XML data integration in SixP2P – a theoretical framework

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

ABSTRACT

In the paper we discuss the problem of data integration in a P2P environment. In such setting each peer stores schema of its local data, mappings between the schema and schemas of some other peers (peer's partners), and schema constraints. The goal of the integration is to answer queries formulated against arbitrarily chosen peers. 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. A special attention is paid to discovering missing values using schema constraints and to reconcile inconsistent data using reliability levels assigning to the sources of data. The discussed approach has been implemented in SixP2P system (*Semantic Integration of XML data in P2P environment*).

1. INTRODUCTION

In peer-to-peer (P2P) data management systems, the autonomous computing nodes (the *peers*) cooperate to share resources and services. The peers are connected to some other peers they know or discover. In such systems 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, which are used to specify how data structured under one schema (the source schema) can be transformed into data structured under another schema (the target schema) [8, 9]. 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. While merging, we face the problem of discovering missing data and reconciling inconsistent data. In this paper we propose a theoretical framework to deal with these issues. The approach was verified in implementation of SixP2P system.

Related work. In [13, 11], the peer-to-peer data management systems (PDMS) are defined as decentralized, easily extensible data management architectures in which any user can contribute new data, schema information, or mappings between other peers' schemas. They are a natural step beyond data integration systems with a global schema, where the single logical schema is replaced with an interlinked collection of semantic mappings between peers' individual schemas. The formal foundation of mapping specification for relational data was proposed in [8] as *source-to target dependencies* [1] or GLAVs [12, 5]. An adaptation of this concept to XML data integration was discussed in [3], where *tree-pattern formulas* [21] were used instead of relational ones. Yu and Popa [22] proposed a constraint-based query rewriting for answering queries through a target schema, given a set of mappings between source schemas and the target schema. In addition to the source-to-target mappings they consider a set of target constraints specifying additional properties on the target schema. This approach was used in the Clio system [10, 6]. Integrated data needs to be "repaired" every time we have a violation of constraints imposed by the target schema [3, 18]. In [3] the "easy" and "hard" violations are distinguished: the former are those when nodes do not have the right attributes, they miss some – then we add them and give them distinguishable *null* values. The "hard" violations are those when sequences of children do not satisfy the constraints imposed by regular expressions in the target schema – then a special repairing function *ChangeReg* that tries to repair these violations is proposed. In [7] a problem of uncertain mappings is considered; for example, we could be uncertain whether the attribute *mailing-address* is to be mapped to *home-address*, *permanent-address* or *office-address*. Then each mapping has a probability assigned to it and answers are ranked accordingly. In [7] two semantics for probabilistic relational mappings are considered: *by-table* (probability of data depends only on the table(s) where the data comes from) and *by-sequence* (probability of data depends also on the context created by the sequence of tables involved in the mapping). Reconciliation of data in cooperative system using reliability levels of data sources was discussed in [19].

Contributions. This paper describes formal foundations and some algorithms used for XML data integration in SixP2P system. Schemas of XML data are described by means of a class of tree-pattern formulas, like in [3]. These formulas are used to define both schema mappings and queries. In contrast to [3], except for schemas we use tree pattern formulas also to specify constraints (functional dependencies and keys) over schemas. In contrast to [22], we do not use any special language for specifying mappings.

DAMAP'08, March 25th, 2008, Nantes, France.

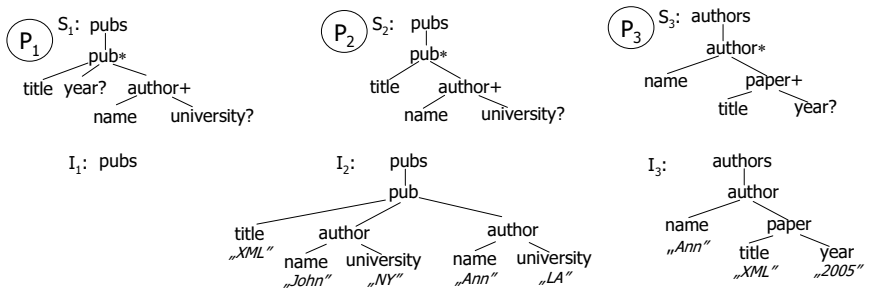

Figure 1: XML schema trees S_1, S_2, S_3, and their instances I_1, I_2 and I_3, located in peers P_1, P_2, and P_3

In SixP2P all, schemas, mappings, queries and constraints are specified in a uniform way as a class of tree-pattern formulas. Thanks to this we are able to translate high-level specifications into XQuery programs. We propose also special procedures to reconciliation of inconsistent data based on reliability levels of data sources (like in [19]). To do this we developed a quite different model from this proposed in [7]. The probabilistic mappings are taken into account if the violation of data constraints (functional dependencies) is detected.

To summarize, the main contributions of the paper are:

- **Formal framework:** A uniform formalism is used to define structure, constraints, mappings and queries. Some formal properties have been proven, in particular we shown the relationship between functional dependencies defined over a schema and the strategy of propagation of queries and merging answers (Proposition 6.4). We developed a novel approach to reconciling inconsistent data using probabilistic mappings based on reliability levels of data sources.

- **Translation algorithms:** We developed and implemented a number of algorithms reformulating queries and translating formal specifications into XQuery programs (queries). The demanded XQuery programs are generated automatically from high level specifications. Such programs are used for: data transformation, query evaluation, discovering missing data, removing duplicates, grouping and nesting.

The paper is organized as follows. Section 2 introduces a running example and discuss execution strategies in P2P environment. We pay attention to the problem of discovering missing values, which impacts on the propagation and merging modes. Basic definitions of XML schemas and instances are introduced in Section 3. Schema mappings and schema constraints are discussed in Section 4 and Section 5, respectively. In Section 6 we define queries and query reformulation. The method of reconciling inconsistent data is illustrated in Section 7. Section 8 concludes the paper.

2. QUERY EXECUTION STRATEGIES

In Figure 1 there are three peers P_1, P_2, and P_3 along with XML schema trees, S_1, S_2, S_3, and schema instances I_1, I_2, and I_3, respectively. Further on, we will assume that XML attributes are represented by elements. We also assume that no element in XML schema has recursive definition. Then XML schemas can be represented by trees, which in turn enables us to specify schemas in a form of tree-pattern formulas.

In P2P data integration systems a query is formulated against an arbitrary target schema (owned by a target peer). In order to obtain a complete answer, the query is to be propagated to all partners of the target peer, these peers propagate it further to their partners, etc. In this way the query can reach all sources, which can contribute to the final answer. Partial answers are merged step-by-step and successively sent towards the target peer. In such scenario the following three issues are of special importance:

1. *Query propagation* – using the information provided by the query and by available schemas, the peer has to decide who to send (propagate) the query to, and whether a coming propagation should be accepted in order to avoid cycles and to increase the expected amount of information included in the answer.

2. *Query reformulation* – a query received and accepted by a peer P_i from a peer P_j has to be reformulated in such a way that it can be evaluated over P_i and its answer conforms to the schema of P_j.

3. *Merging partial answers.* A peer can decide whether the received partial answers should be merged with (the *full merge*) or without (the *partial merge*) the whole peer's local instance. This decision is made based on the functional dependencies defined over the local schema. In this process some missing data can be discovered and some inconsistent data can be reconciled.

We do not assume any centralized control of the propagation. Instead, we assume that a peer makes decision locally based on its knowledge about its schema and schema constraints and about the query that should be executed and propagated. It turns out that the chosen strategy and the way of merging partial answers determine both the final answer and the cost of the execution.

Let us consider some possible strategies of execution of the following query q_{11} against P_1:

Give all available information concerning "John".

The data in the answer should be structured according to the schema S_1. The query is specified as follows:

$$q_{11} := /pubs[pub[title = x_{title} \wedge year = x_{year} \wedge author[name = x_{name} \wedge university = x_{univ}]]] \wedge x_{name} = "John"$$

In q_{11} variables $x_{title}, x_{year}, x_{name}$, and x_{univ} are bound to text values of an XML tree conforming to the source schema (tree-pattern formula) defined by the first conjunct of the query. The second conjunct, $x_{name} = "John"$, is the query qualifier. The answer to the query should contain information stored in all three sources shown in Figure 1 (I_1 is empty).

Thus, one of the following three strategies can be realized (Figure 2):

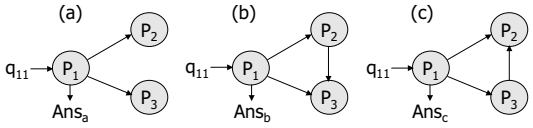

Figure 2: Three execution strategies of the query q_{11}

Strategy (a). Query q_{11} is sent to P_2 and P_3, where it is reformulated to, respectively, q_{21} (from P_2 to P_1) and q_{31} (from P_3 to P_1). The answers $q_{21}(I_2)$ and $q_{31}(I_3)$ are returned to P_1. In P_1 these partial answers are merged with the local answer $q_{11}(I_1)$ and a final answer Ans_a is obtained. This process can be written as follows:

$$\begin{array}{lcl} Ans_a & = & merge(\{Ans^a_{11}, Ans^a_{21}, Ans^a_{31}\}), \\ Ans^a_{11} & = & q_{11}(I_1) = \{(x_{title} : \bot, x_{year} : \bot, x_{name} : \bot, \\ & & \quad x_{univ} : \bot)\}, \\ Ans^a_{21} & = & q_{21}(I_2) = \{(x_{title} : XML, x_{name} : John, \\ & & \quad x_{univ} : NY)\}, \\ Ans^a_{31} & = & q_{31}(I_3) = \{(x_{name} : \bot, x_{title} : \bot, x_{year} : \bot)\}, \\ Ans_a & = & \{(x_{title} : XML, x_{year} : \bot, x_{name} : John, \\ & & \quad x_{univ} : NY)\}. \end{array}$$

Note that the instance I_1 consists of one empty valuation of variables, i.e.

$$I_1 = \{(x_{title} : \bot, x_{year} : \bot, x_{name} : \bot, x_{univ} : \bot)\},$$

where the null value is denoted by $\bot$. Then also the answer $q_{11}(I_1)$ consists of the empty valuation.

The instance I_2 consists of two valuations:

$$I_2 = \{(x_{title} : XML, x_{name} : John, x_{univ} : NY), \\ (x_{title} : XML, x_{name} : Ann, x_{univ} : LA)\},$$

and the first valuation satisfies the query.

Strategy (b). It differs from strategy (a) in that P_2 after receiving the query propagates it to P_3 and waits for the answer $q_{32}(I_3)$. It is easily seen that the final result is equal to Ans_a:

$$\begin{array}{lcl} Ans_b & = & merge(\{Ans^b_{11}, Ans^b_{21}, Ans^b_{31}\}) = \\ & = & \{(x_{title} : XML, x_{year} : \bot, x_{name} : John, \\ & & \quad x_{univ} : NY)\}, \end{array}$$

Strategy (c). In contrast to the strategy (b), the peer P_3 propagates the query to P_2 and waits for the answer. Next, the peer P_3 decides to merge the obtained answer $q_{23}(I_2)$ with the whole its instance I_3. The decision follows from the fact that the functional dependency

$$/authors/author/paper/title \rightarrow \\ /authors/author/paper/year$$

is defined over the local schema of P_3, and it is the necessary condition for discovering missing values (if there are any) of variable x_{year}. So we have:

$$\begin{array}{lcl} Ans_c & = & merge(\{Ans^c_{11}, Ans^c_{21}, Ans^c_{31}\}), \\ Ans^c_{23} & = & \{(x_{title} : XML, x_{year} : \bot, x_{name} : John)\}, \\ Ans^c_{31} & = & q_{31}(merge(\{I_3, Ans^c_{23}\})) \\ & = & \{(x_{title} : XML, x_{year} : 2005, x_{name} : John)\} \\ Ans_c & = & \{(x_{title} : XML, x_{year} : 2005, x_{name} : John, \\ & & \quad x_{univ} : NY)\}. \end{array}$$

While computing the merge $merge(\{I_3, Ans^c_{23}\}$ a missing value of x_{year} is discovered. Thus, the answer Ans_c provides more information than those in strategies (a) and (b).

This example shows that it is useful to analyze relationships between the query and functional dependencies defined over the peer's schema. The analysis can influence the decision about the propagation and merging modes (see Proposition 6.4).

3. XML SCHEMAS AND INSTANCES

Schemas for XML data are usually specified by means of XSDL (XML Schema Definition Language) or DTD (Document Type Definition). In this paper an *XML schema* (a *schema* for short) will be understood as a *tree-pattern formula* [3, 17, 16, 21]. Schemas will be used to specify structures of *XML trees*. Some other properties of XML trees are defined as *schema constraints*.

DEFINITION 3.1. *Let L be set of labels, $top \in L$ be a distinguished label (the outermost label in an XML schema tree), and $\mathbf{x}$ be a vector of text variables. A schema over L and $\mathbf{x}$ is an expression conforming to the syntax:*

$$\begin{array}{lcl} S & ::= & /top[E] \\ E & ::= & l = x \mid l[E] \mid E \wedge ... \wedge E, \end{array}$$

where $l \in L$, and $x \in \mathbf{x}$. In order to indicate the set and ordering of variables in S we will write $S(\mathbf{x})$.

Schemas in the above definition, are fragments of XPath 2.0 predicates [20] of the class $XP^{\{/,[],=,var\}}$. These fragments consist of label tests, child axes (/), branches ([]), equality symbol (=), and variables.

EXAMPLE 3.2. *The schema of the data on the peer P_1 is:*

$$S_1(x_1, x_2, x_3, x_4) := /pubs[pub[title = x_1 \wedge year = x_2 \wedge \\ author[name = x_3 \wedge university = x_4]]].$$

Similarly, $S_2(x_1, x_2, x_3)$ and $S_3(x_1, x_2, x_3)$ for P_2 and P_3.

An XML tree I can be represented by a pair (S, Ω), where Ω is a set of valuations of variables occurring in S. Such representation of instances is not unique since elements in instance trees can be grouped and nested in different ways. Thus, Ω represents a class of instances of the same schema. 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.

By the *type* of a variable we understand the path leading from the root to the leafe which is bound to this variable

DEFINITION 3.3. *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 $type_S(x) = p$.*

The type of x_1 in S_2 is: $type_{S_2}(x_1) = /pubs/pub/title$.

4. 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*). A schema mapping specifies the semantic relationship between a source schema and a target schema.

DEFINITION 4.1. *A mapping from a source schema S to a target schema T is an expression of the form (a source-to-target formula [8])*

$$m := \forall \mathbf{x}(S(\mathbf{x}) \Rightarrow \exists \mathbf{y} T(\mathbf{x}', \mathbf{y})), \quad (1)$$

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

Variable names are used to indicate correspondences between text values of paths bound to variables. In practice, a correspondence also involves a function that transforms values of source and target variables. These functions are irrelevant to our discussion, so they will be omitted.

In fact, a mapping is a special case of a query (see later on), where the query qualifier is $TRUE$. The result of a mapping is the canonical instance of the target schema. All variables in $\mathbf{y}$ have null values (denoted by $\perp$).

EXAMPLE 4.2. *m_{31} is a mapping from S_3 to S_1:*

$m_{31} := \forall x_1, x_2, x_3(/authors[author[name = x_1 \wedge paper[title = x_2 \wedge year = x_3]]] \Rightarrow \exists x_4 / pubs[pub[title = x_2 \wedge year = x_3 \wedge author[name = x_1 \wedge university = x_4]]])$.

In SixP2P mappings are implemented by means of XQuery programs (queries). Algorithm 1 translates a mapping m_{ik} into an appropriate XQuery program. By x, y, v (possibly with subscripts) we denote SixP2P variables, while $\$x, \$y, \$v$ are corresponding XQuery variables.

Algorithm 1 (*translating a mapping to XQuery program*)
Input: A mapping $m_{ik} := \forall \mathbf{x}(S_i \Rightarrow \exists \mathbf{y} S_k)$, where $S_i := /top'[E']$, $S_k := /top[E]$, $\mathbf{y} = (y_1, ..., y_m)$.
Output: Query in XQuery over S_i transforming an instance of S_i into the corresponding canonical instance of S_k.

mappingToXQuery(
$\forall \mathbf{x}(/top'[E'] \Rightarrow \exists y_1, ..., y_m / top[E]))=$

<top>{
let $\$y_1 :=$ "*null*", ..., $\$y_m :=$ "*null*"
for $\$v$ **in** $/top'$,
$\tau(v, E')$
return
$\rho(E)$}
</top>

where v is a newly-invented SixP2P variable, and:

1. $\tau(v, l = x) = \$x$ **in if** $(\$v[l])$ **then** $string(\$v/l[1])$ **else** "*null*",
2. $\tau(v, l[E]) = \$v'$ **in if** $(\$v[l])$ **then** $\$v/l$ **else** /, $\tau(v', E)$,
3. $\tau(v, E_1 \wedge \cdots \wedge E_k) = \tau(v, E_1), \cdots, \tau(v, E_k)$,
4. $\rho(l = x) =$ **if** $defined(\$x)$ **then** <l>{$\$x$}</$l$> **else** <$l$>*null*</$l$>
5. $\rho(l[E]) =$ <l>$\rho(E)$</l>
6. $\rho(E_1 \wedge \cdots \wedge E_k) = \rho(E_1) \ \cdots \ \rho(E_k)$

For the mapping m_{31} (Example 4.2), the XQuery program generated by Algorithm 1 is:

```
<pubs>{ for $_v in /authors,
    $_v1 in if ($_v[author]) then $_v/author else /,
      $x_1 in if ($_v1[name]) then
                string($_v1/name[1]) else "null",
      $_v22 in if ($_v1[paper]) then
                $_v1/paper else /,
        $x_2 in if ($_v22[title]) then
                string($_v22/title[1]) else "null",
        $x_3 in if ($_v22[year]) then
                string($_v22/year[1]) else "null"
  return
  <pub>
     <title>{$x_2}</title>
     <year>{$x_3}</year>
     <author>
        <name>{$x_1}</name>
        <university>null</university>
     </author>
  </pub> }
</pubs>
```

Note that the program creates a canonical instance of S_1, i.e. elements are not grouped and all missing values are replaced by nulls.

5. SCHEMA CONSTRAINTS

Among schema constraints we distinguish *XML functional dependencies* (XFD), and *keys*. To define them we use XPath path expressions of the form:

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

where P is a path, l is a label, and x is a variable.

An XFD constrains the relationship between text values of sets of paths, that in [2] has the form: $\{p_1, ..., p_k\} \rightarrow p$, and a tuple of values denoted by the left-hand side uniquely determines a value of the right-hand side.

In SixP2P, an XFD of the above form is specified by means of the expression:

$$f(x_1, ..., x_k) := /P_1[C_1]/.../P_n[C_k](x_1, ..., x_k),$$

where $p_i = type(x_i)$, $p = type(f) = /P_1/.../P_n$.

EXAMPLE 5.1. *XFD over S_3 is*
$f(x_2) := /authors/author/paper[title = x_2]/year$,
corresponding to
$/authors/author/paper/title \rightarrow /authors/author/paper/year$.

Let $f(x_1, ..., x_k)$ be an XFD over $S(\mathbf{x})$, and x be a variable in $\mathbf{x}$ such that $type(x) = type(f)$. 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. $f(x_1, ..., x_k)$ produces one-item sequence for any valuation of its variables. It means, that XFD can be used to infer missing values of the variable x in data trees which

are expected to satisfy this XFD [14]. Let ω and ω' be two valuations for variables in $\mathbf{x}$ and:

$$\omega(x_1, ..., x_k) = \omega'(x_1, ..., x_k),$$
$$\omega(x) \neq \perp, \text{ and } \omega'(x) = \perp.$$

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

The following algorithm generates an XQuery program for a given schema S and a set F of XFD constraints over this schema. The program discovers all possible missing values, with respect to the set F.

Algorithm 2 (*XFD to XQuery*)
Input: A schema $S = /top[E]$ and a set of XFD constraints, $getfd(x)$ returns XFD f such that $type(f) = type(x)$.
Output: Query in XQuery over S returning the instance of S, where XFD constraints are used to discover missing values.

$xfdToXQuery(/top[E])$.

□

The translation function $xfdToXQuery(/top[E])$ is identical to the translation function in Algorithm 1

$$mappingToXQuery(/top[E] \Rightarrow /top[E]),$$

except the rule (4) is replaced by the rule (4'):

4'. $\rho(l = x) = <l>\{$**if** $(\$x = "null")$ **then** $string((getfd(\$x)[text() != null])[1])$ **else** $\$x\}</l>$.

EXAMPLE 5.2. *Discovering missing values in an instance of S_1 can be done using the XQuery program generated for the schema S_1 and XFD $getfd(x_2)$. The corresponding XQuery program is similar to this of Algorithm 1, where expression defining "year" is:*

```
<year>{
  if ($x2="null") then string((/pubs/pub
    [title=$x1]/year[text()!="null"])[1]) else $x2}
</year>
```

An XML *key* says that an subtree in an XML tree uniquely depends on text values of a specified tuple of path [4]. A key over a schema $S(\mathbf{x})$ is an XPaths path expression of the form: $f(x_1, ..., x_k)$, where any variable x_i is in $\mathbf{x}$, and $type(f)$ is a path denoting a subtree. An instance I of $S(\mathbf{x})$ satisfies the key if a tuple of text values of the tuple $(type(x_1), ..., type(x_k))$ of paths uniquely identifies the subtree of the type $type(f)$.

We assume that there is a key for any subtree defined by a schema S. If the subtree is denoted by P, then its key is denoted by $key(p)$, or $key(l)$, where l is the last label in P.

EXAMPLE 5.3. *For $S_1(x_1, x_2, x_3, x_4)$ we can define:*
$key(pub) = /pubs/pub[title = x_1]$, *or alternatively:*
$key'(pub) = /pubs/pub[title = x_1 \wedge author/name = x_3]$.

The following algorithm generates an XQuery program transforming an XML tree into the tree satisfying all given keys.

Algorithm 3 (*keys to XQuery*)
Input: A schema $S := /top[E]$, and $key(l)$, for each label l occurring in S.
Output: Query in XQuery over S returning an instance satisfying all given keys.

$keysToXQuery(/top[E]) =$ <top>
{ $\tau(E, \emptyset)$ }
</top>,

where:

1. $\tau(l = x, \Gamma) :=$
for $\$x$ **in** distinct-values($key(l).replace(\Gamma)$)
return
<l>{ $\$x$}</l>
2. $\tau(l[E], \Gamma) :=$
if $(key(l) = \kappa/l[P_1 = x_1 \wedge ... \wedge P_k = x_k])$ *then*
foreach x_i *in* $(x_1, ..., x_k)$
if $(x_i$ *is not defined in* $\Gamma)$ *then*
begin
for $\$v$ **in** distinct-values($key(l).replace(\Gamma)/P_i$)
return
$\Gamma := \Gamma \cup \{x_i \mapsto \$v\}$
end
<l>{ $\tau(E, \Gamma)$}</l>
3. $\tau(E_1 \wedge ... \wedge E_k, \Gamma) := (\tau(E_1, \Gamma), ..., \tau(E_k, \Gamma))$.

Γ is a set of replacements for variables. A replacement is an expression of the form $x \mapsto y$ and says that any occurrence of a variable x is to be replaced by the variable y. If κ is a key then $\kappa.replace(\Gamma)$ produces an expression κ', where all variables are replaced according to Γ. If there is no replacement for a variable x_i in a conjunct $P_i = x_i$ in κ, then this conjunct is not included into κ'.

For the schema $S_1(x_1, x_2, x_3, x_4)$ (Example 3.2) and the set of keys defined in Example 5.3, Algorithm 4 generates the following XQuery program:

```
<pubs>{
 for $v_1 in distinct-values(/pubs/pub/title)
 return
 for $v_2 in distinct-values(/pubs/pub[title=$v_1]/author/name)
 return
   <pub>{
    (for $x1 in distinct-values(/pubs/pub[title=$v_1 and
       author/name=$v_2]/title)
     return
      <title>{ $x1 }</title>,
      for $x2 in distinct-values(/pubs/pub[title=$v_1 and
        author/name=$v_2]/year)
      return
       <year>{ $x2 }</year>,
      <author>{
      for $x3 in distinct-values(/pubs/pub[title=$v_1 and
                  author/name=$v_2]/author[name=$v_2]/name)
      return
       (<name>{ $x3 } </name>,
       for $x4 in distinct-values(/pubs/pub[title=$v_1 and
         author/name=$v_2]/author[name=$v_2]/university)
       return
        <university>{ $x4 } </university>)
      }</author>)
   }</pub>
}</pubs>
```

6. QUERIES AND QUERY REFORMULATION

Given a schema S, a *qualifier* ϕ over S is a formula built from constants and variables occurring in S. A query from a source schema S to a target schema T is defined as a mapping from S to T extended with a *query qualifier* ϕ (for simplicity, we will omit the quantifications):

$$q := S \wedge \phi \Rightarrow T.$$

An answer to a query is defined as follows:

DEFINITION 6.1. *Let $I = (S(\mathbf{x}), \Omega)$ be a source instance. An answer to a query $q := S(\mathbf{x}) \wedge \phi \Rightarrow T(\mathbf{x}', \mathbf{y})$ is a target instance $I' = (T(\mathbf{x}', \mathbf{y}), \Omega')$ such that:*

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

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 6.2. *The following query*

$$q_{21} := \quad S_2(x_1, x_2, x_3) \wedge x_2 = "John" \\ \Rightarrow S_1(x_1, x_4, x_2, x_3)$$

filters an instance of the source schema S_2 according to the qualifier $x_2 = "John"$, extends the valuation $\omega(x_1, x_2, x_3)$ to $\omega'(x_1, \perp, x_2, x_3)$, and produces an instance of S_1.

In the reformulation process, we will be interested in the left-hand side of the query, because it contains the qualifier that is to be reformulated. The reformulation consists in an appropriate renaming of variables.

Assume that a query $q_i = S_i(\mathbf{x_i}) \wedge \phi_i(\mathbf{z_i})$ is issued against a target peer P_i. If the query is propagated to a source peer P_k then it must be reformulated into such a query q_k that can be evaluated over data stored on the peer P_k.

1. We want to determine the qualifier $\phi_k(\mathbf{z_k})$ in a query

   $$q_k := S_k(\mathbf{x_k}) \wedge \phi_k(\mathbf{z_k})$$

   over the source schema $S_k(\mathbf{x_k})$, $\mathbf{z_k} \subseteq \mathbf{x_k}$. To do this we use the mapping from $S_k(\mathbf{x_k})$ to $S_i(\mathbf{x_{ik}}, \mathbf{y_{ik}})$, $\mathbf{x_{ik}} \subseteq \mathbf{x_k}$.

2. The qualifier $\phi_k(\mathbf{z_k})$ is obtained as rewriting of the qualifier $\phi_i(\mathbf{z_i})$ according to $S_i(\mathbf{x_i})$ and $S_i(\mathbf{x_{ik}}, \mathbf{y_{ik}})$:

   $$\phi_k(\mathbf{z_k}) := \phi_i(\mathbf{z_i}).rewrite(S_i(\mathbf{x_i}), S_i(\mathbf{x_{ik}}, \mathbf{y_{ik}})).$$

   The rewriting consists in appropriate replacement of variable names. A variable $z \in \mathbf{z_i}$ (z is also in $\mathbf{x_i}$) is replaced by such a variable $x \in \mathbf{x_{ik}}$ that the type of z in $S_i(\mathbf{x_i})$ is equal to the type of x in $S_i(\mathbf{x_{ik}}, \mathbf{y_{ik}})$. If such replacement is impossible, then the qualifier is non-rewritable. (In such the case the corresponding conjunct might be replaced by TRUE giving an approximate query).

EXAMPLE 6.3. *For the query qualifier*

$$\phi_1(x_3) := x_3 = "John"$$

over $S_1(x_1, x_2, x_3, x_4)$, we have the following reformulation over $S_3(x_1, x_2, x_3)$ with respect to the mapping m_{31}:

$$\phi_1(x_3).rewrite(S_1(x_1, x_2, x_3, x_4), S_1(x_2, x_3, x_1, x_4)) \\ = \phi_3(x_1) := x_1 = "John",$$

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

Answers to a query propagated across the P2P systems must be collected and merged. In the merge operation we try to discover 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 two merging modes should be selected in the peer while partial answers are to be merged:

- *partial merge* – all partial answers are merged without taking into account the source instance stored in the peer,
- *full merge* – the whole source instance in the peer is merged with all received partial answers; during this operation XFDs are used to discover missing values; finally the query is evaluated on the result of the merge.

Criterion of the selection is the possibility of discovering missing values during the process of merging. To make the decision one has to analyze XFD constraints specified for the peer's schema and the query qualifier.

Proposition 6.4 states the sufficient condition when there is no sense in applying full merge because no missing value can be discovered.

PROPOSITION 6.4. *Let $S(\mathbf{x})$ be a schema, q be a query with qualifier $\phi(\mathbf{y})$, $\mathbf{y} \subseteq \mathbf{x}$, and I_A be an answer to q received from a propagation. Let $f(\mathbf{z})$ be an XFD defined over $S(\mathbf{x})$, and $type(f) = type(x)$ for some $x \in \mathbf{x}$. Then no missing value can be discovered by full merge, i.e.*

$$q(merge(I, I_A)) = merge(q(I), I_A), \tag{2}$$

if at least one of the following two conditions holds:

(a) $x \in \mathbf{y}$, *or*

(b) $\mathbf{z} \subseteq \mathbf{y}$.

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. In P_2 we have: query qualifier $\phi_2 := x_2 = "John"$, and XFD */pubs/pub/author*[*name* = x_2]/*university*. In force of Proposition 6.4 there is no chance to discover any missing value of *John*'s university in any obtained answer from anywhere.

Indeed, if we obtain an answer with $university = \perp$, then the real value is either in the local answer $\phi_2(I_2)$ or it does not occur in I_2 at all. So, the full merge is senseless.

In P_3 we have: query qualifier $\phi_3 := x_1 = "John"$, and XFD */authors/author/paper*[*title* = x_2]/*year*. Assumptions of proposition 6.4 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 $\{(x_1 : "John", x_2 : "XML", x_3 : \perp)\}$. The local answer is empty. But performing the full merge and using ϕ_3, we obtain:
$\phi_3(merge(\{(x_1 : "John", x_2 : "XML", x_3 : \perp)\},$
$\{(x_1 : "Ann", x_2 : "XML", x_3 : 2005)\}) =$
$= \{(x_1 : "John", x_2 : "XML", x_3 : 2005)\}$.

Thus, the year of *John*'s publication has been discovered (see Section 2, Strategy (c)).

The consequences of Proposition 6.4 impact 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 discussed above.

7. RECONCILING INCONSISTENT DATA

By *inconsistent data* we understand data which values violate a functional dependency defined over the target schema. If the violation is caused by null values we can try to replace them by some non-null values, as was discussed in the previous sections. In the case of non-null values violating a functional dependency, we calculate trustworthiness of data and choose the most reliable [7, 19].

In many cases (e.g. in bioinformatics) some data sources are known to be more credible than others (e.g. SWISS-PROT is human-curated, making it more authoritative than others) [19]. We assume that from a peer's point of view a numeric *reliability level* is assigned to every peer's partner and the following *trust policy* is followed:

1. A vector $r_1, ..., r_n$ of reliability levels is assigned to source peers. A value r_i is treated as the trustworthiness of the answer obtained from the source S_i, provided that answers from different sources are not consistent. Then we apply a *reconciliation procedure* aiming to choose this one that is the most reliable.

2. Reliability levels will be understood as *probabilities* which will be assigned to mappings from the target to source schemas. In this way arise *probabilistic schema mappings* [7, 15].

A probabilistic schema mapping models the uncertainty about which data is the correct one. Like [7] we assume that there are two ways to interpret this uncertainty:

- a mapping is applied to all the data in S – this interpretation will be referred to as the *by-peer* semantics;

- the applied mapping may depend on the particular subtree identified by a given key in S – this interpretation will be referred to as the *by-subtree* semantics.

To illustrate the approach let us consider Figure 3, and the query *"Get all pairs (title, year) issued against S_3"*. Assume that reliability levels of sources S_1, S_2, and S_3 are 0.5, 0.2, and 0.3, respectively. In Table 1 there are answers returned from the three sources. If we interpret the answers according to the by-peer semantics, then probabilities of them are listed in Table 2. The probability of an answer is the sum of the probabilities of the sources it comes from. We see that probabilities of $(XML, 2005)$ and $(XML, 2004)$ are the same.

In the by-subtree semantics we distinguish subtrees where the answers come from [15]. In the source S_3 we consider subtrees identified by the key $/authors/author[name = x_1]$, i.e. we force that a subtree of type $/authors/author$ is uniquely identified by the *name* of the author – in our case by *Ann* and *John*. Now, probabilities are calculated taking into account all possible combinations of answers coming from all subtrees (Table 4). The probability in the row is the product of probabilities of the mappings producing the sequence of answers. In this semantics the highest probability has the answer $(XML, 2004)$ (Table 5) (it is the sum of probabilities of all sequences in which occurs $(XML, 2004)$), so it can be assumed as the correct answer and can be used rather than $(XML, 2005)$. Alternatively, all answers ranked with their probabilities can be returned to the user [7].

The problem of probabilistic data integration we discuss deeply in [15].

Table 1: Answers of Q analyzed in the *by-peer* semantics

S_1 (0.5)	$\langle(C\#, 2006), (XML, 2005)\rangle$
S_2 (0.2)	$\langle(XML, 2004), (XML, 2004)\rangle$
S_3 (0.3)	$\langle(XML, 2004), (SQL, 2004), (C\#, 2005)\rangle$

Table 2: Probabilities of the *by-peer* answers of Q

Tuple	*Probability*
$(C\#, 2006)$	0.5
$(XML, 2005)$	0.5
$(XML, 2004)$	0.5
$(SQL, 2004)$	0.3
$(C\#, 2005)$	0.3

Table 3: Answers of Q analyzed in the *by-subtree* semantics

Key	S_1 (0.5)	S_2 (0.2)	S_3 (0.3)
Ann	$\langle(C\#, 2006)\rangle$	$\langle(XML, 2004)\rangle$	$\langle(XML, 2004), (SQL, 2004)\rangle$
John	$\langle(XML, 2005)\rangle$	$\langle(XML, 2004)\rangle$	$\langle(C\#, 2005)\rangle$

Table 4: Probabilities of pairs of mappings in the *by-subtree* semantics

Subtree with key "Ann"	*Subtree with key "John"*	*Prob*
$\langle(C\#, 2006)\rangle$	$\langle(XML, 2005)\rangle$	0.25
$\langle(C\#, 2006)\rangle$	$\langle(XML, 2004)\rangle$	0.10
$\langle(C\#, 2006)\rangle$	$\langle(C\#, 2005)\rangle$	0.15
$\langle(XML, 2004)\rangle$	$\langle(XML, 2005)\rangle$	0.10
$\langle(XML, 2004)\rangle$	$\langle(XML, 2004)\rangle$	0.04
$\langle(XML, 2004)\rangle$	$\langle(C\#, 2005)\rangle$	0.06
$\langle(XML, 2004)\rangle$, $\langle(SQL, 2004)\rangle$	$\langle(XML, 2005)\rangle$	0.15
$\langle(XML, 2004), (SQL, 2004)\rangle$	$\langle(XML, 2004)\rangle$	0.06
$\langle(XML, 2004)\rangle$, $(SQL, 2004)\rangle$	$\langle(C\#, 2005)\rangle$	0.09

Table 5: Probabilities of the *by-subtree* answers of Q

Tuple	*Probability*
$(C\#, 2006)$	0.5
$(XML, 2005)$	0.5
$(XML, 2004)$	0.6
$(SQL, 2004)$	0.3
$(C\#, 2005)$	0.3

8. CONCLUSION

The paper presents a novel method for schema mapping and query reformulation in XML data integration systems in P2P environment. The discussed formal approach enables us to specify schemas, schema constraints, schema mappings, and queries in a uniform and precise way. Based on this approach we define some basic operations used for query reformulation and data merging, and propose algorithms for automatic generation of operational means (XQuery programs in our case) to perform these operations in real. We discussed some issues concerning query propagation strategies and merging modes, when missing data is to be discov-

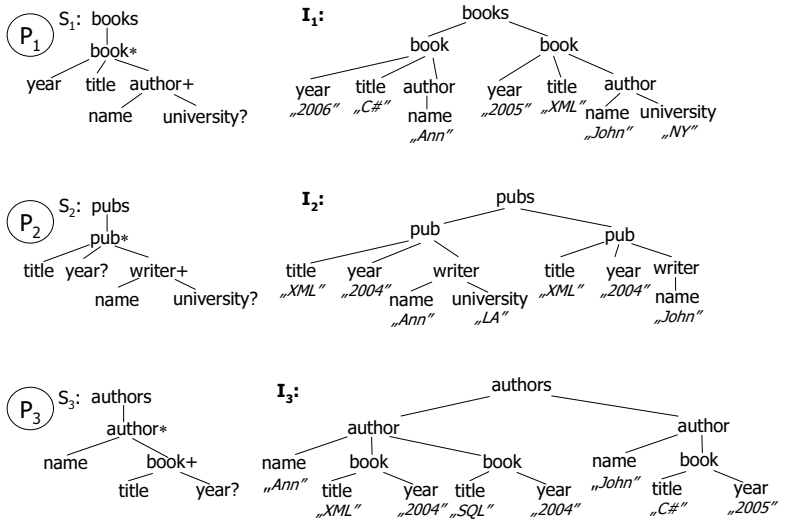

Figure 3: XML schema trees S_1, S_2, S_3, and their instances I_1, I_2 and I_3, located in peers P_1, P_2, and P_3

ered in the P2P integration processes. We showed, how to use schema constraints, mainly functional dependency constraints, to select the way of query propagation and data merging, to increase the information content of the answer to a query. A method for reconciliation of data violating functional dependencies is proposed. The method is based on reliability levels assigned to data sources and on calculating probabilities that answers are correct.

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

9. REFERENCES

[1] Abiteboul, S., Hull, R., Vianu, V.: *Foundations of Databases*, Addison-Wesley, Reading, Massachusetts, 1995.

[2] Arenas, M.: Normalization theory for XML, *SIGMOD Record*, **35**(4), 2006, 57–64.

[3] Arenas, M., Libkin, L.: XML Data Exchange: Consistency and Query Answering, *PODS Conference*, 2005, 13–24.

[4] Buneman, P., Davidson, S. B., Fan, W., Hara, C. S., Tan, W. C.: Reasoning about keys for XML, *Information Systems*, **28**(8), 2003, 1037–1063.

[5] Calvanese, D., Giacomo, G. D., Lenzerini, M., Rosati, R.: Logical Foundations of Peer-To-Peer Data Integration., *Proc. of the 23rd ACM SIGMOD Symposium on Principles of Database Systems (PODS 2004)*, 2004, 241–251.

[6] Chiticariu, L., Hernández, M. A., Kolaitis, P. G., Popa, L.: Semi-Automatic Schema Integration in Clio, *VLDB*, 2007, 1326–1329.

[7] Dong, X. L., Halevy, A. Y., Yu, C.: Data Integration with Uncertainty, *VLDB*, ACM, 2007, 687–698.

[8] Fagin, R., Kolaitis, P. G., Popa, L., Tan, W. C.: Composing Schema Mappings: Second-Order Dependencies to the Rescue, *PODS*, 2004, 83–94.

[9] Fuxman, A., Kolaitis, P. G., Miller, R. J., Tan, W. C.: Peer data exchange, *ACM Trans. Database Syst.*, **31**(4), 2006, 1454–1498.

[10] Haas, L. M., Hernández, M. A., Ho, H., Popa, L., Roth, M.: Clio grows up: from research prototype to industrial tool, *SIGMOD Conference*, 2005, 805–810.

[11] Halevy, A. Y., Ives, Z. G., Suciu, D., Tatarinov, I.: Schema mediation for large-scale semantic data sharing, *VLDB J.*, **14**(1), 2005, 68–83.

[12] Lenzerini, M.: Data Integration: A Theoretical Perspective., *PODS* (L. Popa, Ed.), ACM, 2002, 233–246.

[13] Madhavan, J., Halevy, A. Y.: Composing Mappings Among Data Sources., *VLDB*, 2003, 572–583.

[14] Pankowski, T.: Management of executable schema mappings for XML data exchange, *Database Technologies for Handling XML Information on the Web, EDBT 2006 Workshops*, Lecture Notes in Computer Science **4254**, 2006, 264–277.

[15] Pankowski, T.: Reconciling inconsistent data in probabilistic XML data integration, *British National Conference on Databases (BNCOD) 2008*, Lecture Notes in Computer Science **5071**, 2008, 75–86.

[16] Pankowski, T., Cybulka, J., Meissner, A.: Reasoning About XML Schema Mappings in the Presence of Key Constraints and Value Dependencies, *Web Reasoning and Rule Systems (RR 2007)*, Lecture Notes in Computer Science **4524**, 2007, 374–376.

[17] Pankowski, T., Cybulka, J., Meissner, A.: XML Schema Mappings in the Presence of Key Constraints and Value Dependencies, *ICDT 2007 Workshop EROW'07*, CEUR Workshop Proceedings Vol. 229, CEUR-WS.org,Vol.229, 2007, 1–15.

[18] Staworko, S., Chomicki, J.: Validity-Sensitive Querying of XML Databases, *Database Technologies for Handling XML Information on the Web, EDBT 2006 Workshops*, Lecture Notes in Computer Science **4254**, 2006, 164–177.

[19] Taylor, N. E., Ives, Z. G.: Reconciling while tolerating disagreement in collaborative data sharing, *SIGMOD Conference*, ACM, 2006, 13–24.

[20] XML Path Language (XPath) 2.0: 2006. www.w3.org/TR/xpath20

[21] Xu, W., Özsoyoglu, Z. M.: Rewriting XPath Queries Using Materialized Views, *Int. Conference on Very Large Data Bases, 2005*, 2005, 121–132.

[22] Yu, C., Popa, L.: Constraint-Based XML Query Rewriting For Data Integration., *SIGMOD Conference*, 2004, 371–382.