Query Propagation in a P2P Data Integration System in the Presence of Schema Constraints

Tadeusz Pankowski[1,2]

[1] Institute of Control and Information Engineering,
Poznań University of Technology, Poland
[2] Faculty of Mathematics and Computer Science,
Adam Mickiewicz University, Poznań, Poland
tadeusz.pankowski@put.poznan.pl

Abstract. This paper addresses the problem of data integration in a P2P environment, where each peer stores schema of its local data, mappings between the schemas, and some schema constraints. The goal of the integration is to answer queries formulated against a chosen peer. The answer consists of data stored in the queried peer as well as data of its direct and indirect partners. We focus on defining and using mappings, schema constraints, query propagation across the P2P system, and query reformulation in such scenario. The main focus is the exploitation of constraints for merging results from different peers to derive more complex information, and utilizing constraint knowledge to query propagation and the merging strategy. We show how the discussed method has been implemented in SixP2P system.

1 Introduction

In a peer-to-peer (P2P) data integration scenario, the user issues queries against an arbitrarily chosen peer and expects that the answer will include relevant data stored in all P2P connected data sources. The data sources are related by means of schema mappings. A query must be propagated to all peers in the system along semantic paths of mappings and reformulated accordingly. The partial answers must be merged and sent back to the user peer [9,14,16].

Much work has been done on data integration systems both with a mediated (global) schema and in P2P architecture, where the schema of any peer can play the role of the mediated schema [4,9,10,19]. There is also a number of systems built in P2P data integration paradigm [8], notably Piazza [17], PeerDB [12]). In these research the focus was on overcoming syntactic heterogeneity and schema mappings were used to specify how data structured under one schema (the source schema) can be transformed into data structured under another schema (the target schema) [6,7]. A little work has been paid on how schema constraints influence the query propagation.

In this paper we discuss the problem of query propagation where schemas are defined by means of tree-pattern formulas and there are constraints (XML

A. Hameurlain (Ed.): Globe 2008, LNCS 5187, pp. 46–57, 2008.

functional dependencies) defined over the schemas. We show how mutual relationships between schema constraints and queries can influence both propagation of queries and merging of answers. Taking into account such interrelationships may improve both efficiency of the system and information content included in answers. We shortly show how the issues under consideration have been implemented SixP2P (*Semantic Integration of XML in P2P environment*) system.

In Section 1, formal concepts underlying XML data integration are discussed. The main theoretical result concerning query propagation and merging of answers is included in Section 3. In Sections 4 we show some details about query propagation in SixP2P implementation. Section 5 concludes the paper.

2 Pattern-Based Schemas, Mappings and Queries

2.1 Schemas

In this paper an *XML schema* (a *schema* for short) will be understood as a *tree-pattern formula* [4,14]. Schemas will be used to specify structures of *XML trees*. Other properties of XML trees are defined as *schema constraints*.

Definition 1. *A schema over a set L of labels and a set $\mathbf{x}$ of variables is an expression conforming to the syntax:*

$$\begin{aligned} S &::= /l[E] \\ E &::= l = x \mid l[E] \mid E \wedge ... \wedge E, \end{aligned} \tag{1}$$

where $l \in L$, and x is a variable in $\mathbf{x}$. If variable names are significant, we will write $S(\mathbf{x})$.

Example 1. The schema S_1 in Figure 1 can be specified as follows:

$$\begin{aligned} S_1(x_1, x_2, x_3, x_4) := /pubs[pub[title = x_1 \wedge year = x_2 \wedge \\ author[name = x_3 \wedge university = x_4]]] \end{aligned}$$

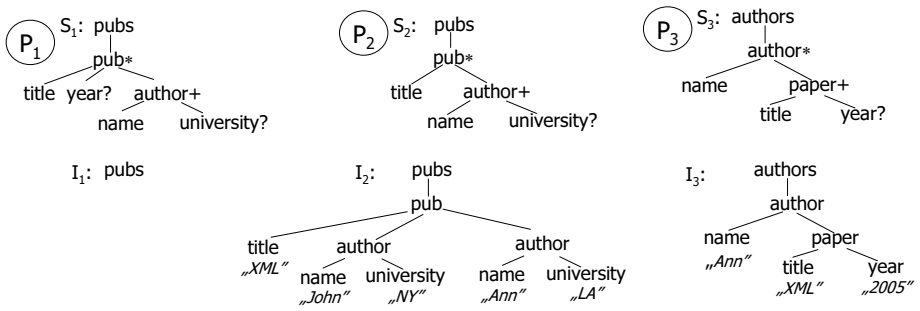

Fig. 1. XML schema trees S_1, S_2, S_3, and their instances I_1, I_2 and I_3, located in peers P_1, P_2, and P_3, respectively

Definition 2. *Let S be a schema over $\mathbf{x}$ and let an atom $l = x$ occur in S. Then the path P starting in the root and ending in l is called the type of the variable x, denoted $type_S(x) = P$.*

Example 2. The type of x_1 in schema S_1 is

$$type_{S_1}(x_1) = /pubs/pub/title.$$

An XML database consists of a set of XML data trees. It will be useful to represent an XML tree I with schema $S(\mathbf{x})$ as a pair $(S(\mathbf{x}), \Omega)$, where Ω is a set of valuations of variables in $\mathbf{x}$.

Definition 3. *Let $Str \cup \{\perp\}$ be a set of strings used as values of text nodes, where $\perp$ is the distinguished null value. Let $\mathbf{x}$ be a set of variable names. A* valuation *ω for variables in $\mathbf{x}$ is a function*

$$\omega \ : \ \mathbf{x} \to Str \cup \{\perp\},$$

assigning values in $Str \cup \{\perp\}$ to variables in $\mathbf{x}$.

Each instance of a schema S can be represented by a pair (S, Ω), where Ω is a set of valuations for variables occurring in S. However, this representation is not unique, since elements in instance trees can be grouped and nested in different ways. By a *canonical instance* we will understand the instance with the maximal width, i.e. the instance where subtrees corresponding to valuations are pair-wise disjoint. For example, the instance I_2 in Figure 1 is not canonical since two authors are nested under one publication. In SixP2P we use canonical instances to handle XML trees efficiently – in particular to merge XML trees with discovering missing values.

2.2 Schema Mappings

The key issue in data integration is this of *schema mapping*. Schema mapping is a specification defining how data structured under one schema (the *source schema*) is to be transformed into data structured under another schema (the *target schema*). In the theory of relational data exchange, *source-to-target dependencies* (STDs) [2] are usually used to express schema mappings [6].

A schema mapping specifies the semantic relationship between a source schema and a target schema. We define it as a source-to-target dependency adapted for XML data [4,15].

Definition 4. *A mapping from a source schema S to a target schema T is a formula of the form*

$$m_{S \to T} := \forall \mathbf{x}(S(\mathbf{x}) \Rightarrow \exists \mathbf{y} T(\mathbf{x}', \mathbf{y})), \tag{2}$$

where $\mathbf{x}' \subseteq \mathbf{x}$, and $\mathbf{y} \cap \mathbf{x} = \emptyset$.

The result of a mapping is the canonical instance of the right-hand side schema, where each variables $y \in \mathbf{y}$ has the $\perp$ (*null*) value.

Example 3. The mapping m_{31} from S_3 to S_1 is specified as:

$$m_{31} := \forall x_1, x_2, x_3 (S_3(x_1, x_2, x_3) \Rightarrow \exists x_4 S_1(x_2, x_3, x_1, x_4)).$$

Then, for $I_3 = (S_3(x_1, x_2, x_3), \Omega)$, where $\Omega = \{(Ann, XML, 2005)\})$,

$$m_{31}(I_3) = J,$$

where $J = (S_1(x_1, x_2, x_3, x_4), \{(XML, 2005, Ann, \bot)\})$.

The set $\Omega' = \{(XML, 2005, Ann, \bot)\}$ is created from Ω using variable correspondences specified in the mapping m_{31}.

2.3 Queries and Query Reformulation

Given a schema S, a *qualifier* ϕ over S is a formula built from constants, as well as paths and variables occurring in S. Let $m_{S \to T} = \forall \mathbf{x}(S(\mathbf{x}) \Rightarrow \exists \mathbf{y} T(\mathbf{x}', \mathbf{y}))$ be a mapping from a source schema S to a target schema T and ϕ be a *query qualifier* over S. A query q over the mapping $m_{S \to T}$ with qualifier ϕ is

$$q := \forall \mathbf{x}(S(\mathbf{x}) \wedge \phi(\mathbf{x}) \Rightarrow \exists \mathbf{y} T(\mathbf{x}', \mathbf{y})). \tag{3}$$

For short, we will denote a query as $q = (m_{S \to T}, \phi)$.

Let $q = (m_{S \to T}, \phi)$ be a query from S to T and $I = (S, \Omega)$ be an instance of S. An answer to a query $q(I)$ is such an instance $J = (T, \Omega)$ of T that its description Ω' is defined as:

$$\Omega' = \{\omega.restrict(\mathbf{x}') \cup null(\mathbf{y}) \mid \omega \in \Omega \wedge \phi(\omega) = true\}, \tag{4}$$

where $\omega.restrict(\mathbf{x}')$ is the restriction of the valuation ω to the variables in $\mathbf{x}'$, and $null(\mathbf{y})$ is a valuation assigning nulls to all variables in $\mathbf{y}$.

Example 4. The query

$$q_{12} = (m_{S_1(x_1,x_2,x_3,x_4) \to S_2(x_1,x_3,x_4)}, x_3 = \text{“}John\text{”} \wedge x_2 = \text{“2005”}),$$

filters an instance of the source schema S_1 according to the qualifier and produces an instance of the schema S_2.

A query is issued by the user against a peer. The user sees the target schema $T(\mathbf{z})$, and defines a qualifier $\phi(\mathbf{z})$, so initially the query is of the form $q = (m_{T(\mathbf{z}) \to T(\mathbf{z})}, \phi(\mathbf{z}))$. When the query is propagated to a source peer with the schema $S(\mathbf{x})$, it must be reformulated accordingly. Thus, the query is to be reformulated into a query $q' = (m_{S(\mathbf{x}) \to T(\mathbf{x}', \mathbf{y})}, \phi'(\mathbf{x}))$.

The reformulation is performed as follows (Figure 2):

1. We want to determine the qualifier $\phi'(\mathbf{x})$ over the source schema $S(\mathbf{x})$. To do this we use the mapping $m_{S(\mathbf{x}) \to T(\mathbf{x}', \mathbf{y})}$.
2. The qualifier $\phi'(\mathbf{x})$ is obtained as the result of rewriting the qualifier $\phi(\mathbf{z})$

$$\phi'(\mathbf{x}) := \phi(\mathbf{z}).rewrite(T(\mathbf{z}), T(\mathbf{x}', \mathbf{y})). \tag{5}$$

   The rewriting consists in appropriate replacement of variable names. A variable $z \in \mathbf{z}$ occurring in $\phi(\mathbf{z})$ is replaced by such a variable $x \in \mathbf{x}'$ that the type of z in $T(\mathbf{z})$ is equal to the type of x in $T(\mathbf{x}', \mathbf{y})$. If such $x \in \mathbf{x}'$ does not exist, the query is not rewritable.

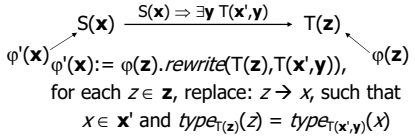

Fig. 2. Reformulation of a query $(m_{T(\mathbf{z})\to T(\mathbf{z})}, \phi(\mathbf{z}))$ into a query $(m_{S(\mathbf{x})\to T(\mathbf{x}',\mathbf{y})}, \phi'(\mathbf{x}))$ using the mapping $\forall \mathbf{x}(S(\mathbf{x}) \Rightarrow \exists \mathbf{y} T(\mathbf{x}', \mathbf{y}))$

Example 5. For the query $q_{11} = (m_{S_1(x_1,x_2,x_3,x_4)\to S_1(x_1,x_2,x_3,x_4)}, x_3 = \text{“}John\text{”})$, we have the following reformulation for its propagation to S_2,

$$q_{21} = (m_{S_2(x_1,x_2,x_3)\to S_1(x_1,x_4,x_2,x_3)}, x_2 = \text{“}John\text{”}),$$

since $type_{S_1(x_1,x_2,x_3,x_4)}(x_3)$=$type_{S_1(x_1,x_4,x_2,x_3)}(x_2)$=$/pubs/pub/author/name$.

3 Deciding about Merging and Propagation Modes

In this section we will discuss haw the existence of *XML functional dependencies* (XFDs) defined over schemas can influence the way of propagating queries and merging partial answers. Our aim is the exploitation of functional dependencies for increasing the amount of information obtained in the process of merging partial results. While merging data from different sources, we can use XFDs to discover some *missing values*, i.e. values denoted by $\perp$.

Definition 5. *An XML functional dependency (XFD) over a set L of labels and a set $\mathbf{x}$ of variables is an expression with the syntax:*

$$\begin{array}{l} f ::= /P[C]/.../P[C], \\ P ::= l \mid P/l, \\ C ::= TRUE \mid P = x \mid C \wedge ... \wedge C, \end{array} \tag{6}$$

where $l \in L$, and x is a variable in $\mathbf{x}$. If variable names are significant, we will write $f(\mathbf{x})$.

Example 6. XFD over S_3 is

$$f(x_2) := /authors/author/paper[title = x_2]/year,$$

meaning that the value of $f(x_2)$ is uniquely determined by the values of x_2.

Types of variables are defined in the same way as for schemas. Let

$$f = /P_1[C_1]/.../P_i[\cdots \wedge P_{ij} = x_j \wedge \cdots]/.../P_n[C_n],$$

be an XFD. Then: $type_f(x_j)$=$/P_1/.../P_i/P_{ij}$, additionally, $type(f)$=$/P_1/../P_n$.

An XFD $f(\mathbf{x})$ says that the value $[\![f(\mathbf{x})(\omega)]\!]$ of $f(\mathbf{x})$ according to the XPath semantics [18], is uniquely determined by a valuation ω of variables in $\mathbf{x}$

Let $f(x_1, ..., x_k)$ be an XFD over $S(\mathbf{x})$, and x be a variable in $\mathbf{x}$ such that $type_S(x) = type(f)$. An XML tree $I = (S(\mathbf{x}), \Omega)$ satisfies this XFD, if for any two valuations $\omega, \omega' \in \Omega$, the implication holds:

$$\omega(x_1, ..., x_k) = \omega'(x_1, ..., x_k) \Rightarrow \omega(x) = \omega'(x),$$

i.e. equality of arguments implies the equality of values.

Thus, XFD can be used to infer missing value of the variable x in the data tree that is expected to satisfy this XFD [14]. Let ω and ω' be two valuations for variables in $\mathbf{x}$ and:

$$\begin{array}{l}\omega(x_1,...,x_k) = \omega'(x_1,...,x_k),\\ \omega(x) \neq \perp, \text{ and } \omega'(x) = \perp.\end{array} \tag{7}$$

Then, we can take $\omega'(x) := \omega(x)$.

Answers to a query propagated across the P2P systems must be collected and merged. In the merge operation we incorporate the discovery of missing values, i.e. null values $\perp$ are replaced everywhere where it is possible, and this replacement is based on XFD constraints.

Thus, it is important to decide which of the following two merging modes should be selected in the peer while partial answers are to be merged:

- *Partial merge* – A partial answer $q(I_S)$ obtained from the propagation is merged with the local answer $q(I_T)$ over the target schema. The answer is: $Ans_{part} = merge(q(I_S), q(I_T))$.
- *Full merge* – The whole instance I_T in the target peer is merged with received partial answer $q(I_S)$, and then the query is evaluated on the result of the merge. Then the answer is: $Ans_{full} = q(merge(q(I_S), I_T))$.

It is quite obvious that the full merge is much more costly than the partial one. However, during full merge more missing values can be discovered. Thus, it should be performed when there is a chance to discover missing values. The following Proposition 1 states the sufficient condition when there is no sense in applying full merge because no missing value can be discovered.

Proposition 1. *Let $S(\mathbf{x})$ be a schema, $f(\mathbf{z})$ be an XFD over $S(\mathbf{x})$, and $type(f) = type_S(x)$ for some $x \in \mathbf{x}$. Let q be a query with qualifier $\phi(\mathbf{y})$, $\mathbf{y} \subseteq \mathbf{x}$, I be the instance of S and I_A an answer to q received from a propagation. Then*

$$q(merge(I_A, I)) = merge(q(I), q(I_A)). \tag{8}$$

holds if one of the following two conditions holds

(a) $x \in \mathbf{y}$, *or*
(b) $\mathbf{z} \subseteq \mathbf{y}$.

Proof. The equality (8) does not hold if there are valuations $\omega' \in \Omega_{I_A}$ and $\omega \in \Omega_I$ such that $\omega'(x) = \perp$, $\omega(x) \neq \perp$, and $\omega'(\mathbf{z}) = \omega(\mathbf{z})$ (see (7)).

Let us consider conditions (a) and (b):

1. *Condition (a).* If $x \in \mathbf{y}$, then there cannot be $\omega' \in \Omega_{I_A}$ such that $\omega'(x) = \perp$, because then $\phi(\mathbf{y})(\omega') \neq true$. Thus, the theorem holds.
2. *Condition (b).* Let $\mathbf{z} \subseteq \mathbf{y}$. If there is such $\omega' \in \Omega_{I_A}$ that $\omega'(x) = \perp$, then:
   - if $\phi(\mathbf{y})(\omega') = true$ then $\omega' \in \Omega_{q(I)}$ and (8) holds;
   - if $\phi(\mathbf{y})(\omega') \neq true$ then ω can belong neither to $\Omega_{q(I)}$ nor to $\Omega_{q(I_A)}$, and ω is not relevant for discovering missing values. So, (8) holds.

To illustrate application of the above proposition let us consider a query about *John*'s data in peers P_2 and P_3 in Figure 1.

1. Let q be a query with qualifier $\phi_2 := x_2 =$ "*John*" in the peer P_2. There is also XFD $f_2 := /pubs/pub/author[name = x_2]/university$ specified over $S_2(x_1, x_2, x_3)$. In force of Proposition 1 there is no chance to discover any missing value of *John*'s university. Indeed, if we obtain an answer with $university = \perp$, then the real value is either in the local answer $q(I_2)$ or it does not occur in I_2 at all. So, in P_2 the partial merge should be performed. Performing the full join in this case is pointless.
2. Let q be a query with qualifier $\phi_3 := x_1 =$ "*John*" issued against peer P_3. There is XFD $f_3 := /authors/author/paper[title = x_2]/year$ specified over $S_3(x_1, x_2, x_3)$. Assumptions of Proposition 1 are not satisfied, so there is a chance to discover missing values of *year* using the full merge. Indeed, from P_2 we obtain the answer $q(I_2) = (S_3, \{(\text{"}John\text{"}, \text{"}XML\text{"}, \perp)\})$. The local answer $q(I_3)$ is empty. But performing the full merge and using f_3, we obtain: $q(merge((S_3, \{(\text{"}John\text{"}, \text{"}XML\text{"}, \perp)\}), (S_3, \{(\text{"}Ann\text{"}, \text{"}XML\text{"}, \text{"2005"})\}))= (S_3, \{(\text{"}John\text{"}, \text{"}XML\text{"}, \text{"2005"})\})$. Thus, the year of *John*'s publication has been discovered and the using of full merge is justified.

The consequences of Proposition 1 impacts also the way of query propagations. The $P2P$ propagation (i.e. to all partners with the $P2P$ propagation mode) may be rejected because of avoiding cycles. However, when the analysis of the query qualifier and XFD's shows that there is a chance to discover missing values, the peer can decide to propagate the query with the *local* mode (i.e. it expects only the local answer from a partner, without further propagations). Such behavior can take place in peer P_3 in the case (2) discussed above.

4 Data Integration in SixP2P

4.1 Overall Architecture

SixP2P is built around a set of peers having a common architecture and communicating each other by sending (propagating) queries and returning answers. According to the P2P technology, there is not any central control over peer's behavior and each peer is autonomous in performing its operations, such as accepting queries, query answering and query propagation. Overall architecture of the system is depicted in Figure 3.

Each peer in SixP2P has its own local database consisting of two parts: data repository of data available to other peers, and 6P2P repository of data necessary for performing integration processes (e.g., information about partners, schemas, constraints, mappings, answers). Using the query interface (QI) a user formulates a query. The query execution module (QE) controls the process of query reformulation, query propagation to partners, merging of partial answers, discovering missing values, and returning partial answers [5,13,14,15]. Communication between peers (QAP) is realized by means of Web Services technology.

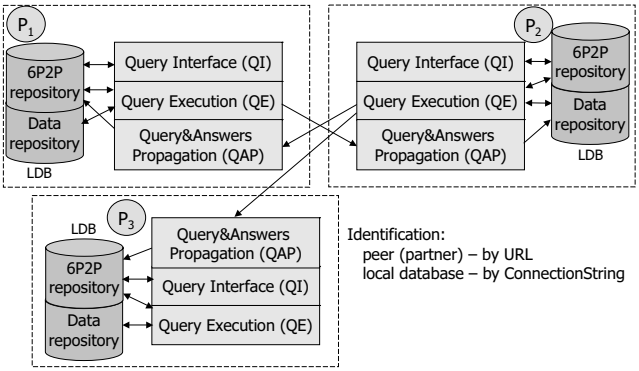

Fig. 3. Overall architecture of SixP2P

4.2 SixP2P Modeling Concepts

Basic notions constituting the SixP2P data model are: peers, data sources, schemas, constraints, mappings, queries, answers, and propagations.

1. A *peer*, @p, is identified by its URL address identifying also the Web Service representing the peer. There are two methods exported by a peer: *sendAnswer* – used by peers to send to @p the answer to a query received from @p, and *propagateQuery* – used by peers to send to @p a query to be answered (possibly with further propagations).
2. A *data source* is an XML document or an XML view over a centralized or distributed data. Different techniques can be used to implement such a view – it can be AXML documents [1,11], a gateway to another SixP2P system or even to different data integration systems. Thus, a community of various collaborating information integration engines can be created.
3. A *schema* is used to specify structural properties of the data source and also the structure of the intermediate answers to queries. In SixP2P, schemas are defined as tree-pattern formulas discussed in previous sections.
4. *Constraints* delivers additional knowledge about data. Two kinds of constraints are taken into consideration: *XML functional dependencies* (XFD) and XML *keys* [3,15]. XFD will be used to control query propagation and answer merging (especially to discover some *missing data*), and keys for eliminating duplicates and appropriate nesting of elements. In this paper we restrict ourselves to XFDs.
5. *Mappings* specify how data structured under a source schema is to be transformed into data conforming to a target schema [6,14]. Information provided by mappings is also used to query reformulation. In [14] we presented algorithms translating high level specifications of mappings, queries and constraints into XQuery programs.

6. A *query* issued against a peer can be split up to many *query threads* – one query threads for one trace incoming to the peer (corresponding to one propagation). Partial answers to all query threads are merged to produce the answer to the query. A peer can receive many threads of the same query.
7. An *answer* is the result of query evaluation. There are *partial* and *final* answers. A partial answer is an answer delivered by a partner who the query was propagated to. All partial answers are merged and transformed to obtain the final answer. In some cases (when the peer decides about discovering missing values, see Section 3), a whole peer's data source may be involved into the merging process. In [13] we discuss a method of dealing with *hard* inconsistent data, i.e. data that is other than null and violates XFDs. The method is based on trustworthiness of data sources.
8. A *propagation* is a relationship between a peer (the *target peer*) and another peer (the *source peer*) where the query has been sent (propagated) to. While propagating queries, the following three objectives are taken into account: (1) avoiding cycles, (2) deciding about propagation modes ($P2P$ or $local$), and (3) deciding about merging modes ($full$ or $partial$) (see Proposition 1).

4.3 SixP2P Database

A peer's database (PDB) consists of five tables: $Peer$, $Constraints$, $Partners$ (Figure 4), $Queries$, and $Propagations$ (Figure 5).

- $Peer(myPeer, myPatt, myData, xfdXQuery, keyXQuery)$–has exactly one row, where: $myPeer$ – the URL of the peer owning the database; $myPatt$ – the schema (tree-pattern formula) of the data source; $myData$ – the peer's data source, i.e. an XML documents or an XML view over some data repositories. $xfdXQuery$ and $keyXQuery$ are XQuery programs obtained by the translation of constrain specifications, XFDs and keys, respectively [14].
- $Constraints(constrId, constrType, constrExp)$ – stores information about the local data constraints (in this paper we discuss only XFDs).
- $Partners(partPeer, partPatt, mapXQuery)$ – stores information about all peer's partners (acquaintances), where: $partPeer$ – the URL of the partner; $partPatt$ – the right-hand side of the schema mapping to the partner (variable names reflect correspondences between paths in the source and in the

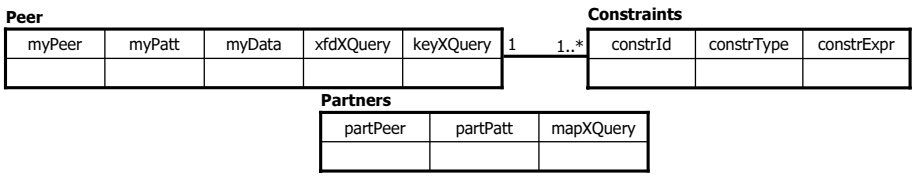

Fig. 4. Structure of tables $Peer$, $Constraints$ and $Partners$

target schema). $mapXQuery$ is an XQuery program obtained by translation of the mapping determined by the $Peer.myPatt$ and $Partners.partPatt$[14].

– $Queries$ and $Propagations$ (Figure 5) maintain information about queries, $qryId$, and their threads, $qryThreadId$, managed in the SixP2P system. The user specifies a qualifier of the query, $myQualif$, as well as propagation ($propagMode$) and merging ($mergeMode$) modes.

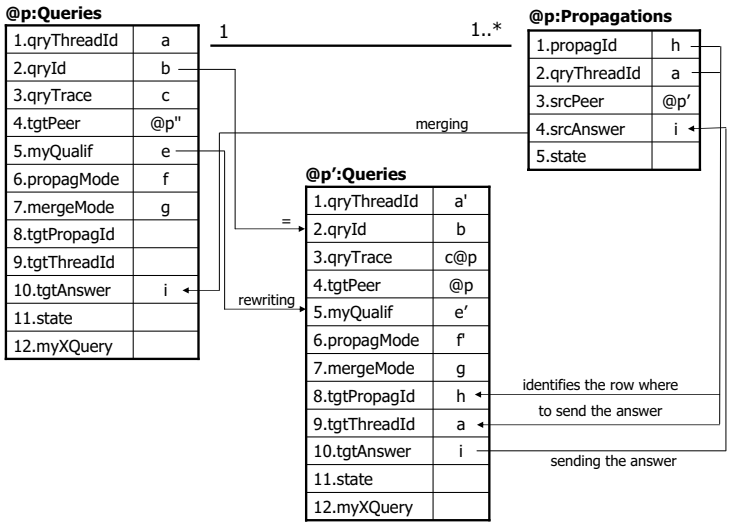

Fig. 5. $Queries$ and $Propagations$ tables in SixP2P. Sample data illustrates instances of tables when a query is propagated from a peer $@p$ to a peer $@p'$.

Symbolic values in tables (Figure 5) indicate relationships between tuples in tables maintained by peers $@p$ and $@p'$. Algorithm 1 describes propagation of a query (more precisely, a thread of the query) stored in $@p : Queries$. The propagation mode can be either $P2P$ (a query is to be propagated to all partners with $P2P$ mode), or *local* (a query is to be answered in the peer's data source without propagation).

Tuple $q1$ contains a query and its context (e.g. $q1.myPatt$ is a schema against which $q1.myQualif$ has been formulated). $LeadsToCycle(q1, @p')$ returns *true* if the propagation causes a cycle, i.e. $@p'$ occurs in $q1.qryTrace$. $discovery\ MayBeDone(q1, @p)$ is *true* if the hypothesis of Proposition 1 does not hold. $acceptedPropagation(q1, @p')$ is *true* if $@p'$ accepts the propagation of $q1$ with given parameters.

If the source peer $@p'$ accepts a propagation $q1$ then it creates the following tuple $q2$ and inserts it into the $@p' : Queries$ table:

Algorithm 1 (*query propagation*)

Input: $@p$ – a current peer; $@p : Peer$, $@p : Partners$,
$@p : Queries$, $@p : Propagations$ – tables on the peer $@p$;
Output: New states of tables $@p : Queries$ and $@p : Propagations$
if a partner peer $@p'$ accepts the propagation.

q :=@p:Queries; *// a row describing the query thread to be propagated*
if *q.propagMode='P2P'* {
  $q1$:= **new** $propagationParametersType$;
    // used to prepare propagations to all partners
  $q1.propagId$:= **new** $propagId$;
  $q1.qryThreadId$:= $q.qryThreadId$;
  $q1.qryId$:= $q.qryId$;
  $q1.qryTrace$:= $q.qryTrace + @p$;
    // the sequence of visited peers used to avoid cycles
  $q1.myPeer$:= $@p$; *// the peer where the answer should be returned*
  $q1.myQualif$:= $q.myQualif$; *// the query qualifier*
  $q1.propagMode$:= "$P2P$";
  $q1.mergeMode$:= $q.mergeMode$;
  $q1.myPatt$:= *@p:Peer.myPatt*; *// the schema of @p*
  foreach $@p'$ **in** *@p:Partners.partPeer*
  { *// attempt to propagate the query to all partners*
    if $LeadsToCycle(q1, @p')$ **and not** $discoveryMayBeDone(q1, @p)$
      then next
    if $LeadsToCycle(q1, @p')$ **then** $q1.propagMode :=$ "$local$";
    if $acceptedPropagation(q1, @p')$ **then**
      insert into *@p:Propagations*
      values $(q1.propagId, q1.qryThreadId, @p', null,$ "$Waiting$") }
}

5 Conclusions

The paper presents a method for schema mapping and query reformulation in a P2P XML data integration system. The discussed formal approach enables us to specify schemas, schema constraints, schema mappings, and queries in a uniform and precise way. We discussed some issues concerning query propagation strategies and merging modes, when missing data is to be discovered in the P2P integration processes. The approach is implemented in SixP2P system. We presented its general architecture, and sketched the way how queries and answers were sent across the P2P environment.

Acknowledgement. The work was supported in part by the Polish Ministry of Science and Higher Education under Grant N516 015 31/1553.

References

1. Abiteboul, S., Benjelloun, O., Manolescu, I., Milo, T., Weber, R.: Active XML: Peer-to-Peer Data and Web Services Integration. In: VLDB, pp. 1087–1090. Morgan Kaufmann, San Francisco (2002)
2. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
3. Arenas, M.: Normalization theory for XML. SIGMOD Record 35(4), 57–64 (2006)
4. Arenas, M., Libkin, L.: XML Data Exchange: Consistency and Query Answering. In: PODS Conference, pp. 13–24 (2005)
5. Brzykcy, G., Bartoszek, J., Pankowski, T.: Schema Mappings and Agents' Actions in P2P Data Integration System. Journal of Universal Computer Science 14(7), 1048–1060 (2008)
6. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.C.: Composing Schema Mappings: Second-Order Dependencies to the Rescue. In: PODS, pp. 83–94 (2004)
7. Fuxman, A., Kolaitis, P.G., Miller, R.J., Tan, W.C.: Peer data exchange. ACM Trans. Database Syst. 31(4), 1454–1498 (2006)
8. Koloniari, G., Pitoura, E.: Peer-to-peer management of XML data: issues and research challenges. SIGMOD Record 34(2), 6–17 (2005)
9. Madhavan, J., Halevy, A.Y.: Composing Mappings Among Data Sources. In: VLDB, pp. 572–583 (2003)
10. Melnik, S., Bernstein, P.A., Halevy, A.Y., Rahm, E.: Supporting Executable Mappings in Model Management. In: SIGMOD Conference, pp. 167–178 (2005)
11. Milo, T., Abiteboul, S., Amann, B., Benjelloun, O., Ngoc, F.D.: Exchanging intensional XML data. ACM Trans. Database Syst. 30(1), 1–40 (2005)
12. Ooi, B.C., Shu, Y., Tan, K.-L.: Relational Data Sharing in Peer-based Data Management Systems. SIGMOD Record 32(3), 59–64 (2003)
13. Pankowski, T.: Reconciling inconsistent data in probabilistic XML data integration. In: British National Conference on Databases (BNCOD 2008). LNCS, vol. 5071, pp. 75–86. Springer, Heidelberg (2008)
14. Pankowski, T.: XML data integration in SixP2P – a theoretical framework. In: EDBT Workshop Data Management in P2P Systems (DAMAP 2008), pp. 1–8. ACM Digital Library (2008)
15. Pankowski, T., Cybulka, J., Meissner, A.: XML Schema Mappings in the Presence of Key Constraints and Value Dependencies. In: ICDT 2007 Workshop EROW 2007, CEUR Workshop Proceedings. CEUR-WS.org, vol. 229, pp. 1–15 (2007)
16. Tatarinov, I., Halevy, A.Y.: Efficient Query Reformulation in Peer-Data Management Systems. In: SIGMOD Conference, pp. 539–550 (2004)
17. Tatarinov, I., Ives, Z.G., Madhavan, J., Halevy, A.Y., Suciu, D., Dalvi, N.N., Dong, X., Kadiyska, Y., Miklau, G., Mork, P.: The Piazza peer data management project. SIGMOD Record 32(3), 47–52 (2003)
18. XML Path Language (XPath) 2.0 (2006), `http://www.w3.org/TR/xpath20`
19. Yu, C., Popa, L.: Constraint-Based XML Query Rewriting For Data Integration. In: SIGMOD Conference, pp. 371–382 (2004)