Control and Cybernetics

vol. **38** (2009) No. 1

Schema mapping and query reformulation in peer-to-peer XML data integration system*

by

Tadeusz Pankowski

Institute of Control and Information Engineering
Poznań University of Technology, Poland
Faculty of Mathematics and Computer Science
Adam Mickiewicz University, Poznań, Poland

Abstract: In the paper we discuss the problem of XML data integration in a P2P environment. In this 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 the 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, and query reformulation in such scenario. We discuss a strategy that allows us to obtain a complete answer with the possibility of discovering missing values. The method is the theoretical foundation of the implementation of SixP2P system (*Semantic Integration of XML data in P2P environment*).

Keywords: semantic data integration, XML schema mapping, XML functional dependencies, query reformulation, query answering in peer-to-peer systems.

1. Introduction

Information integration and data exchange play central role in building of large scale systems of P2P data management and a new generation of internet based applications such as e-commerce systems, scientific grids or EAI (*Enterprise Application Integration*) and EII (*Enterprise Information Integration*) systems (see Calvanese et al., 2004, Bernstein et al., 2002, Haas, 2007, Tatarinov and Halevy, 2004). In such systems user is interested in accessing data available at many different peers in many different schemas (see Arenas and Libkin, 2005, Madhavan and Halevy, 2003, Rahm and Bernstein, 2001, Yu and Popa, 2004). To overcome syntactic heterogeneity, schema mappings are used to specify how

*Submitted: June 2008; Accepted: October 2008.

data structured under one schema (the source schema) can be transformed into data structured under another schema (the target schema) (see Fagin et al., 2004, Fuxman et al., 2006). The user issues queries against an arbitrarily chosen peer and expects the answer to 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 (see Madhavan and Halevy, 2003, Tatarinov and Halevy, 2004).

Contributions. This paper describes a formal framework underlying a class of XML data integration systems. The main contributions of the paper are:

- **Formal framework:** XML schemas are represented by means of a class of tree-pattern formulas. This formalism is used to specify schema mappings, schema constraints (functional dependencies), and queries. Some formal properties have been proven, in particular we show the relationship between functional dependencies defined over a schema and the strategy of propagation of queries and merging answers. This is significant both for the amount of information in the answer (some missing data can be discovered) and efficiency of query evaluation.
- **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 and discovering missing data.

Related work. Commonly, source-to-target dependencies were used to specify schema mappings, where both the source and target schemas are relational (see Abiteboul, Hull and Vianu, 1995, Fagin et al., 2002, 2004, Fuxman et al., 2006, Xiao and Cruz, 2006). In Arenas (2005) the relational source-to-target dependencies were adapted to XML data exchange problem by replacing relational atoms with tree-pattern formulas. We follow this approach and enrich it significantly, by adding schema constraints (XML functional dependencies) (see Arenas, 2006). There is an extensive literature on P2P data integration (see Halevy, 2005, Koloniari and Pitoura, 2005, Koubarakis et al. 2003, Tatarinov and Halevy, 2003, 2004), where some applications and optimization issues are discussed.

In this paper we are mainly interested in the impact of the relationship between schema constraints and the query on the way of query execution (i.e. query propagation and merging answers delivered by interrogated peers). In data integration, one can face the problem of inconsistent data (see Arenas, Bertossi and Chomicki, 2003, Chomicki, 2007, Staworko, Chomicki and Marcinkowski, 2006). Then, a process of *repairing* can be performed (see Arenas et al., 2005, Staworko, Chomicki and Marcinkowski, 2006). We propose a process of discovering missing values (denoted by null) which is a way for increasing both

the information content of the answer (see Pankowski, 2006) and the quality of the data integration system.

The paper is organized as follows. Section 2 introduces a running example and gives motivation of the research. Basic definitions of XML schemas and XML trees are introduced in Section 3. Schema mappings are discussed in Section 4. In Section 5 schema constraints, functional dependencies and keys are investigated. Queries and query reformulation, and some problems related to merging are considered in Sections 6 and 7, respectively. In Section 8 we describe some solutions implemented in SixP2P system. Section 9 concludes the paper.

2. XML schemas, schema constraints and XML trees

In this section we introduce some fundamental notions used in this paper and relevant to query answering in XML data integration systems. We assume that XML data are stored in local data repositories located on autonomous peers. We will focus on properties of data (schemas and constraints), and – in the following sections – also on schema mappings and queries. Problems concerning peers and communications between peers are beyond the scope of this paper.

In Fig. 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, enable us to specify schemas in a form of tree-pattern formulas.

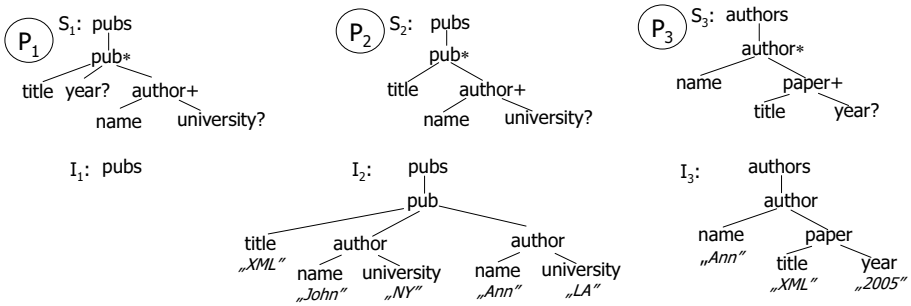

Figure 1. XML schema trees S_1, S_2, S_3, and their instances I_1, I_2 and I_3, located in peers P_1, P_2, and P_3

2.1. XML schemas

In this paper an *XML schema* (a *schema* for short) will be understood as a *tree-pattern formula* (see Arenas and Libkin, 2005, Pankowski, Cybulka and Meissner , 2007, Xu, 2005). 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:*

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

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

Schemas in the above definition are fragments of XPath 2.0 predicates (see XPath 2.0, 2006) of the class $XP^{\{/,[],=,var\}}$. These fragments consist of label tests, child axes ($/$), branches ([]), equality symbol ($=$), and variables.

EXAMPLE 1 *The schema S_1 in Fig. 1 can be specified as follows:*

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

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_3 is*

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

2.2. XML functional dependencies

An important class of schema constraints consists of *XML functional dependencies* (XFD).

DEFINITION 3 *An XML functional dependency (XFD) over a set L of labels and a set $\mathbf{x}$ of variables is an expression with the syntax:*

$$\begin{array}{lcl} fd & ::= & /P[C]/.../P[C], \\ P & ::= & l \mid P/P, \\ C & ::= & TRUE \mid P = x \mid C \wedge ... \wedge C, \end{array} \tag{2}$$

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

In the same way as for schemas, we define types of variables for XFD. If

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

is an XFD, then $type_{fd}(x_j) = /P_1/.../P_i/P_{ij}$. Additionally, we assume $type(fd) = /P_1/.../P_n$.

We will say that an XFD fd over L' and $\mathbf{x}'$, is defined on a schema S over L and $\mathbf{x}$, if:

- $\mathbf{x}' \subseteq \mathbf{x}$, $L' \subseteq L$,
- $type_{fd}(x) = type_S(x)$, for each $x \in \mathbf{x}'$,
- $type(fd)$ is a path in S.

Let p_i be the type of variable $x_i \in \mathbf{x}'$, $p_i = type(x_i)$, and $p = type(fd)$. Then the fd can be written in the form (see Arenas, 2006)

$$\{p_1, ..., p_n\} \rightarrow p$$

and it can be said that a tuple of text values $(a_1, ..., a_n)$, where a_i is the text value of the path p_i, $1 \leq i \leq n$, uniquely determines the text value of the path$p$.

Example 3 *XFD over S_3 is*

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

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

2.3. XML trees

An XML database consists of a set of XML data. We define XML data as an unordered rooted node-labeled tree (called XML tree) over a set L of labels (used to label element nodes), a set $Str \cup \{\perp\}$ of strings, and the distinguished null value $\perp$ (used as values of text nodes).

Definition 4 *An XML tree I is a tuple $(r, N^e, N^t, child, \lambda, \nu)$, where:*

- *r is a distinguished root node, N^e is a finite set of element nodes, and N^t is a finite set of text nodes;*
- *$child \subseteq (\{r\} \cup N^e) \times (N^e \cup N^t)$ – a relation introducing tree structure into $\{r\} \cup N^e \cup N^t$; the root has exactly one child (the top element); each element node must have a child;*
- *$\lambda : N^e \rightarrow L$ – a function labeling element nodes;*
- *$\nu : N^t \rightarrow Str \cup \{\perp\}$ – a function labeling text nodes with text values from Str or with the null value $\perp$.*

In order to define a relation of satisfaction between XML trees and schemas we need the notion of *variable valuation*. A valuation ω of variables from a set $\mathbf{x}$ to the set $Str \cup \{\perp\}$ is a function assigning values from $Str \cup \{\perp\}$ to variables in $\mathbf{x}$, i.e. $\omega : \mathbf{x} \rightarrow Str \cup \{\perp\}$.

By Ω a set of valuations will be denoted. A pair (S, ω) denotes the formula (sentence) created from S by replacing any occurrence of variable x with its value $\omega(x)$.

DEFINITION 5 *Let S be a schema over $\mathbf{x}$, and ω be a valuation for variables in $\mathbf{x}$. An XML tree I satisfies S by valuation ω, denoted $I \models (S, \omega)$, if the root r of I satisfies S by valuation ω, denoted $(I, r) \models (S, \omega)$, where:*

1. $(I, r) \models (/l[E], \omega)$, *iff* $\exists n \in N^e\ child(r, n) \wedge (I, n) \models (l[E], \omega)$;
2. $(I, n) \models (l[E], \omega)$, *iff* $\lambda(n) = l$ *and* $\exists n' \in N^e(child(n, n') \wedge (I, n') \models (E, \omega))$;
3. $(I, n) \models (l = x, \omega)$, *iff* $\lambda(n) = l$ *and* $\exists n' \in N^t(child(n, n') \wedge \nu(n') \sqsubseteq_v \omega(x))$, *where the subsumption relation $\sqsubseteq_v$ on values is defined as follows:* $a' \sqsubseteq_V a \Leftrightarrow a' = a \vee a = \perp$;
4. $(I, n) \models (E_1 \wedge ... \wedge E_k, \omega)$, *iff* $(I, n) \models (E_1, \omega) \wedge \cdots \wedge (I, n) \models (E_k, \omega)$.

In Definition 5 we use the subsumption relation $\sqsubseteq_v$ instead of the regular equality symbol $=$. In the rule 3 we assume that the relation of satisfaction holds also when the text value in XML tree is *subsumed by* (is *more specific* than) the value determined by the valuation. For example, for I_3 and S_3, we have:

$$I_3 \models (S_3(x_1, x_2, x_3), (Ann, XML, 2005)),$$

but also

$$I_3 \models (S_3(x_1, x_2, x_3), (Ann, XML, \perp)).$$

For a set Ω of valuations, we assume

$$I \models (S, \Omega) \Leftrightarrow \forall \omega \in \Omega (I \models (S, \omega)), \tag{3}$$

i.e. an XML tree satisfies Ω if it satisfies any valuation ω in Ω. If Ω is the maximal set of valuations, which is satisfied by I, then it is said to be the *description* for I.

The value of a path expression exp in an XML tree I, denoted by $[\![exp(\mathbf{x})(I)]\!]$, is a sequence of strings, or a sequence of nodes. We assume that evaluation of exp in I is performed according to the XPath semantics (see XPath 2.0, 2006, Wadler, 2000, Gottlob, Koch and Pichler, 2002).

DEFINITION 6 *An XFD constraint $fd(x_1, ..., x_k)$ holds in an XML tree I, or I satisfies $fd(x_1, ..., x_k)$, denoted $I \models fd(x_1, ..., x_k)$, if the following formula holds*

$$\forall x_1, ..., x_k, x, x' (x \in [\![fd(x_1, ..., x_k)(I)]\!] \wedge x' \in [\![fd(x_1, ..., x_k)(I)]\!] \Rightarrow x = x').$$

3. Schema mappings

The key issue in data integration is the one 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) (see Abiteboul, Hull and Vianu, 1995) are usually used to express schema mappings (see Fagin et al., 2004).

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 (see Arenas and Libkin, 2005, Pankowski, Cybulka and Meissner, 2007).

DEFINITION 7 *A mapping from a source schema S to a target schema T is an expression of the form*

$$m := \forall \mathbf{x}(S(\mathbf{x}) \Rightarrow \exists \mathbf{y} T(\mathbf{x}', \mathbf{y})), \tag{4}$$

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

Further on, the quantifications within mappings will be often omitted. Mappings are special cases of queries (see Definition 11), where the query qualifier is $TRUE$. The result of a mapping is the canonical instance of the right-hand side schema, where each variable $y \in \mathbf{y}$ has the $\perp$ (null) value.

The semantics of mappings is defined as follows:

DEFINITION 8 *Let I be an instance of a source schema S and Ω be the description of I. The result of a mapping $m := S(\mathbf{x}) \Rightarrow T(\mathbf{x}', \mathbf{y})$ is an instance J of T such that its description Ω' is*

$$\Omega' = \{m(\omega) \mid \omega \in \Omega\},$$

where the valuation $m(\omega) = (\omega[\mathbf{x}'], \perp_{\mathbf{y}})$ is composed of: $\omega[\mathbf{x}']$ – the restriction of the valuation ω to variables in $\mathbf{x}'$, and $\perp_{\mathbf{y}}$ – the valuation assigning $\perp$ to each variable $y \in \mathbf{y}$. Then we write $J = m(I)$ and $\Omega' = m(\Omega)$.

EXAMPLE 4 *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 $I_3 \models (S_3(x_1, x_2, x_3), (Ann, XML, 2005))$,
$m_{31}(x_1, x_2, x_3, x_4)(x_1 : Ann, x_2 : XML, x_3 : 2005) =$
$(x_1 : Ann, x_2 : XML, x_3 : 2005, x_4 : \perp)$. *Thus, a valuation has been created, which is satisfied by an instance J of schema S_1, i.e.*
$J \models (S_1(x_2, x_3, x_1, x_4), (x_1 : Ann, x_2 : XML, x_3 : 2005, x_4 : \perp))$.

In SixP2P, mappings are implemented by means of XQuery programs. For a mapping m_{ik} the corresponding program is generated by Algorithm 1.

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: $S_i := /l'[E']$, $S_k := /l[E]$, $\mathbf{y} = (y_1, ..., y_m)$.

Output: Query in XQuery over S_i transforming an instance of S_i into the corresponding canonical instance of S_k.

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

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

where:

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. $\rho(l = x) = <l>\{\$x\}</l>$
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), the XQuery program generated by Algorithm 1 is:

```
Query 1:
<pubs>{
  let x_4:="null"
  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>{$x_4}</university>
      </author>
  </pub> }
</pubs>
```

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

4. Operations on XML trees

In XML data integration, XML trees are first transformed into trees of a common schema, and then two trees with the same schema are merged (see Pankowski et al., 2005). The transformation can be performed using schema mappings, as was described in the previous section. Merging consists in many operations which include *union*, *repairing* inconsistencies (see Greco et al., 2003), *duplicate* discovery and elimination, and finally *grouping and nesting* with respect to given keys (see Pankowski, Cybulka and Meissner, 2007). In this section we will focus on the two first of these operations, union and a simple case of repairing that will be understood as *discovering missing values*, i.e. we will discuss a method for replacing some null values with non-null values. The process of repairing will be controlled by a given set of XML functional dependencies.

4.1. Union of XML trees

DEFINITION 9 *Let*
$J_1 = (r_1, N_1^e, N_1^t, child_1, \lambda_1, \nu_1)$, *and*
$J_2 = (r_2, N_2^e, N_2^t, child_2, \lambda_2, \nu_2)$
be two XML trees satisfying a schema S. *The XML tree*
$J = (r, N^e, N^t, child, \lambda, \nu)$
is called the union of J_1 *and* J_2, *denoted* $J = J_1 \cup J_2$, *if there are two functions (embeddings)* $h_1 : J_1 \to J$, *and* $h_2 : J_2 \to J$ *such that:*

- $h_1(r_1) = h_2(r_2) = r$ – *root nodes* r_1 *and* r_2 *are mapped to the root node* r;
- $h_1(top_1) = h_2(top_2) = top$, *where* $child_1(r_1, top_1)$, $child_2(r_2, top_2)$, *and* $child(r, top)$, *i.e. the outermost elements (top elements) of* J_1 *and* J_2 *are mapped to the outermost element of* J;
- $h_1(N_1^e) \cup h_2(N_2^e) = N^e$ – *union of images of* N_1^e *and* N_2^e *is equal to* N^e;
- $h_1(N_1^e) \cap h_2(N_2^e) = \{top\}$ – *the only common element node in images of* N_1^e *and* N_2^e *is* $top \in N^e$;
- $h_1(N_1^t) \cup h_2(N_2^t) = N^t$ – *union of images of* N_1^t *and* N_2^t *is equal to* N^t;
- $h_1(N_1^t) \cap h_2(N_2^t) = \emptyset$ – *images of* N_1^t *and* N_2^t *are disjoint;*
- $h_i(child_i(n, n')) = child(h_i(n), h_i(n'))$, $i = 1, 2$ – *the embeddings preserve childhood relationships;*
- $h_i(\lambda_i(n)) = \lambda(h_i(n))$, $i = 1, 2$ – *the embeddings preserve labeling;*
- $h_i(\nu_i(n)) = \nu(h_i(n))$, $i = 1, 2$ – *the embeddings preserve text values.*

The union of J_1 and J_2 is illustrated in Fig. 2 (root nodes are, as usual, not depicted in the figure).

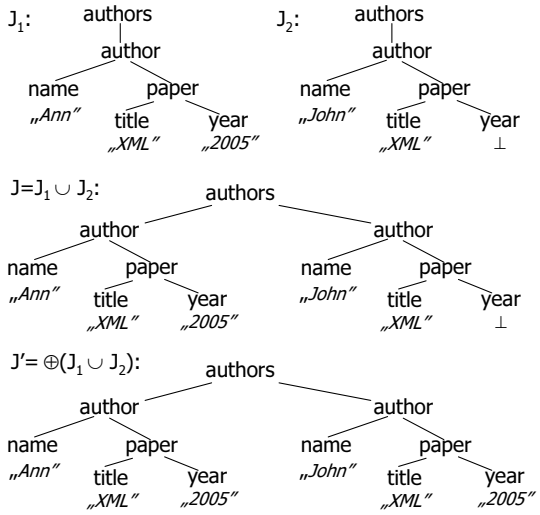

Figure 2. Illustration of merging two instances J_1 and J_2 with discovering missing values

LEMMA 1 *If* $J \models (S, \Omega)$ *and* $J' \models (S, \Omega')$ *then* $J \cup J' \models (S, \Omega \cup \Omega')$.

EXAMPLE 5 *The XML tree* J *in Fig. 2 is a tree with description* $\Omega = \Omega_1 \cup \Omega_2$, *and we write:*

$$J = J_1 \cup J_2,$$
$$J \models (S_3, \Omega), \text{ where}$$
$$\Omega = \Omega_1 \cup \Omega_2 = \{(Ann, XML, 2005), (John, XML, \perp)\}.$$

The above union operation, $\cup$, is defined on two XML trees of the same schema. The description of the union is the union of descriptions of both arguments.

4.2. Repairing inconsistent XML trees

In general, in data integration systems (especially in P2P data management, see Madhavan and Halevy, 2003) violations of consistency constraints cannot be avoided (see Greco et al., 2003, Staworko, Chomicki and Marcinkowski, 2006). Data could violate consistency constraints defined over the target schema, although they satisfy constraints defined over source schemas considered in separation. An XML tree that does not satisfy an integrity constraint will be referred

to as *inconsistent* XML tree. In this paper we discuss a class of *weak inconsistencies*, in which XML functional dependencies are violated, and all violations are caused by null values. So, we assume that non-null values are consistent.

For example, both J_1 and J_2 in Fig. 2 satisfy the XFD from Example 3, but its union, $J = J_1 \cup J_2$, does not. We see that there are two different values of *year*, 2005 in the first subtree and $\perp$ in the second, for the same value XML of *title*.

Now, we can define the repairing operation over a set Ω of valuations with respect to an XML constraint fd, denoted by $\oplus_{fd}$. The XFD will be used to discover missing data.

DEFINITION 10 *Let $S(\mathbf{x})$ be a schema, $fd(\mathbf{y})$ be an XFD over $S(\mathbf{x})$, $type(fd) = type(z)$, and I be an XML tree such that $I \models (S(\mathbf{x}), \Omega)$. An XML tree I' is called the repairing of I with respect to fd, denoted $I' = \oplus_{fd}(I)$, if*

- $I' \models (S(\mathbf{x}), \Omega')$, *and*
- $\Omega' = \oplus_{fd}(\Omega)$, *where*

$$\begin{aligned}\oplus_{fd}(\Omega) = \{ \quad & \omega \mid \exists \omega_1 \in \Omega \wedge \\ & \textbf{if } \omega_1(z) = \perp \ \wedge \exists \omega_2 (\omega_1(\mathbf{y}) = \omega_2(\mathbf{y}) \wedge \omega_2(z) \neq \perp) \\ & \textbf{then } \omega = \omega_1[z \mapsto \omega_2(z)] \\ & \textbf{else } \omega = \omega_1\},\end{aligned}$$

where $\omega_1[z \mapsto \omega_2(z)]$ is the valuation ω identical to ω_1 except for $\omega(z) = \omega_2(z)$.

The repairing operation can be easily generalized to an arbitrary set of XFDs. If $F = \{fd_1, ..., fd_m\}$ is a set of XFDs, then:

$$\begin{aligned}&\oplus_F(I) = \oplus_{fd_m}(... \oplus_{fd_1} (I)...), \\ &\oplus_F(\Omega) = \oplus_{fd_m}(... \oplus_{fd_1} (\Omega)...),\end{aligned} \tag{5}$$

and $\oplus_F(I)$ and $\oplus_F(\Omega)$ are called *repairing* of I and Ω with respect to F, respectively.

The following Algorithm 2 generates an XQuery program for a given schema S and a set F of XFD constraints over this schema. For every XML tree of schema S the program produces the repairing of the tree with respect to F.

Algorithm 2 (*generating XQuery program performing the repairing operation*)

Input: A schema S and a set of XFD constraints, $xfd(x)$ denotes the XFD of the type equal to the type of variable x.

Output: Program in XQuery over instances of S returning for a given instance the repaired version of these instances.

$xfdToXQuery(/top[E])$.

The translation function

$xfdToXQuery()$

is identical to the translation function in Algorithm 1
$mappingToXQuery()$,
except that the rule (4) is replaced by the following rule (4'):

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

EXAMPLE 6 *Discovering missing values in an instance of S_1 (Example 1) can be done using the XQuery program generated for the schema S_1, where XFD constraints for S_1, i.e. $fd(x_2)$, and $fd(x4)$ (Example 3), are taken into account. The corresponding XQuery program is similar to Query 1. However, expressions defining the elements "year" and "university" have the following forms:*

```
Query 2:
...
<year>{
  if ($x2="null") then
    string((/pubs/pub[title=$x1]/year[text()!="null"])[1]) else $x2}
</year>
<university>{
  if ($x4="null") then
  string((/pubs/pub/author[name=$x3]/university[text()!="null"])[1])
         else $x4}
</university>
...
```

As the result of repairing, all nulls violating XFDs are replaced by certain non-null values.

5. Queries and query reformulation

5.1. Queries

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* over S.

DEFINITION 11 *Let $\forall \mathbf{x}(S(\mathbf{x}) \Rightarrow \exists \mathbf{y} T(\mathbf{x}', \mathbf{y}))$ be a schema mapping from S to T and $\phi(\mathbf{x})$ be a qualifier over S. A query from S to T with qualifier ϕ is an expression of the form*

$$q := \forall \mathbf{x}(S(\mathbf{x}) \wedge \phi(\mathbf{x}) \Rightarrow \exists \mathbf{y} T(\mathbf{x}', \mathbf{y})), \tag{6}$$

or, for short: $q := S \wedge \phi \Rightarrow T$. If the target schema is clear from the context, we will simply write: $q := S \wedge \phi$.

An answer to a query is defined as follows:

DEFINITION 12 *Let I be an instance of a source schema S and Ω be the description of I. An answer to a query $q := S \wedge \phi \Rightarrow T$ is such an instance J of T that its description Ω' is defined as:*

$$\Omega' = \{m(\omega) \mid \omega \in \Omega \wedge \phi(\omega) = true\}, \tag{7}$$

where m is the mapping, $m := S \Rightarrow T$.

EXAMPLE 7 *The following query*

$$\begin{aligned} q_{12} := \quad & S_1 \wedge \phi \Rightarrow S_2 \\ & := /pubs[pub[title = x_1 \wedge year = x_2 \\ & \quad \wedge author[name = x_3 \wedge university = x_4]]] \\ & \quad \wedge x_3 = "John" \wedge x_2 = "2005" \\ & \Rightarrow /pubs[pub[title = x_1 \wedge author[name = x_3 \\ & \quad \wedge university = x_4]]] \end{aligned}$$

filters an instance of the source schema S_1 according to the qualifier $x_3 = "John" \wedge x_2 = "2005"$, and produces an instance of the schema S_2.

5.2. Query reformulation

Assume that a query $q_i := S_i(\mathbf{x_i}) \wedge \phi_i(\mathbf{z_i})$ is issued against a peer P_i. If the query is propagated to a peer P_k then it must be reformulated into such a query q_{ki} which can be evaluated over data stored on the peer P_k and the answer must be structured according to the schema S_i.

1. We want to determine the qualifier ϕ_k in the query
   $$q_{ki} := S_k(\mathbf{x_k}) \wedge \phi_k(\mathbf{z_k}) \Rightarrow S_i(\mathbf{x_{ik}}, \mathbf{y_{ik}}),$$
   over the source schema S_k, where $\mathbf{z_k} \subseteq \mathbf{x_k}$, $\mathbf{x_{ik}} \subseteq \mathbf{x_k}$, $\mathbf{y_{ik}} \cap \mathbf{x_k} = \emptyset$.
2. The qualifier $\phi_k(\mathbf{z_k})$ is obtained as the result of rewriting 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.

EXAMPLE 8 *For the query qualifier*

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

over $S_1(x_1, x_2, x_3, x_4)$ (Example 1), we have the following reformulation over $S_2(x_1, x_2, x_3)$ (Example 1) with respect to the mapping m_{21} (Example 4):

$$\begin{aligned} & \phi_1(x_3).rewrite(S_1(x_1, x_2, x_3, x_4), S_{12}(x_1, x_2, x_3) \\ & = \phi_1(x_2) := x_2 = "John", \end{aligned}$$

since $type_{S_1(x_1,x_2,x_3,x_4)}(x_3) = type_{S_{12}(x_1,x_2,x_3)}(x_2) = /pubs/pub/author/name$.

Note that the qualifier over S_1 (Example 7)

$$\phi_1(x_2, x_3) := x_2 = \text{"2005"} \wedge x_3 = \text{"}John\text{"}$$

is non-rewritable over S_2 with respect to m_{21} and is rewritable over S_3 with respect to m_{31}. In the latter case we obtain:

$$\phi_3(x_3, x_1) := x_3 = \text{"2005"} \wedge x_1 = \text{"}John\text{"}.$$

Algorithm 3 translates a query q_{ij} into XQuery. The algorithm is a slight modification of Algorithm 1, where the clause **where** is added.

Algorithm 3 (*translating query to XQuery program*)
Input: A query $q_{ik} := \forall \mathbf{x}(S_i \wedge \phi_i \Rightarrow \exists \mathbf{y} S_k)$, where:
$S_i := /l'[E']$, $S_k := /l[E]$, $\mathbf{y} = (y_1, ..., y_m)$.
Output: Query in XQuery over S_i returning the
answer conforming to S_k when applied to an instance of S_i

$queryToXQuery(\forall \mathbf{x}(/l'[E'] \wedge \phi_i \Rightarrow \exists y_1, ..., y_m / l[E]))=$

```
<l>{
   let $y1 := "null", ..., $ym := "null"
   for $v in /l',
     τ(v, E')
     where φ
   return
     ρ(E)}
</l>
```

where functions τ and ρ are defined in Algorithm 1.

---

6. Deciding about merging modes

Answers to a query propagated across the P2P systems must be collected and merged. The merge operation on an arbitrary set of instances with the same schema, is defined as follows:

DEFINITION 13 *Let S be a schema, F be a set of XFD over S, and $I_1, ..., I_k$ be XML trees of schema S. The merging of $I_1, ..., I_k$ with respect to F is understood as the repairing (Definition 10) of the union $I_1 \cup ... \cup I_k$, i.e.*

$$\sqcup_F(\{I_1, ..., I_k\}) = \oplus_F(I_1 \cup ... \cup I_k). \tag{8}$$

In the definition of merging, we incorporate the discovery of missing values. To do this operation, it is to be decided, which of the two merging modes should be selected in the peer while partial answers are to be merged:

- *partial merging* – all partial answers are merged without taking into account the source instance stored in the peer,

- *full merging* – the whole source instance in the peer is merged with all received partial answers; finally, the query is evaluated in terms of the result of the merging.

Criterion of 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.

Theorem 1 states the condition when there is no sense in applying full merging because no missing value can be discovered.

THEOREM 1 *Let $S(\mathbf{x})$ be a schema, $fd(\mathbf{z})$ be an XFD over $S(\mathbf{x})$, and $type(fd) = type_S(x)$ for some $x \in \mathbf{x}$. Let 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. Then the equality*

$$q(\sqcup_{fd}\{I, I_A\}) = \sqcup_{fd}\{q(I), I_A\} \tag{9}$$

holds if one of the following two conditions holds
(a) $x \in \mathbf{y}$, *or*
(b) $\mathbf{z} \subseteq \mathbf{y}$.

Proof. The equality (9) 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 $\omega'(\mathbf{z}) = \omega(\mathbf{z})$. 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 (9) 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, (9) holds. ■

7. Query execution strategies

In this section we show how Theorem 9 can be used to control query execution in P2P environment. Based on this theorem we can decide about propagating queries and merging answers. 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 (see Brzykcy, Bartoszek and Pankowski, 2007). It turns out that the chosen strategy and the way of merging partial answers determine both the final answer and the cost of execution. We focus mainly on the amount of information provided by the answer when the merging procedure involves discovering of missing values.

Let us consider some possible strategies of execution of the following query q against P_1 (Fig. 1) (the result of the query should be structured according to

its source schema, so we do not specify this explicitly):

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

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 target schema is by default equal to the source schema. The answer to the query should contain information stored in all three sources shown in Fig. 1.

Thus, one of the following three strategies can be realized (Fig. 3):

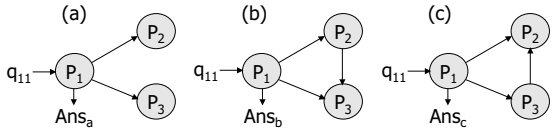

Figure 3. Three different execution strategies of the query q

Strategy (a). Query q is sent to P_2 and P_3, where it is reformulated to, respectively, q_{21} (from S_2 to S_1) and q_{31} (from S_3 to S_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. We use the set $F1 = \{fd_1, fd_2\}$ of XFDs, where: $fd_1 := /pubs/pub[title = x_1]/year$, $fd_2 := /pubs/pub/author[name = x_3]/university$. Then

$$\begin{aligned}
Ans_a &= \sqcup_{F1}\{Ans^a_{11}, Ans^a_{21}, Ans^a_{31}\},\\
Ans^a_{11} &= q_{11}(I_1) = \{(x_{title} : \perp, x_{year} : \perp, x_{name} : \perp, x_{univ} : \perp)\},\\
Ans^a_{21} &= q_{21}(I_2) = \{(x_{title} : XML, x_{name} : John, x_{univ} : NY)\},\\
Ans^a_{31} &= q_{31}(I_3) = \{(x_{name} : \perp, x_{title} : \perp, x_{year} : \perp)\},\\
Ans_a &= \{(x_{title} : XML, x_{year} : \perp, x_{name} : John, x_{univ} : NY)\}.
\end{aligned}$$

In this strategy the value of x_{year} is $\perp$ and we cannot discover it, although it is implicitly included in I_3.

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 $q_{32}(I_3)$. The result is equal to Ans_a ($F2 = \{fd_2\}$):

$$\begin{aligned}
Ans_b &= \sqcup_{F2}\{Ans^b_{11}, Ans^b_{21}, Ans^b_{31}\} =\\
&= \{(x_{title} : XML, x_{year} : \perp, x_{name} : John, x_{univ} : NY)\}.
\end{aligned}$$

It is easily seen that this strategy leads to the same result as the strategy (a) but its cost is higher. We see that involving I_2 in the expression for computing Ans^b_{21} is not justified – it only degrades performance.

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 of its instance I_3. The decision is based on the existence of the functional dependency $fd_3 = /authors/author/paper[title = x_2]/year$. Let $F3 = \{fd_3\}$, then

$$\begin{array}{lcl} Ans_c & = & \sqcup_{F1}\{Ans^c_{11}, Ans^c_{21}, Ans^c_{31}\}), \\ Ans^c_{23} & = & q_{23}(I_2) = \{(x_{title} : XML, x_{year} : \perp, x_{name} : John)\}, \\ Ans^c_{31} & = & q_{31}(\sqcup_{F3}\{I_3, Ans^c_{23}\}) = \\ & = & \{(x_{title} : XML, x_{year} : 2005, x_{name} : John)\} \\ Ans_c & = & \{(x_{title} : XML, x_{year} : 2005, x_{name} : John, \\ & & \quad x_{univ} : NY)\}. \end{array}$$

While computing the merger $\sqcup_{F3}\{I_3, Ans^c_{23}\}$ a missing value of x_{year} is discovered. Thus, the answer Ans_c provides more information than Ans_a and Ans_b.

8. Conclusions

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 are to be discovered in the P2P integration processes. We showed how to use schema constraints, mainly functional dependency constraint, to select the way of query propagation and data merging, to increase the information content of the answer to a query.

Acknowledgment

The work was supported in part by the Polish Ministry of Science and Higher Education under Grant N516 3695/36.

References

Abiteboul, S., Hull, R. and Vianu, V. (1995) *Foundations of Databases*. Addison-Wesley, Reading, Massachusetts.

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

Arenas, M., Bertossi, L.E. and Chomicki, J. (2003) Answer sets for consistent query answering in inconsistent databases. *Theory and Practice of Logic Programming* **3** (4-5), 393–424.

Arenas, M. and Libkin, L. (2005) XML Data Exchange: Consistency and Query Answering. *Proceedings of the 24th ACM Symposium on Principles of Database Systems, PODS '05*, Baltimore, USA. ACM Press, 13–24.

Bernstein, P.A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L. and Zaihrayeu, I. (2002) Data management for peer-to-peer computing: A vision. *Proceedings of the 5th Workshop on the Web and Databases, Web DB 2002*, Madison, USA. ACM Press, 89–94.

Brzykcy, G., Bartoszek, J. and Pankowski, T. (2007) Semantic data integration in p2p environment using schema mappings and agent technology. *The 1st KES Symposium on Agent and Multi-Agent Systems – Technologies and Applications.* **LNCS 4496**, Springer, 385–394.

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

Chomicki, J. (2007) Consistent query answering: Five easy pieces. In: T. Schwentick and D. Suciu, eds., *Proceedings of the 12th International International Conference on Database Theory.* **LNCS 4353**, Springer, 1–17.

Fagin, R., Kolaitis, P.G., Miller, R.J. and Popa, L. (2002) Data Exchange: Semantics and Query Answering. *ICDT 2003.* **LNCS 2572**, Springer, 207–224.

Fagin, R., Kolaitis, P.G., Popa, L. and Tan, W.C. (2004) Composing schema mappings: Second-order dependencies to the rescue. *Proceedings of the 23th ACM SIGMOD Symposium on Principles of Database Systems (PODS 2004)*, Paris, France. ACM Press, 83–94.

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

Gottlob, G., Koch, C. and Pichler, R. (2002) Efficient algorithms for processing XPath queries. *Proc. of the 28th International Conference on Very Large Data Bases, VLDB 2002*, Hong Kong, China. Morgan Kaufmann, 95–106.

Greco, S., Sirangelo, C., Trubitsyna, I. and Zumpano, E. (2003) Preferred repairs for inconsistent databases. *7th International Database Engineering and Applications Symposium – IDEAS 2003.* IEEE Computer Society, 202–211.

Haas, L.M. (2007) Beauty and the beast: The theory and practice of information integration. *Proceedings of the 12th International Conference on Database Theory.* **LNCS 4353**, Springer, 28–43.

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

Koloniari, G. and Pitoura, E. (2005) Peer-to-peer management of XML data: issues and research challenges. *SIGMOD Record* **34** (2), 6–17.

Koubarakis, M., Tryfonopoulos, C., Idreos, S. and Drougas, Y. (2003) Selective information dissemination in P2P networks: problems and solutions. *SIGMOD Record* **32** (3), 71–76.

Madhavan, J. and Halevy, A.Y. (2003) Composing mappings among data sources. *Proceedings of the 29th International Conference on Very Large Data Bases, VLDB 2003*, Berlin, Germany. Morgan Kaufmann, 572–583.

Pankowski, T. (2006) Management of executable schema mappings for XML data exchange. *Database Technologies for Handling XML Information on the Web, EDBT 2006 Workshops*. **LNCS 4254**, Springer, 264–277.

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

Pankowski, T. and Hunt, E. (2005) Data merging in life science data integration systems. *Intelligent Information Systems, New Trends in Intelligent Information Processing and Web Mining. Advances in Soft Computing*, Springer Verlag, 279–288.

Rahm, E. and Bernstein, P.A. (2001) A survey of approaches to automatic schema matching. *The VLDB Journal* **10**, 4, 334–350.

Staworko, S. and Chomicki, J. (2006) Validity-Sensitive Querying of XML Databases. *Database Technologies for Handling XML Information on the Web, EDBT 2006 Workshops*. **LNCS 4254**, Springer, 164–177.

Staworko, S., Chomicki, J. and Marcinkowski, J. (2006) Preference-Driven Querying of Inconsistent Relational Databases. *Incompleteness and Inconsistency in Databases, EDBT 2006 Workshop*. **LNCS 4254**, Springer, 318–335.

Tatarinov, I. and Halevy, A.Y. (2004) Efficient query reformulation in peer-data management systems. *Proc. of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2004*, Paris, France. ACM Press, 539–550.

Tatarinov, I., Ives, Z.G., Madhavan, J., Halevy, A.Y., Suciu, D., Dalvi, N.N., Dong, X., Kadiyska, Y., Miklau, G. and Mork, P. (2003) The Piazza peer data management project. *SIGMOD Record* **32** (3), 47–52.

Wadler, P. (2000) Two semantics for XPath. Available on: http://homepages.inf.ed.ac.uk/wadler/papers/xpath-semantics/xpath-semantics.pdf, 26 July 1999, revised 4 January 2000.

Xiao, H. and Cruz, I.F. (2006) Integrating and Exchanging XML Data Using Ontologies. *Journal on Data Semantics VI: Special Issue on Emergent Semantics*. **LNCS 4090**, Springer, 67–89.

XML Path Language (XPath) 2.0 (2006) www.w3.org/TR/xpath20.

Xu, W. and Özsoyoglu, Z.M. (2005) Rewriting XPath Queries Using Materialized Views. *Proceedings of the 30th International Conference on Very Large Data Bases, VLDB 2005*, Trondheim, Norway. Morgan Kaufmann, 121–132.

Yu, C. and Popa, L. (2004) Constraint-Based XML Query Rewriting For Data Integration. *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2004*, Paris, France. ACM Press, 371–382.