B)$ and $(A = a)$, where A and B are attribute names and a is a constant. Note that projection is supported in our framework but is handled independently.

Common query representations include tableau, which is a tabular representation of a query which resembles a database instance, and query trees, which is a graph-like representation of a query [3]. In this work, the tabular representation is the preferred choice. One reason for this is uniformity. Notice that we already use a similar representation, namely, mapping tables, to address issues of heterogeneity among different peers. In the following paragraphs, we introduce T-queries which is a tabular representation of queries and we show that for each $S{+}J$ query we can have an equivalent T-query. Then, we show how T-queries can be used to test whether a query q' is a sound translation of query q. Finally, we show how T-queries can be used to compute sound and complete query translations.

5.1 T-queries

We start by defining T-queries over one relation. Thus, we only consider selections. We later show how our definitions are extended to consider queries over multiple relations, thus taking into account joins. The following paragraph presents the syntax and semantics of T-queries.

A T-query q_T over relation schema $R[U]$ is a table T with attributes U where each variable appears in at most one row. Intuitively, one can think of each $t \in T$ as a tableau query whose corresponding tableau only has a single tuple. Then, T represents a set of tableau queries. Given a T-query q_T over schema R and an instance r of R, the result of executing q_T on r, denoted as $q_T(r)$ is:

$$q_T(r) = \{\rho(t) \mid \rho \text{ is a valuation of } t \in T \text{ and } \rho(t) \in r\}.$$

Example 11 *Consider the following query that returns all articles from PubMed mentioning proteins APH or p75 ICD:*

$\mathcal{Q}_7$: ***select*** ***
from *PubMed*
where *kw = "APH"* ***OR*** *kw = "p75 ICD"*

Then, the corresponding T-query is shown below:

paper_id	*kw*	*py*	*pm*
$\mathcal{X}_1$	*APH*	$\mathcal{Y}_1$	$\mathcal{Z}_1$
$\mathcal{X}_2$	*p75 ICD*	$\mathcal{Y}_2$	$\mathcal{Z}_2$

Proposition 12 *For any S+ query q over a relation $R[U]$ there is an equivalent T-query q_T, and vice versa.*

To construct query q_T from q, first we have to transform q into an equivalent query q' of the form $\sigma_E(R)$, where E is in disjunctive normal form. T-query q_T is of size linear in the size of q', which, in turn, has a size which is exponential, in general, with respect to the size of the initial query q. However, this is not a problem in practice since large selection formulas rarely occur.

We extend the definition of T-queries over multiple relations in the following way. Let $R = \{R_1[U_1], R_2[U_2], \ldots, R_n[U_n]\}$ be a relational schema and U be $\cup_{i=1}^{n} U_i$. A T-query q_T over R is a table T with attributes U where each variable appears in at most one row. In terms of semantics, consider a T-query q_T over R and an instance $r = \{r_1, r_2, \ldots, r_n\}$ of R. Then the result of executing q_T on r, denoted by $q_T(r)$, is:

$$q_T(r) = \{\rho(t) \mid \rho \text{ is a valuation of } t \in T \text{ and } \rho(t) \in r_1 \bowtie \cdots \bowtie r_n\}.$$

Proposition 13 *For any S+J query q over a relational schema R, there is an equivalent T-query q_T, and vice versa.*

We conclude this subsection by presenting the notion of *join* between T-queries. This will play a central role in all the algorithms presented in the following subsections. Let T_1 and T_2 be the tables of T-queries q_T^1 and q_T^2 with attributes U_1 and U_2, respectively. Attribute sets U_1 and U_2 are not necessarily disjoint. Then $T_1 \overset{\text{var}}{\bowtie} T_2$ is a T-query with attributes $U_1 \cup U_2$ defined as follows. Recall that

a substitution is a function that maps only variables to either variables or constants. For every $t_1 \in T_1$ and $t_2 \in T_2$, find substitutions θ_1 and θ_2 for the variables of t_1 and t_2, respectively, such that $\theta_1(t_1[U_1 \cap U_2]) = \theta_2(t_2[U_1 \cap U_2])$. Furthermore, we require that for any other pair of substitutions θ_1' and θ_2' satisfying this condition, θ_1 is as general as θ_1' and θ_2 is as general as θ_2', that is, there exist substitutions γ_1 and γ_2 such that $\gamma_1 \circ \theta_1 = \theta_1'$ and $\gamma_2 \circ \theta_2 = \theta_2'$ (this corresponds to the notion of *most general unifier* used in logic programming [13]). If substitutions θ_1 and θ_2 exist, then add to $T_1 \overset{\text{var}}{\bowtie} T_2$ a $U_1 \cup U_2$-tuple t defined as: $t[U_1] = \theta_1(t_1)$ and $t[U_2] = \theta_2(t_2)$.

Intuitively, given relation instances r_1 and r_2 over U_1 and U_2, respectively, the join of T_1 and T_2 gives us a new T-query q_T such that:

$$q_T(r_1 \bowtie r_2) = q_T^1(r_1) \bowtie q_T^2(r_2).$$

5.2 Algorithms for sound translations

In the previous section, we introduced sound translations and we provided a definition through which we can test, given two queries q and q' over peers P and P', respectively, whether query q' is a sound translation of q with respect to a mapping table m. The definition of sound translations, however, does not provide us with a practical way to do the testing. In the next paragraphs, we show that one of the benefits of representing queries as T-queries is that we are able to perform such a test efficiently.

In brief, the proposed algorithm accepts as input two queries q and q' over peers P and P', respectively, and a mapping table m between the sets of attributes U and U' exposed by these peers. Initially, it converts both q and q' to their corresponding T-queries, say $q_T = T$ and $q_T' = T'$, respectively. Then the algorithm uses mapping table m to constrain the association of query disjunctions. Formally, given T, m and T', the algorithm constructs a T-query q_T^c with attributes $att(q')$ defined as $\pi_{att(q')}(T \overset{\text{var}}{\bowtie} m \overset{\text{var}}{\bowtie} T')$. We note that it is possible to perform this join since mapping tables and T-queries use the same syntax. In the last step, the algorithm checks whether q_T^c is *equivalent* to q_T', that is, for every instance r' of peer P', $q_T^c(r') = q_T'(r')$. To perform such a test we use an algorithm that checks for containment of union of conjunctives queries [17]. If q_T^c is equivalent to q_T', then the algorithm outputs *yes*.

Theorem 14 *The above algorithm outputs* yes *on input (q, m, q') if and only if q' is a sound translation of q with respect to mapping table m.*

The following proposition is used in the proof of this theorem. It also shows that the previous algorithm only needs to check whether $q_T' \subseteq q_T^c$ to verify whether q' is a sound translation of q.

article_id	*keyword*	*year*	*paper_id*	*kw*	*py*	*pm*
$\mathcal{X}_1$	OPH	1998	$\mathcal{X}_2$	APH	1998	$\mathcal{Y}_2$
$\mathcal{X}_3$	OPH	1998	$\mathcal{X}_4$	AARE	1998	$\mathcal{Y}_4$
$\mathcal{X}_5$	OPH	2003	$\mathcal{X}_6$	APH	2003	$\mathcal{Y}_6$
$\mathcal{X}_7$	OPH	2003	$\mathcal{X}_8$	AARE	2003	$\mathcal{Y}_8$

Figure 4: Storing query translations

Proposition 15 *The queries q_T' and q_T^c computed by the above algorithm on input (q, m, q') are such that $q_T^c \subseteq q_T'$.*

Finally, we establish the exact complexity of our problem.

Theorem 16 *The problem of testing whether a query is a sound translation of another query is Π_2^p-complete.*

5.3 Computing complete translations

Here we describe an algorithm that, given a query q and a mapping table m, computes a query q' such that q' is a sound and complete translation of q with respect to mapping table m. The algorithm extends the algorithm for testing sound translations. Let P and P' be two peers that expose attributes U and U', respectively, and assume that q is a query over P and m is mapping table between the set of attributes U and U'. The algorithm begins by converting query q to its corresponding T-query $q_T = T$. Then, it considers mapping table m and computes T-query $q_T' = \pi_{U'}(T \overset{\text{var}}{\bowtie} m)$. Finally, the algorithm outputs the query q' represented by q_T'. The following theorem shows that q' is a sound and complete translation of query q.

Theorem 17 *Query q' computed by the above algorithm on input (q, m) is a complete translation of q.*

5.4 Composing translations

In this section, we show the benefits of using a common formalism for representing both data and query associations. In more detail, we show that the algorithms that were created for inference of mapping tables can be used to effectively perform query composition. Consider the point in time after a complete translation has been computed. This translation may be stored within a mapping table. An example is shown in Figure 4. The query on the left of the table retrieves MedLine articles that mention protein OPH and were published in 1998 or 2003.

The query on the right represents a complete translation on PubMed. Notice that each tuple in the mapping table pairs a query with a sound translation of the query. Such storage permits us to reuse the data association inference algorithm of our earlier work [11] to compose query translations. Consider a path $\theta = P_1, P_2, ..., P_n$ of peers with a set of mapping tables m_i storing data associations between peers P_i and P_{i+1}, for $1 \leq i \leq n-1$. Now let T_i be a mapping table containing pairs of sound query translations (that is, each tuple (q, q') in the mapping table represents a T-query q on P_i and a sound translation q' of q with respect to the mapping table m_i).

Let m denote the mapping table that results from our inference algorithm [11] over the path θ and the set of mapping tables m_i. And let T denote the mapping table that results from our inference algorithm over the path θ and the set of (query) mapping tables T_i. Then, each tuple (q, q') in T contains a T-query q' that is a sound translation of q with respect to the mapping table m.

5.5 Using multiple mapping tables

The algorithms presented in the previous sections assume the existence of a single mapping table that maps all the attributes in the relations involved. In real life, we expect that multiple mapping tables are provided and that some attributes are not mapped. In what follows, we investigate how to handle these two situations.

Assume that instead of a single mapping table m, we are given a set of mapping tables M to use during the computation of a complete translation. The exact way in which these tables are combined can be either pre-specified or it can be left to the user. Here, we propose a technique for combining multiple tables automatically. The following example illustrates that combining all available tables during the computation of translations might yield counter-intuitive translations.

Example 18 *Consider query $\mathcal{Q}_1$ from our motivating example that retrieves articles from MedLine mentioning protein OPH. Assume that instead of just using mapping table keyword2kw, in Figure 1(c), we consider both mapping tables keyword2kw and id2id to compute the translation. Furthermore, assume that every retrieved tuple from PubMed must be associated with a local OPH article with respect to both mapping tables. Then, the resulting translation is equivalent to the following query:*

$\mathcal{Q}_8$: ***select*** ***
from *PubMed*
where *(kw = "APH"* ***OR*** *kw = "AARE")*
AND *paper_id = "10719179"*

That is, by using both mapping tables, we are forced to restrict the identifiers of the retrieved articles. Since no restriction is imposed in the article_id attribute of the MedLine retrieved articles, a similar reasoning should be followed when translating this query for the PubMed articles. This reasoning supports not using mapping table id2id, in Figure 1(d), in translating this query.

The proposed technique accepts as input a query q over a peer P and a set of mapping tables M from P to a second peer P', and it uses the set M to compute a complete translation q' of q. Initially, the algorithm converts q into its equivalent disjunctive normal form. Then, it proceeds by considering each disjunct D_j of q in isolation. For each disjunct D_j, it selects a table for the translation, if this table's local attributes participate in an atom of the disjunct. Call M_j the set of selected mapping tables. If the set of attributes of D_j is contained in the set of attributes of M_j (M_j is relevant to D_j), the algorithm combines the mapping tables in M_j into a single mapping table m_j by using the $\wedge$-operator [11]. For the time being assume that m_j mentions all the attributes exposed by peers P and P'. Then, the algorithm considers D_j as a T-query T_j containing only one row and computes a sound and complete translation of T_j with respect to mapping table m_j. Then the translated T-query is converted into an equivalent relational algebra expression whose selection formula becomes a disjunct in the resulting query q'.

Example 19 *By using the algorithm described here, query $\mathcal{Q}_1$ is translated to query $\mathcal{Q}_2$ which is indeed its complete translation. As another example, assume that $\mathcal{Q}_9$ is a query retrieving information from MedLine about articles that mention protein OPH and were published in 1999:*

$\mathcal{Q}_9$: ***select*** ***
from *MedLine*
where *keyword = "OPH"* ***AND*** *year = "1999"*

Then, our algorithm selects only mapping tables year2py and keyword2kw for the translation, resulting in the following query:

$\mathcal{Q}_{10}$: ***select*** ***
from *PubMed*
where *(kw = "APH"* ***OR*** *kw = "AARE")*
AND *py = "1999"*

If any of the mapping tables m_j computed by the above algorithm does not mention all the attributes exposed by P and P', then it is extended to a mapping table m'_j that maps the extra attributes to *any* value. For example, mapping table *ML2PM* in Figure 3 is the extension of the following mapping table

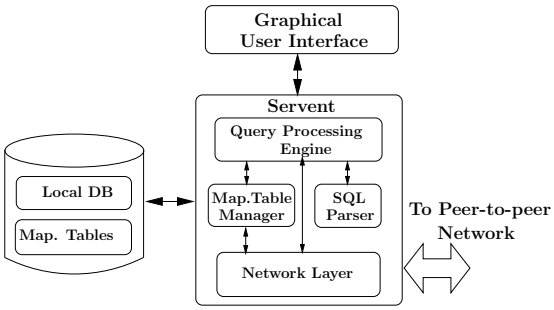

Figure 5: The architecture of a peer

keyword	*kw*
OPH	APH
OPH	AARE

to the set of attributes {*article_id*, *keyword*, *year*, *paper_id*, *kw*, *py*, *pm*}.

6 Implementation

We implemented our ideas over the prototype implementation of the Hyperion peer-to-peer data management system [4]. The structure of each peer in this system is shown in Figure 5. We provide a graphical user interface through which a user can set up acquaintances with new peers or pose local queries. Each peer in the system manages its own collection of data and it autonomously chooses a logical design and physical organization for the data. We use the MySQL relational DBMS to store both the peer data and any possible mapping tables that each peer might maintain.

The servent, which is the main component of this architecture, consists of four main modules. From these four modules, the query processing engine is the main focus of this work. It includes the implementation of the algorithms that convert SQL queries to their disjunctive normal form; convert SQL queries to their equivalent T-queries and back; and compute the complete translation of a query given a mapping table. Furthermore, it includes the algorithm for testing containment of T-queries and the algorithm to test whether a query is a sound translation of another one, with respect to a given mapping table. Finally, we also implemented the algorithm that, given a query q and a set of mapping tables M, selects the set of mapping tables to use in order to compute a complete translation.

We implemented a number of optimizations to improve the efficiency of our algorithms. One such optimization relies on the fact that our representation of queries as T-queries allows us to store in the database both the query itself and the relationship with its sound translations. As an example, consider again the stored translation shown in Figure 4. Assume that the calculation of this translation happened some time in the past, but the system stores this relationship between the two T-queries in the database. Now, assume that a new query is issued on MedLine asking only for articles that mention protein OPH and were published in 1998. At this point, we could run the optimized version of the algorithm for computing complete translations in order to retrieve the corresponding PubMed articles. Alternatively, one can use the algorithm for T-query containment to conclude that the correspondence between T-queries in Figure 4 can be used to compute the translation. In more detail, we test whether the T-query representation of the current query is contained in the left part of the table in the figure. Since this is the case in our running example, we treat the table in the figure as a mapping table and we use it to compute the translation of the current query. In this example, this computation will result in the selection of the right parts from the first two tuples.

An interesting application of the previous optimization relies on the observation that in peer-to-peer systems users are often satisfied with answers that are not complete, as long as they are given the guarantee that anything that is retrieved is correct. With this in mind, even a query that retrieves all articles in MedLine mentioning proteins OPH or NGF receptor can be answered satisfactorily from the PubMed peer by just retrieving articles that mention these proteins and were published in some particular time interval (for example, the last 5 years). Testing for T-query containment is also central in this approach since the stored query is contained in the query being posed.

We conclude our overview by noting that the implementation of the query processing engine was done in the C programming language and it contains approximately four thousand lines of source code.

7 Experiments

To evaluate our algorithms, we undertook two studies. The objective of our first study is to investigate the performance of our translation algorithms with respect to three problem parameters, namely, the size of the input query, the size of the output query, and the size of the mapping tables used in the translation. The second study investigates the performance of our algorithms under large query load and examines the benefits of storing and re-using past translations. Due to lack of space, our second study is only available in the extended version of the paper [10]. The data used by both studies are *real* and are extracted from publicly available sources. We use these data to create distinct peers, one per

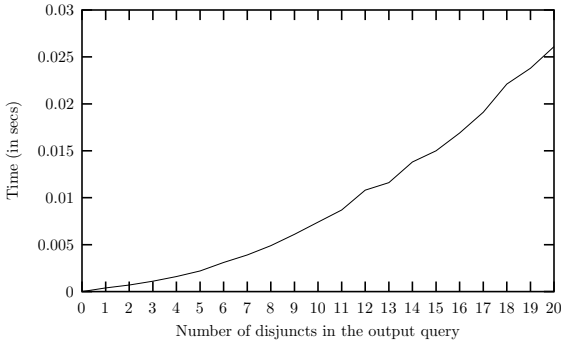

Figure 6: Generating large output queries

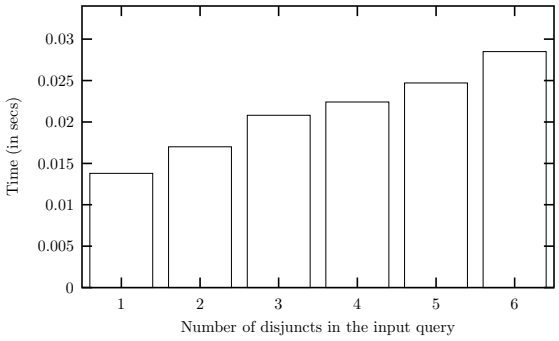

Figure 7: Translating large input queries

machine within the same LAN, and each peer has the structure shown in Figure 5.

7.1 Biological Databases

The data for this study belong to the MedLine, PubMed and SwissProt [2] databases. We populated each of the MedLine and PubMed relations with approximately 25000 tuples corresponding to protein-related articles. We assumed that MedLine articles are indexed, through the keyword attribute, by the currently approved protein name while PubMed articles are indexed by aliases of the approved names. Mapping table *id2id* had approximately 24000 tuples while mapping table *keyword2kw* had approximately 12000 tuples. The former table maps an MedLine article to at most one PubMed article while the latter table maps currently approved protein names to their corresponding aliases. Both mapping tables were retrieved from SwissProt.

Given query $\mathcal{Q}_1$ and mapping table *keyword2kw*, we expect that the time to translate $\mathcal{Q}_1$ is influenced by the number of values that OPH is associated with, since this number influences the number of disjuncts, and thus the size, of the output query. Thus, the objective of our first experiment is to investigate the exact relationship between the time to perform a query translation and the size, in terms of disjuncts, of the translated query. For this purpose, we select 20 distinct input queries each of which is similar, in spirit, to query $\mathcal{Q}_1$, i.e., it retrieves MedLine articles for a particular keyword/protein. The queries were selected in such a manner that the first query refers to a protein with a single alias in *keyword2kw*, the second query refers to a protein with 2 aliases, and so on, while the last query refers to a protein with 20 aliases. Figure 6 shows the translation times (in seconds) for each of these queries. As we can see, the translation time scales gracefully and, even for large output queries, it is still fractions of a second.

In the previous experiment, all the input queries have only a single disjunct. In this experiment, we vary the number of disjuncts in the input query and we investigate how this influences the translation time. We start by selecting 20 distinct input queries each of which, once translated, results in an output query with 14 disjuncts. From the 20 input queries, 4 queries have 2 disjuncts, 4 have 3 disjuncts, and so on, and the last 4 queries have 6 disjuncts. We consider 4 alternative queries for the same number of disjuncts since, given the number of disjuncts in a query, there are different combinations with which these disjuncts can contribute to result in 14 output disjuncts. For example, for a query with just two disjuncts, each of the two input disjuncts can be translated to 7 output disjuncts, or alternatively, the first input disjunct can result in 10 output disjuncts while the second input disjunct can result in the remaining 4. In Figure 7, we average the translation times of input queries with the same number of disjuncts and we also report, in the first column, the translation of an input query with a single disjunct. Notice that as the number of disjuncts in the input query increases, there is a corresponding increase in the translation time of the query. Although there is a correlation between these two quantities, our next experiment shows that there is another factor that also comes into play during the translation process.

For this experiment, we use the input query that had the worst performance, in terms of time, in our previous experiment. This is the input query with 6 disjuncts, denoted with D_1 to D_6, where disjunct D_1, once translated, results in 9 output disjuncts while the remaining 5 input disjuncts all result in a single output disjunct. Notice again that the number of disjuncts in the output query is 14. During this experiment, we translate this input query 6 times and the only difference between the translations is the order with which we translated the 6 input disjuncts. In particular, in the first run, the input disjunct D_1 is considered first for translation, while in the second run, disjunct D_1 is considered second in order for translation. Continuing in this fashion, in the sixth run, the five single-output disjuncts are

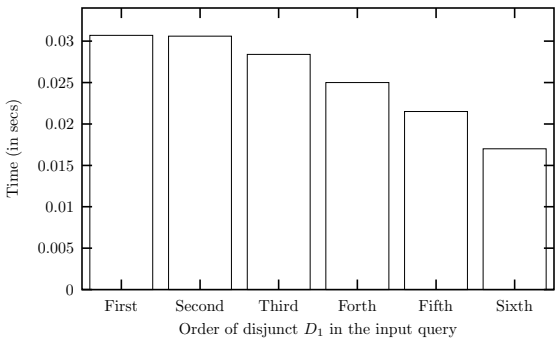

Figure 8: Translating a large input query

considered first, while disjunct D_1 is considered last. Figure 8 show the translation time for each of the six runs of the algorithm. The drop in translation time is due to the following reason. After a disjunct of the input query is translated, the algorithm checks whether any of generated output disjuncts is already part of the output query due to a previously translated input disjunct. The objective of this check is to avoid duplicate disjuncts in the output query, or pairs of disjuncts where one is contained in another. However, the check generates more comparisons, and thus more computation, if a large number of output disjuncts is generated early in the translation process. Hence, delaying the translation of a disjunct like D_1 causes reduced translation times. In general, reordering of input disjuncts seems beneficial and it can be achieved by storing frequency information about the values of a mapping table. Using these frequencies we can estimate the number of output disjuncts for each input disjunct.

We also experimented with varying the mapping table sizes. Our experiments show that this parameter does not influence the translation time. This is because we do not scan the whole table in order to locate the tuples to be used in the translation, but we use in-memory hash indexes. Our implementation of the hash indexes is customized to the semantics of mapping tables and thus takes into consideration the existence of variables in the tables.

8 Conclusions

We have considered the problem of data sharing between autonomous data sources. We used mapping tables to associate data from different sources and we have shown how the tables can be used in the translation of structured queries. We introduced the notions of sound and complete translations and we proposed algorithms to compute such translations and an algorithm to test if a query is a sound translation of another. We implemented our algorithms and we have presented experiments which show that these can be used in practice.

Our future work investigates algorithm optimizations along with support for more expressive query languages that include, for example, negation.

References

[1] PubMed/MedLine. http://www.ncbi.nlm.nig.gov/entrez/.

[2] SwissProt. http://www.ebi.ac.uk/swissprot/.

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

[4] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. J. Miller, and J. Mylopoulos. The Hyperion Project: From Data Integration to Data Coordination. *SIGMOD Record*, 32(3):53–58, 2003.

[5] C.-C. K. Chang and H. Garcia-Molina. Mind your vocabulary: Query mapping across heterogeneous information sources. In *SIGMOD*, pages 335–346, 1999.

[6] R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, pages 207–224, 2003.

[7] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management Systems. In *ICDE*, 2003.

[8] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica. Complex Queries in DHT-Based P2P Networks. In *IPTPS*, 2002.

[9] R. Huebsch, J. M. Hellerstein, N. L. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *VLDB*, 2003.

[10] A. Kementsietsidis and M. Arenas. Data sharing through query translation in autonomous sources. Technical Report CSRG-491, CS Dept., University of Toronto, 2004.

[11] A. Kementsietsidis, M. Arenas, and R. J. Miller. Data mapping in p2p systems: Semantics and algorithmic issues. In *SIGMOD*, pages 325–336, 2003.

[12] M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, 2002.

[13] J. W. Lloyd. *Foundations of logic programming*. Springer-Verlag New York, Inc., 1987.

[14] J. Madhavan and A. Y. Halevy. Composing Mappings Among Data Sources. In *VLDB*, pages 572–583, 2003.

[15] 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.

[16] E. Rahm and P. A. Bernstein. On Matching Schemas Automatically. *The VLDB Journal*, 10(4):334–350, Dec. 2001.

[17] Y. Sagiv and M. Yannakakis. Equivalences Among Relational Expressions with the Union and Difference Operators. *JACM*, 27(4):633–655, 1980.