Relational Peer Data Sharing Settings and Consistent Query Answers

Mehedi Masud
Department of Computer Science
College of Computers and Information Technology
Taif University, Taif, Saudi Arabia
Email: mmasud@tu.edu.sa

Sultan Aljahdali
Department of Computer Science
College of Computers and Information Technology
Taif University, Taif, Saudi Arabia
Email: aljahdali@tu.edu.sa

Abstract— **In this paper, we study the problem of consistent query answering in peer data sharing systems. In a peer data sharing system, databases in peers are designed and administered autonomously and acquaintances between peers are established thorough data sharing constraints. Since data are managed in peer databases autonomously, peers' data may be inconsistent with respect to the data sharing constraints. Inconsistencies of data may also result due to the change of constraints (e.g. adding, modifying, and deleting constraints between peers). In order to solve inconsistencies between peers, one possible solution could be modify data physically in inconsistent peers by propagating updates. However, it is not always feasible to change data physically in peers since peers are autonomous and a peer may not have permission to modify other peers' data. Considering the possible inconsistent situations, this paper presents a semantics of obtaining consistent answers in a peer data sharing system such that the answers are consistent wrt to data sharing constraints. Consistent answers are obtained at query time by avoiding the inconsistent data. The paper also shows a method to achieve consistent answers of a query.**

Keywords-database; consistent query; data sharing;

I. INTRODUCTION

Typical data integration [5] and data exchange settings [12], [14], [15], we are often dealing with *one world*, for example, a set of sources all containing information about videos or music files. Hence, data integration or data exchange between data sources is provided mainly through the use of views or schema mappings, i.e., queries that map and restructure data between schemas. However, in data sharing systems [3], [4], [8] sources may represent different worlds with different schemas and data vocabularies, and the real world entities denoted by different symbols in different sources may be semantically related [7]. In order to share data between peers in a data sharing system, semantic relationships are created. For creating semantic relationships, a possible solution is to create a domain relation [8] through value correspondences or value-level mappings [1] between sources since data vocabularies are different in peers. The value-level mappings map the data elements of a source domain to the data elements of another source domain. Once the semantic relationships are established, answers to a query posed to a peer in the system is obtained by translating the query in terms of the vocabularies of the semantically related peers and propagating and executing the query into the peers.

Authors in [3], [4] introduced a value-mapped peer data sharing system, called Hyperion, considering the value-level mappings between peers. The value-level mappings are created by the use of mapping tables [1], which map corresponding data values, logically tuples, that reside in different peers. Mapping tables also bridge the differences of data vocabularies as well as impose data sharing constraints between peers. Query answering mechanism in such a system specifies a query posed to a peer's instance is applied to the peer's local database instance and the query is propagated to all other related peers for accumulating related data. Authors in [2] proposed a query translation mechanism between peers in a value-mapped peer data sharing system, however, they did not consider consistent query answering in the system.

In this paper, we investigate a consistent query answering mechanism and present a semantics of obtaining consistent answers in a value-mapped peer data sharing system. We also present a technique for obtaining consistent answers. Note that data in peers may be inconsistent wrt the mappings, meanwhile, peers are consistent wrt their local integrity constraints. Moreover, data and mappings may be updated in peers in course of time which can make the data in related peers inconsistent wrt the new mappings, although, peers were consistent before the occurrence of the updates. Since peers are autonomous in peer data sharing systems, it is not feasible to modify the data and mappings physically in related peers to solve inconsistencies between peers. Moreover, a peer may not have permission to modify other peers' data. Considering these inconsistent situations, we present a semantics of query answering that characterize what are the intended and correct answers to a query posed to and answered by a value-mapped peer data sharing system. Basically, we adopt a semantics for consistent query answering in consistent databases that was originally introduced by Arenas et al. [9]. We extend the semantics to be used in a peer data sharing system that is characterized by defining a set of virtual global instances called solutions for a peer where a query originates. A solution

X'	Y'	W
a'_1	b'_1	d_1

(a) relation R_1 at P_1

X	Y	Z
a_1	b_1	c_1
a_2	b_1	c_2

(b) relation R_2 at P_2

Figure 1. Instances of peers P_1 and P_2

X'	X
a'_1	a_1

(a) Mapping table m_1

Y'	Y
b'_1	b_1

(b) Mapping table m_2

Figure 2. Mapping tables

for a peer is an intended global instance that respects the mappings with its acquainted peers. After having a definition of the intended solutions for a peer, the consistent answers to a query in a peer are those that are *certain* [10] in every possible solution.

The paper is structured as follows. Section 2 presents the technical preliminaries and presents a model of data sharing systems. Section 3 explains the semantics of consistent answering of queries in data sharing systems. In Section 4, we show the process of obtaining consistent answers, and Section 5 discusses related work. Finally, the paper concludes with some future directions.

II. Technical Preliminaries and System Model

Attributes are symbols taken from a given finite set $U=\{A_1,\ldots,A_q\}$ called the universe. We use the letters A, B, C, … to denote single attributes and X, Y, … to denote sets of attributes. Each attribute A_j is associated with a set of values called the *domain* of A_j and is denoted by $dom(A_j)$. If $X=\{A_1,A_2,\ldots,A_k\}\subseteq U$, then $dom(X)\subseteq dom(A_1)\times dom(A_2)\times\ldots\times dom(A_k)$. A non-empty subset of U is called a *relation schema* R. A *database schema* is a finite collection $R = (R_1,\ldots,R_m)$ of relation schemas. A tuple t with attribute X is a X-value. A relation is a set of tuples. We shall use r_i to denote a relation that interpreters R_i. An *instance I* over *R* is a function that associates to each relation schema R_i a relation $r_i=I(R_i)$. For a tuple t and a set $Y \subseteq U$, we denote the restriction of t to Y by t[Y].

Data sharing constraints in data sharing systems impose constraints on data values by associating values between two sources, where databases may use different vocabularies and different domains. Intuitively, the associations of values create a *domain relation* between two semantically related domains and relate data objects (tuples) to be shared between peers. Authors in [1] show how to create data sharing constraints by using mapping tables. A mapping table, denoted m(X,Y), simply is a relation over the attributes X and Y. A tuple (x, y) in m(X,Y) indicates a mapping that the value $x \in dom(X)$ is associated with the value $y \in dom(Y)$. Formally, a mapping over a set of attributes U of $X \cup Y$, alternatively called a tuple *t*, in a mapping table represents that for each $A \in U$, t[A] is either a constant in dom(A), a variable in V or an expression of the form v−S, where $v \in V$ and S is a finite subset of dom(A) [1].

Note that a tuple in a mapping table may contain constants or variables. The variables are used to increase expressiveness power of mapping tables. Given the presence of variables in mappings, it is necessary to introduce the notion of a valuation. A valuation ρ over a mapping table m is a function that maps each constant value in m to itself and each variable v of m to the value in the intersection of the domains of the attributes where v appears [1]. Furthermore, if v appears in an expression of the form v-S, then $\rho(v) \notin S$. In general, if there are multiple mapping tables $M=\{m_1,m_2,\ldots,m_k\}$ then we can combine the tables into a single mapping table using the $\wedge$-operator [1]. Therefore, we can apply the valuation ρ on M which we represent as $\rho(m_1,m_2,\ldots,m_k)$. Since a mapping table m from X to Y associates values from dom(X) to dom(Y), we can determine the set of Y−values with which a particular value $x \in dom(X)$ is associated by the following definition.

Definition 1 *[Y−values] Let m be a mapping table from X to Y. We define Y−values, denoted as $Y_m(x)$, with which a particular value $x \in dom(X)$ is associated as follows: $Y_m(x)=\{y| \exists t \in m$ and there exists valuation ρ over m such that $\rho(t[X])=x$ and $\rho(t[Y])=y\}$*

We now explain how a mapping table creates valid associations of tuples between two relations. Consider relations r_1 and r_2 with relation schemas $R_1[U_1]$ and $R_2[U_2]$, respectively, and also consider a mapping table m(X,Y) from X to Y, where $X \subseteq U_1$ and $Y \subseteq U_2$. Consider a relation r_{12}, where $r_{12}=r_1\times r_2$. We say that a tuple t_{12} in r_{12} associates a tuple $t_1 \in r_1$ to a tuple $t_2 \in r_2$ if there is a valuation ρ over m such that $t_{12}[X] \in \pi_X(\rho(m))$ and $t_{12}[Y] \in \pi_Y(\sigma_{X=t12[X]}(\rho(m)))$. The intuition behind this association of tuples is that a tuple $t_1 \in r_1$ such that $t_1[X]=x$ is associated with respect to the mapping table m(X,Y), only the tuples $t_2 \in r_2$ for which $t_2[Y] \in Y_m(x)$. Therefore, we can consider the mapping table m as a condition to filter relation r_{12} that is subset of r_{12} contains only the tuples that m associates the tuples of relations r_1 and r_2.

Now we show how a mapping table associates data values between two peers P_i and P_j. Assume two relations r_i and r_j in peers P_i and P_j with schemas $R_i[U_i]$ and $R_j[U_j]$, respectively and a mapping table m(X,Y) over the attributes $X \subseteq U_i$ and $Y \subseteq U_j$. A mapping (x, y) in m indicates that a tuple $t \in r_i$ such

that t[X]=x is associated with a tuple t′ ∈ r_j such that t′[Y]=y. Considering the existence of mapping tables between peers, we now define the data sharing constraints between peers.

Definition 2 *[Data Sharing Constraint] A data sharing constraint Σ_{ij} between two peers P_i and P_j in a value-mapped peer data sharing system is a set of mapping tables $\{m_1, m_2, \ldots, m_k\}$. Existence of a mapping table $m_i \in \Sigma_{ij}$ between peers P_i and P_j has the following logical implication:*
$\forall XY\ (m_i(X,Y) \rightarrow \exists ZR(X,Z) \wedge \exists WQ(Y,W))$
Here, R, Q are relations in $\mathbf{R_i}$ and $\mathbf{R_j}$ of P_i and P_j, respectively.

Example 1 *Consider Figure where two peers P_1 and P_2 with relational schemas $\mathbf{R_1}=\{R_1(X',Y',W)\}$, $\mathbf{R_2}=\{R_2(X,Y,Z)\}$. Assume that P_1 is connected to peer P_2 by the data sharing constraint $\Sigma_{12}=\{m_1(X',X), m_2(Y',Y)\}$. The first mapping table expresses that a tuple t′ in R_2 is related to a particular tuple t in R_1 wrt m_1 such that $t[X'] \in \pi_{X'}(\rho(m_1))$ and $t'[X] \in Y_{m1}(t[X'])$. Hence, from Figure 1, we observe that the tuple $t=(a_1',b_1',d_1)$ is related to the tuple $t'=(a_1,b_1,c_1)$ wrt m_1. Moreover, $t=(a_1',b_1',d_1)$ is related to the tuples $t'_1=(a_1,b_1,c_1)$ and $t'_2=(a_2,b_1,c_2)$ wrt m_2. However, if we count both m_1 and m_2, tuple t is only related to the tuple t'_1 since only for t we have $t[X',Y'] \in \pi_{X',Y'}(\rho(m_1,m_2))$ and $t'[X] \in Y_{m1}(t[X'])$ and $t'[Y] \in Y_{m2}(t[Y'])$.*

A. Data Sharing System

A value-mapped peer data sharing system is a set $P = \{P_1, P_2, \ldots, P_n\}$ of n peers with autonomous pre-existing local database systems (LDBSs). Each peer P_i, $1 \le i \le n$, has its own database denoted D_i with its own schema R_i. We assume that each peer P_i is responsible for maintaining its database consistent with respect to its local integrity constraints, independently from other peers.

We now define the notions of a peer and a peer data sharing system.

Definition 3 *[Peer] A peer P_i in a peer data sharing system N consists of:*

- *a database D_i with its own schema R_i.*
- *a set of local integrity constraints IC_i on D_i.*
- *a finite set of data sharing constraints from P_i to its neighbor P_j, denoted by Σ_{ij}. The set of data sharing constraints in P_i for all of its neighbors $P_j \in P$ is denoted by Σ_i, where $\Sigma_i = \cup_{Pj}\Sigma_{ij}$.*

Definition 4 *[Peer Data Sharing System] A peer data sharing system $N = < P, \Sigma >$ consists of:*

- *a finite set $P=\{P_1,P_2,\ldots,P_n\}$ of peers, and*
- *a set $\Sigma = \{\Sigma_1,\Sigma_2,\ldots,\Sigma_n\}$ of data sharing constraints.*

Since mappings are created pairwise in a peer data sharing system, logically, a semantic graph is created that we call an *acquaintance graph*. Formally, an acquaintance graph is a graph $\Gamma_N=(V, ACQ)$, where V is a set of vertices and $ACQ \subseteq P \times P$ is a set of direct edges such that every edge $(P_i, P_j) \in ACQ$, $i \neq j$, is associated with data sharing constraints Σ_{ij}. A peer P_j is called an *acquaintee* of a peer P_i in an acquaintance graph *N* if there exists an edge from P_i to P_j in Γ_N; $N(P_i)$ denotes the set of acquaintees of P_i. The notion of acquaintees represents the direct links between two peers. However, two peers in *N* may be linked indirectly by a path of peers. Therefore, from a specific peer a request can propagate to a set of peers that are related indirectly to that peer. The set of peers that are linked directly or indirectly to a peer are called accessible peers for that peer. Formally, a peer P_j is accessible from a peer P_i in *N* if there is a path in Γ_N from P_i to P_j; $A(P_i)$ denotes the set of accessible peers of P_i.

III. Consistent Query Answering

In a data sharing system, when a peer P_i receives a query from its users it answers the query both from its own data and the data stored at its acquaintees $P_j \in N(P_i)$. The decision by P_i on what data it receives from an acquaintee P_j depends on the data sharing constraints Σ_{ij}. When P_i receives data form an acquaintee P_j, P_i repairs its own data for solving inconsistencies wrt to the data sharing constraints Σ_{ij}. The formal definition of repair of a database D is the following.

Definition 5 *[Repair [9]] Let D be a database with integrity constraints IC. We say that a database D′ is a repair of D with respect to IC if:*

- *$D' \models IC$, and*
- *there is no repair database D″ such that $D'' \models IC$ and $\Delta(D,D'') \subset \Delta(D,D')$, where $\Delta(D,D')=(D-D') \cup (D'-D)$ is the symmetric differences (also called distance) between two databases D and D′.*

This repair concept is originally developed in the area of consistent query answering in an inconsistent database. The repair of an inconsistent database D can produce multiple solutions. The semantics of query answering is given in terms of consistent answers, which we present next.

Definition 6 *[Consistent Answers [9]] Let D be database and IC be a set of integrity constraints. Let q be a query over D. We say that a tuple t is a consistent answer to q wrt IC if t is an answer to query q in every repair D′ of D wrt IC.*

In a data sharing system, the repair of data at P_i with the data of its acquaintees $P_j \in N(P_i)$ creates a solution instance, called acquaintee solution, that satisfies the data sharing constraint Σ_{ij}.

Definition 7 *[Acquaintee Solution] Let P be a data sharing system. Consider a peer P_i in P with a database D_i and D′ is a*

database instance on schema $\cup_{Pj \in N(Pi)} R_j$. *Let D″ is a repair on database instance* $D_i \cup D'$ *wrt* $\cup_{Pj \in N(Pi)} \Sigma_{ij}$. *D″ is an acquaintee solution for* P_i *if: (a)* $D'' \models \cup_{Pj \in N(Pi)} \Sigma_{ij}$ *and (b) there is no instance D‴ that satisfies (a) and such that* $D''' \subset D''$.

Definition 8 *[Consistent Answer] Given a query q to a peer* P_i, *a ground tuple t is in consistent answer iff t is an answer to q in every possible acquaintee solution r wrt* Σ_i.

The definition of acquaintee solution for P_i may suggest that P_i can physically change data of other peers and if required modify its own data at query time, but this is not applicable in a peer to peer context. First, because sources are semiautomatic, and the cost in terms of human involvement may become prohibitive. Second, because sources are autonomous, and may therefore refuse to be modified the data. Actually, the notion of solution is used as an auxiliary notion to characterize the semantically correct answers from P_i's point of view. Ideally, P_i should be able to obtain consistent answers just by querying the already available local instance and instances of acquaintees which may be inconsistent wrt data sharing constraints.

There are mechanisms [11], [12], [13] for computing consistent answers in peer data exchange settings that avoid or minimize the physical generation of repairs. In this paper, we show how to achieve consistent answers of queries in a value-mapped data sharing system where peers are related with value-level constraints. In particular, we propose an approach such that, given a query q at a peer, it generates the consistent answers directly from peer databases in the system that possibly may be inconsistent wrt constraints.

Example 2 *Consider the setting in Example 2. Assume a query is posed to* P_1. *Therefore, it has to return results from its database instance and also from the instance of* P_2. *Note that the data that is received from* P_2 *must satisfy* Σ_{12}. *For checking satisfaction,* P_1 *will ask* P_2 *for its data. If* P_2 *has data sharing constraints with any other peer then* P_2 *will ask data from those peers. In this example, since* P_2 *has no data sharing constraints with other peers, it will return to* P_1 *its data. Now* P_1 *will resolve inconsistency wrt* Σ_{12}. *Here, the data in* P_1 *together with the data in* P_2 *do not satisfy the first mapping table. For instance, value* a_2 *of attribute X in the second tuple in* R_2 *does not satisfy* m_1, *although value* b_1 *maps with a data* b_1' *through mapping table* m_2. *In general, such an inconsistency could be solved by performing repairs. During the repair a peer can generate multiple solutions. A consistent answer is an answer that appears in every solution. Hence, in this example, we could have two solutions using repairs for making databases in* P_1 *and* P_2 *consistent wrt* Σ_{12}. *First, virtually adding* $< a_2', b_1', d_2 >$ *into* R_1 *and inserting* $< a_2', a_2 >$ *in* m_1. *Note that* a_2' *is a value in dom(X'). There could be another solution by removing* $< a_2, b_1, c_2 >$ *from* R_2. *In this case, there are two solutions (instance) wrt to* P_1. *The solutions are shown in Figure 3 and Figure 4.*

Now consider a query $q_1(X', Y', W)$: $R_1(X', b_1', W)$ *that is posed to* P_1. *In usual case, the answers to the query will be* $\{ < a_1', b_1', d_1 > \}$, $\{ < a_1, b_1, c_1 > , < a_2, b_1, c_2 > \}$. *The answer is produced form the answer of* P_1 *and* P_2. *The answer returned by* P_2 *is* $\{ < a_1, b_1, c_1 > , < a_2, b_1, c_2 > \}$. *Note that in order to execute the query* q_1 *at* P_2, *it must be transformed in terms of the vocabularies of* P_2, *which is* $q_2(X, Y, Z)$: $R_2(X, b_1, Z)$. *Authors in [2] show an algorithm for such a translation. However, we observe that the answer is not consistent wrt* Σ_{12} *since the tuple* $< a_2, b_1, c_2 >$ *does not satisfy* m_1 *in* Σ_{12}. *However, the expected consistent answer to the query would be* $\{ < a_1', b_1', d_1 > \}, \{ < a_1, b_1, c_1 > \}$, *since both the tuples exit in the solutions 1 and 2.*

Note that the results returned from peers are shown in different set due to different data vocabularies of peers.

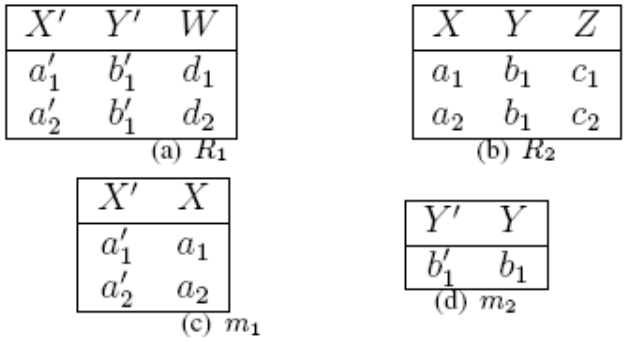

Figure 3. Solution 1

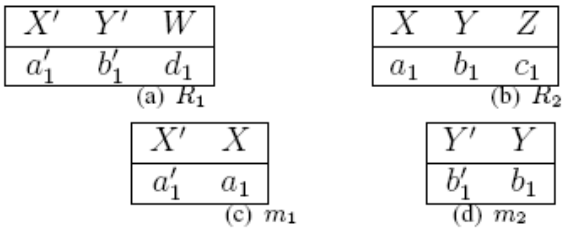

Figure 4. Solution 2

In static environments, data sources are populated only after the schemas and mappings have already been designed. Therefore, it is possible to guarantee that data remains consistent. However, in a dynamic environment, e.g. peer data sharing systems, data in sources are not static and mappings can be changed. Therefore, two sources may be inconsistent wrt the constraints between them. Hence, if a query is asked at a peer then what should be the answer wrt inconsistent state? In the following example, we describe the semantics of consistent query answering during the change of mappings between peers.

Example 3 *Consider a peer data sharing system with three peers P_1, P_2, and P_3 with schemas $R_1(A_1,B_1)$, $R_2(A_2,B_2)$, and $R_3(A_3,B_3)$, respectively. Consider the instances of the peers as follows:*
P_1: {$R_1(a_1,b_1)$, $R_1(a_3,b_3)$},
P_2:{$R_2(a_1^2,b_1^2)$, $R_2(a_2^2,b_1^2)$, $R_2(a_5^2,b_5^2)$},
P_3: {$R_3(a_1^3,b_3)$}
Also assume that the system has the data sharing constraints as follows:

$$\Sigma_{12}=\{m_1(A_1,A_2), m_2(B_1,B_2)\},\Sigma_{13} = \{m_3(A_1,A_3)\}.$$

Assume that the mapping tables are populated as follows: [{$m_1(a_1,a_1^2)$}, {$m_2(b_1,b_1^2)$}], [{$m_3(a_1,a_1^3)$}]
Considering the acquaintances, the global instance from P_1's view is as follows:
r = [{$R_1(a_1,b_1)$, $R_1(a_3,b_3)$}, {$R_2(a_1^2,b_1^2)$, $R_2(a_2^2,b_1^2)$, $R_2(a_5^2,b_5^2)$},{$R_3(a_1^3,b_3)$}]
The solutions from P_1's view are the solutions that satisfy Σ_{12} and Σ_{13}. Note that solutions are obtained by repairing r wrt the data sharing constraints Σ_{12} and Σ_{13}. Considering Σ_{12}, we obtain following two repairs:
r_1 = [{$R_1(a_1,b_1)$}, {$R_2(a_1^2,b_1^2)$},{$R_3(a_1^3,b_3)$}] and
r_2 = [{$R_1(a_1,b_1)$,$R_1(a,b_1)$}, {$R_2(a_1^2,b_1^2)$, $R_2(a_2^2,b_1^2)$}, {$R_3(a_1^3,b_3)$}]

In the first repair, tuples $R_1(a_3,b_3)$, $R_2(a_5^2,b_5^2)$ are removed from r since they do violate Σ_{12}. The tuple $R_2(a_2^2,b_1^2)$ is also removed since it does not satisfy first constraint in Σ_{12}. However, in the second repair, tuple $R_2(a_2^2,b_1^2)$ is considered since it satisfies the second constraint in Σ_{12}. For considering $R_2(a_2^2,b_1^2)$, we need to add a tuple $R_1(a, b_1)$ in R_1 and a mapping (a,a_2^2) in m_1. Note $a \in dom(A_1)$.

Now, we need to do repairs r_1 and r_2 wrt Σ_{13} (but keeping Σ_{12} satisfied). From r_1 we get only one repair as follows:
r_3 = [{$R_1(a_1,b_1)$}, {$R_2(a_1^2,b_1^2)$}, {$R_3(a_1^3,b_3)$}]

Similarly, from r_2, we get the following repair
r_4 = [{$R_1(a_1,b_1)$,$R_1(a,b_1)$}, {$R_2(a_1^2,b_1^2)$, $R_2(a_2^2,b_1^2)$},{$R_3(a_1^3,b_3)$}]

Therefore, with respect to Σ_{12} and Σ_{13}, we have two solutions {r_3, r_4} for peer P_1 considering its acquaintees.

Consider a query $q(A_1,B_1)$: $R_1(A_1,B_1)$ at P_1. The only consistent answer to the query q is [{$R_1(a_1,b_1)$}, {$R_2(a_1^2,b_1^2)$}, {$R_3(a_1^3,b_3)$}].

Now assume that a design decision has been made that the information between P_1 and P_3 should be synchronized corresponding to the attribute B_1 and B_3. Therefore, a new identity mapping table $m_4(B_1,B_3)$ has been introduced and added to Σ_{13}. We can represent the new constraint with the following formula.

$$m_3(A1,A3) \wedge R_1(A_1,B_1) \wedge R_3(A_3,B_3) \rightarrow B1=B3$$

Notice that P_1 and P_3 are now inconsistent wrt the new constraint. For instance, before adding mapping table m_4, tuple $< a_1, b_1 >$ in R_1 and $< a_1^3, b_3 >$ were related wrt m_3 although the tuples have different values in attributes B_1 and B_3. This is because, there is no constraint on the values of B_1 and B_3. But, after adding the new constraint, values of B_1 and B_3 should be equal but they have different values. If the previous query q is asked at P_1, then what should be the answer. The traditional approach to deal this situation is data cleaning [6]. Data cleaning techniques are often not applicable in our context. First, because sources are semiautomatic, and the cost in terms of human involvement may become prohibitive when the cleaning has to be done every time the constraint changes. Second, because sources are autonomous, and may therefore refuse to be "cleaned" just because of changes in the mappings. However, we achieve consistent answers without physically cleaning the sources using the notion of repairs as discussed before.

For example, if the query q is applied to P_1, q should return [{$R_1(a_1,b_1)$}, {$R_2(a_1^2,b_1^2)$}, {$R_3(a_1^3,b_3)$}]. Note that the tuples {$R_1(a_1,b_1)$}, {$R_3(a_1^3,b_3)$} are inconsistent wrt the new constraint. Now we have two solutions. Either we remove {$R_1(a_1,b_1)$} or {$R_3(a_1^3,b_3)$} from the answer. However, we can not remove {$R_1(a_1,b_1)$} since Σ_{12} will be violated. Therefor, the consistent answer is [{$R_1(a_1,b_1)$}, {$R_2(a_1^2,b_1^2)$}].
We can achieve the consistent answer considering the repair concepts. Consider the repairs r_1 and r_2 that we obtained in example 3 that already satisfy Σ_{12}. Now, if we apply new Σ_{13} then we get the following two repairs.

From r_1 we get only one repair which is r_3 = [{$R_1(a_1,b_1)$}, {$R_2(a_1^2,b_1^2)$}].

Notice that only the tuple $R_3(a_1^3,b_3)$ violates Σ_{13}. We could also delete $R_1(a_1,b_1)$ but this leads to violation of Σ_{12}. Similarly, from r_2, we get the repair r_4 = [{$R_1(a_1,b_1)$,$R_1(a,b_1)$}, {$R_2(a_1^2,b_1^2)$, $R_2(a_2^2,b_1^2)$}].

This repair is obtained by deleting the tuple $R_2(a_1^3,b_3)$ since it violates Σ_{13}. Therefore, considering Σ_{12} and Σ_{13}, we have two solutions {r_3, r_4} for P_1 wrt to its acquaintees.

Consider the last query q at P_1. The only consistent answer to the query q is [{$R_1(a_1,b_1)$}, {$R_2(a_1^2,b_1^2)$}].

IV. Computing Consistent Answers

When a query is posed to a peer P_i (called *initial peer*), the query should be processed appropriately in the system in order to gather data distributed across different peers to build a solution instance for P_i and return consistent answers. There are two phases to return consistent answers to a query. First is the query translation and propagation phase and second is the solution building phase. In the first phase, the initial peer executes the query in a straight forward fashion and propagates the query to its acquainted peers after translation

wrt the vocabularies of the acquainted peers. The query translation is required since users submit queries wrt the schema of the local peer. In translation, the query is defined into a compatible form for the schema and data vocabularies of acquaintees. For translating queries between peers, we can use the query translation algorithm that is proposed in [2]. However, here we mainly discuss how to achieve consistent answers. When an acquaintee receives the query it also performs the same task, i.e., local execution, translation, and propagation. The local execution of the query and its translation and propagation to other peers goes on parallel. Note that the solution instances for the initial peer will be determined not only by its relationships with its acquaintees, but also by the acquaintees of its acquaintees, etc. This is a recursive process since the solutions for the acquaintees have to be determined first. Base cases of the recursion are peers those have no acquaintees to forward the query i.e., the peers have no data sharing constraints with any other peer. We call these peers *terminate peers*. Therefore, the query is propagated from the initial peer to all accessible peers that are relevant to the query until the query propagation ends at terminate peers.
The solution building phase starts at terminate peers and ends at initial peer. In the solution building phase, each peer in a query propagation path receives consistent answers from it's acquaintees where the query is propagated. After receiving consistent answers from all acquaintees, a peer builds its solution, called *acquaintee solution* that satisfies the data sharing constraints of the peer's acquaintees.

After building the solution, consistent answers are produced and the result is propagated to the peer that has forwarded the query. This back propagation of consistent answers continues until the initial peer receives results from the acquaintees where the query is initially forwarded. When the initial peer receives data from the acquaintees it builds its own solution instances and returns the consistent answers to the user who initiated the query.

The solutions for a peer are used as a conceptual, auxiliary tool to characterize the consistent answers of a query [12] [13]. It is not practical to build a solution like this since peers are autonomous and heterogeneous. A solution for a peer P_i is a closest database that satisfies P_i's data sharing constraints. Solutions are virtual and the solution concept is used logically to find consistent answers. In order to return consistent answers, each peer follows the steps below.
1. When a peer receives results from peers then the peer performs outer union of the results. Since data vocabularies of peers are different, it is not feasible to merge the results in a common format.
2. Applies the constraints on each of the results and filters the results that satisfy the relevant mapping tables.
3. Passes the results to the sender of the query.
We illustrate the process with an example.

Example 4 *Consider the peer data sharing setting in example 3 and the query* $q(A_1,B_1)$: $R_1(A_1,B_1)$ *at* P_1. *According the algorithm [2], the query will be translated for* P_2 *and* P_3 *as* $q_2(A_2,B_2)$: $R_2(A_2,B_2)$ *and* $q_3(A_3,B_3)$: $R_3(A_3,B_3)$, *respectively.* P_1 *now forwards* q_2 *to* P_2 *and* q_3 *to* P_3. *Since* P_2 *and* P_3 *have no acquaintees then the propagation of q terminates. Now* P_2 *and* P_3 *executes queries* q_2 *and* q_3 *in their local databases and send results to* P_1. P_1 *receives the results* $r_2 = \{R_2(a_1^2,b_1^2), R_2(a_2^2,b_1^2), R_2(a_5^2,b_5^2)\}$ *from* P_2 *and* $r_3=\{R_3(a_1^3,b_3)\}$ *from* P_3. *Moreover, the local results produced by* P_1 *is* $r_1=\{R_1(a_1,b_1), R_1(a_3,b_3)\}$.

Now P_1 *applies the data sharing constraints over the results in order to produce consistent answers. First* P_1 *resolves inconsistencies of the results between* r_1 *and* r_2. *The inconsistencies are resolved by applying the value-level constraints in the relevant mapping tables. A mapping table m(X,Y) is relevant to q if* $X \subseteq att(q)$, *where att(q) denotes the set of attributes in query q. Hence, the relevant mapping tables for the query q are* m_1 *and* m_2 *since* $A_1 \subseteq att(q)$ *and* $A_2 \subseteq att(q)$. *If the mapping tables are applied to filter results then we obtain the consistent result* $r_{12}=[\{R_1(a_1,b_1)\}, \{R_2(a_1^2,b_1^2)\}]$ *from* P_1 *and* P_2. *Now* P_1 *resolves inconsistencies between the results* r_1 *and* r_3. *The relevant mapping table is* m_3. P_1 *applies* m_3 *on the results* r_1 *and* r_3 *and obtains the consistent result* $r_{13}=[\{R_1(a_1,b_1)\}, \{R_3(a_1^3,b_3)\}]$ *from* P_1 *and* P_3. *Therefore, the final consistent answer is* $[\{R_1(a_1,b_1)\}, \{R_2(a_1^2,b_1^2)\}, \{R_3(a_1^3,b_1^3)\}]$. *Now consider the case when a new mapping table* m_4 *between* P_1 *and* P_3 *is introduced. In this case, tuple* $R_3(a_1^3,b_3)$ *is deleted from* r_3. *Therefore, the consistent answer is* $[\{R_1(a_1,b_1)\}, \{R_2(a_1^2,b_1^2)\}]$ *considering the change of mappings.*

V. Related Work

There is an increasing interest in the creation of peer data management systems [3], [16], [17] which includes establishing and maintaining mappings between peers and processing of queries using appropriate propagation techniques. However, the systems do not consider the case of obtaining consistent answers to queries in the presence of the situations where peers may be inconsistent wrt mappings. The inconsistent situations are very common in peer data management systems since peers are autonomous and store data independently. Moreover, mappings may be changed in time. Therefore, it is necessary to find an approach to obtain consistent answers of queries in the systems without changing the physical data in peers to solve inconsistencies.

Authors in [12], [13] introduced a semantics for obtaining consistent answers in peer data exchange systems. The semantics utilize the concepts of repair [9] semantics that is proposed to obtain consistent answers in inconsistent databases. Our work also goes in this direction. However, we consider a system where peers are related with value-level constraints that are created by mapping tables [1]. Authors in [2] proposed a query translation algorithm considering that the peers are related with value-level constraints. However, the

authors do not consider the case of consistent query answering.

VI. Related Work

We have presented an approach for obtaining consistent answers of queries in a peer data sharing system. In our framework, each peer solves data inconsistencies at query time. The inconsistency results when data in peers do not satisfy mappings in mapping tables or when mapping are changed. Therefore, our semantics allows inconsistencies between peers. In our system, we consider that peers are related with value level constraints. We assume that acquaintance graph is acyclic. However, our future goal is to analyze the approach in the presence of cycles in mappings. Moreover, we like to implement and investigate our approach in a large peer data sharing system.

References

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

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

[3] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R.J. Miller, and J. Mylopoulos. The Hyperion Project: From Data Integration to Data Coordination. In *SIGMOD RECORD*, 2003.

[4] P. Rodriguez-Gianolli, M. Garzetti, L. Jiang, A. Kementsietsidis, I. Kiringa, M. Masud, R. Miller, and J. Mylopoulos. Data Sharing in the Hyperion Peer Database System. In *VLDB*, 2005.

[5] M. Lenzerini. Data Integration: A Theoretical Prespective. In *PODS*, 2001.

[6] M Bouzeghoub and M Lenzerini. Introduction to the special issue on data extraction, cleaning, and reconciliation In *Information Systems*, 26(8):535-536, 2001.

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

[8] L. Serafini, F. Giunchiglia, J. Molopoulos, and P. Bernstein. Local Relational Model:A Logocal Formalization of Database Coordination. Technical Report, Informatica e Telecomunicazioni, University of Trento, 2003.

[9] M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Ansers in Inconsistent Databases. In *PODC*, 1999

[10] L. Bertossi and J. Chomicki. Query Answering in Inconsistent Databases. In *Logics for Emerging Applications of Databases.*, 2003.

[11] L. Bertossi and L. Bravo. Information Sharing Agents in a Peer Data Exchange System. In *Proc. First International Conference on Data Management in Grid and P2P Systems (Globe)*, 2008

[12] L. Bertossi and L. Bravo. The Semantics of Consistency and Trust in Peer Data Exchange Systems. In *Proc. International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, 2007.

[13] L. Bertossi and L. Bravo. Query Answering in Peer-to-Peer Data Exchange Systems In *Proc. International (EDBT) Workshop on Peer-to-Peer Computing and DataBases (P2P&DB)*, 2004

[14] R. Fagin, P. Kolaitis, R. J. Miller, and L.Popa. Data exchange: Semantics and query answering. In *Theoretical Computer Science*, 2005.

[15] A. Fuxman, P. Kolaitis, R. Miller, and W. Tan. Peer data exchange. In *PODS*, 2005.

[16] I. Tatarinov, Z. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, and X. Dong. The piazza peer data management project. In *ICDE*, 2003.

[17] W. S. Ng, B. C. Ooi, K. L. Tan, and A. Y. Zhou. PeerDB:A p2p-based system for distributed data sharing. In *Data Engineering*, 2003.