INTEGRATION OF XML DATA IN PEER-TO-PEER E-COMMERCE APPLICATIONS

Tadeusz Pankowski [1,2]

[1] *Institute of Control and Information Engineering*
Poznan University of Technology
Pl. M.S.-Curie 5, 60-965 Poznan
Tadeusz.Pankowski@put.poznan.pl

[2] *Faculty of Mathematics and Computer Science*
Adam Mickiewicz University
ul. Umultowska 87, 61-614 Poznan
tpankow@amu.edu.pl

Abstract E-commerce applications need new solutions in the field of integration, interchange and transformation of data across the global marketplace. Such advanced data management features, which are expected to function automatically or semi-automatically, are necessary when a party looks for potential business partners, a buyer wants to find relevant supplier of the products, a seller wants to find potential customers or business partners negotiate a deal, and so on. In these e-commerce applications distributed systems based on the traditional client-server paradigm are nowadays replaced by peer-to-peer (P2P) systems. The goal of data management in P2P systems is to make use of a decentralized, easily extensible architecture in which any user can contribute new data or new schemas and mappings between other peer's schemas. P2P data management systems replace traditional data integration systems based on single global schema with an interlinked collection of semantic mappings between peers' individual schemas. The paper discusses this kind of P2P data management in e-commerce settings. A new proposal concerning schema mapping specification and query reformulation is presented.

Keywords: Data integration, XML databases, query reformulation, schema mapping, Peer Database Management Systems, P2P e-commerce

1. Introduction

E-commerce (*electronic commerce*) covers trading activities that are supported by variety of information and communication technologies. A new generation of e-commerce applications such as B2B (*Business-To-Business*) and EAI (*Enterprise Application Integration*) requires new standards and new tech-

nologies. Such new technologies are developed and provided among others by P2P (*Peer-to-Peer*) computing and PDMS (*Peer Data Management Systems*) [Bernstein et al., 2002; Tatarinov and Halevy, 2004]. B2B P2P e-commerce opens up new possibilities of trade, where new business partners from around the globe can be found, their offers can be compared, even complex negotiations can be conducted electronically, and a contract can be drown up and fulfilled via an electronic marketplace. Thus, market places for B2B e-commerce require integration and interoperation of several systems (e.g. product databases, order processing systems) across multiple organizations. E-commerce is carried out in a highly dynamic environment where companies enter the marketplace some others drop out. The data flow is bi-directional, i.e. data has to be transferred back to participants as well (orders, product lists). The system must be adaptable for different environments, as the electronic marketplace covers many countries with different languages and different legal regulations.

A basic functionality of a system supporting e-commerce applications is the integration of external data sources or data services. To solve the problem of data integration one can use the idea of federated database system [Sheth and Larson, 1990]. In this scenario a global business data repository is a key component of a federated system where it plays the role of a global schema (mediator): all requests by the client application are sent to the repository. The mediator then looks up its resources (metadata repository) and sends the query to the appropriate source. The result is received by the mediator and then sent back to the client application in the desired format. Such a solution was proposed in [Quix et al., 2002]. This approach is based on a centralized client-server architecture in which servers represent vendors or marketplaces and customers are represented by clients.

Nowadays and after the great success of file sharing systems, such as Napster, Gnutella and BitTorrent, another more decentralized approach referred to as Peer-to-Peer comes into use [Bernstein et al., 2002; Calvanese et al., 2004]. P2P systems are characterized by an architecture constituted by various autonomous nodes (called *peers*) which hold information, and which are linked to other nodes by means of mappings. In P2P data sharing and data integration each peer exports data in terms of its own schema, and data interoperation is achieved by means of mappings among the peer schemas. One of the challenges in these systems is answering queries posed to one peer while taking into account the mappings. This can be achieved by so called *query reformulation* [Halevy, 2001],[Lenzerini, 2002],[Madhavan and Halevy, 2003], [Pankowski, 2005],[Tatarinov and Halevy, 2004],[Yu and Popa, 2004].

A system supporting query reformulation does not need to materialize a target view over sources. Instead, a *target query* is reformulated into a set of *source queries* that can be processed in underlying sources, and partial answers are merged to obtain the final result. In both cases, the central problem is how

to describe the correspondence or *mappings* between the source data and the target data. Mappings are usually specified as high-level assertions that state how source elements correspond to target elements at schema and/or instance level. So, a distinction can be drawn between schema-level, instance-level or hybrid approaches to source-target mappings [Rahm and Bernstein, 2001]. In the case of schema mapping only schema information, not instance data, is taken into account. Schema mappings can be given manually, perhaps supported by a graphical user interface, or they can be derived semi-automatically [Miller et al., 2000; Popa et al., 2002] based on schema matching algorithms [Rahm and Bernstein, 2001]. For query reformulation in relational data integration systems, schema mappings have been defined using GAV (*global-as-view*), LAV (*local-as-view*) or GLAV (*global-and-local-as-view*) approaches [Calvanese et al., 2004; Halevy, 2001; Lenzerini, 2002; Ullman, 1997].

The main contributions of this paper are the following. We propose a method for specifying mappings between schemas of peer XML data repositories. The source-to-target mapping specification is based on Skolem functions. Any invocation $SF(x_1, ..., x_n)$ of a Skolem function SF returns the same node identifier for the same values of arguments $x_1, ..., x_n$. For different Skolem functions and for different values of arguments returned identifiers (nodes) are distinct. Arguments of a Skolem function are defined by means of path expressions returning text values. The mapping is specified as an expression of a mapping language that asserts source-to-target dependencies [Fagin et al., 2004]. Two kinds of dependencies are taken into account: *(a) source-to-target node generating dependencies*, where a Skolem function is assign to each absolute target path (i.e. a path starting from the root of a target schema) and establishes a one-to-one relationship (modulo the function name) between tuples of text values of source nodes and target nodes; and *(b) source-to-target value dependencies* that constrain relationships between leaf node text values in source and target data trees. Next, we propose *rewriting rules for query reformulation.* A query reformulation is defined as a rewriting process leading from a target query to a required source query. Both mapping specification and query reformulation are illustrated by examples relevant to e-commerce applications.

The paper is structured as follows. In Section 2 we introduce some concepts concerning XML data and XML path expressions. Then we define a method to specify schema mappings based on Skolem functions. Classes of source-to-target and value dependences are discussed and illustrated by an example. In Section 3 we propose rewriting rules for query reformulation. Application of these rules is illustrated in Section 4. Section 5 concludes the paper.

2. Specification of schema mapping

In data integration systems, there are one target (or mediated) schema and a number of source schemas. In a P2P computation, a role of the target schema can be played any peer that can be arbitrary chosen by the user. The target schema is commonly treated as a virtual view over sources, and source schemas describe real data stored in source repositories. We assume that both source and target data conform to XML format.

2.1 XML data and path expressions

An XML document is a textual representation of data and consists of hierarchically nested element structure starting with a root element. In the DOM Data Model proposed by the W3C [XQuery 1.0 and XPath 2.0 Data Model. W3C Working Draft, 2002], an XML document is represented by an ordered node-labeled tree (or *instance*) that includes a concept of node identity. Similarly, the schema of an XML document can be described by a node-labeled tree, where multiplicity qualifiers (?, +, or $*$) are assigned with nodes.

We adopt an unordered tree model for XML schemas and instances, where the leaves can be element nodes with text values. We will use a simple path language for data tree navigation and we assume traversing only along the child (/) axis. Variables will be bound to individual nodes (node identifiers). The following categories of expressions are in our path language:

$root$	::=	$@doc \mid \$x$	*an absolute (@doc) or a relative ($x) root*
P	::=	$l \mid root/l \mid P/l$	*a path*,
G	::=	$\$x$ in P	*a range*,

where $@doc$ identifies document schema tree and points to the (absolute) root of a document data tree being an instance of the schema; $\$x$ is a node variable and its value is referred to as a relative root; l is an XML node label (element tag); a path P is referred to as an absolute or a relative path depending on its starting node; G can be an absolute range (if P is an absolute path) or a relative range (if P is a relative path).

The type, $type(p)$, of a path expression p is defined as follows:

$type(p)$	=	p,	for a variable-free expression,
$type(\$x)$	=	$type(P)$,	for $\$x$ defined in a range $\$x$ in P,
$type(\$x/P)$	=	$type(\$x)/P$.	

In Figure 1 there are five XML document schemas (written as schema trees) that will be used as a running example to illustrate schema mapping and query reformulation in a data integration scenario. Along with schemas, text values of leaf nodes are also given. It is quite obvious, how from this simplified presentation, proper XML document instances should be derived. E.g. `@P3` has the instance given in Figure 2.

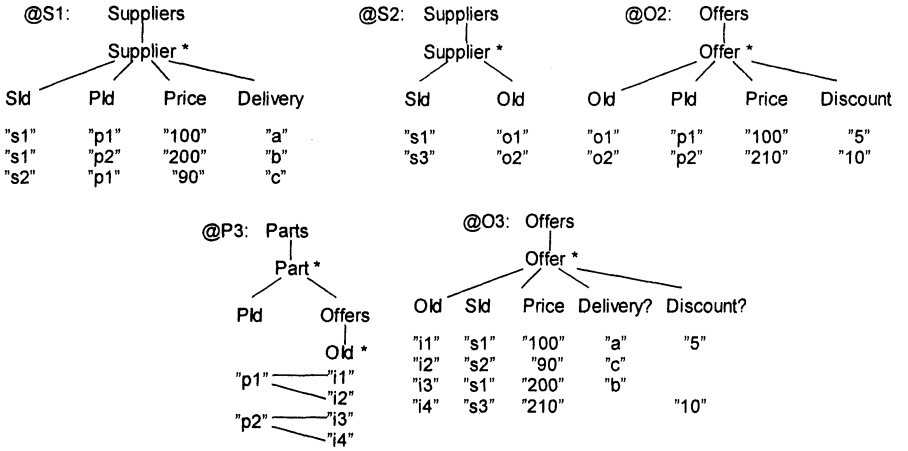

Figure 1. Schemas and leaf values of sample XML documents

```
@P3:
  <Parts>
    <Part>
      <PId>p1</PId>
      <Offers>
        <OId>i1</OId>
        <OId>i2</OId>
      </Offers>
    </Part>
    <Part>
      <PId>p2</PId>
      <Offers>
        <OId>i3</OId>
        <OId>i4</OId>
      </Offers>
    </Part>
  </Parts>
```

Figure 2. Instance of the document schema @P3 from Figure 1

All the documents represent information about suppliers and parts. Elements `SId`, `PId`, `Price`, `Delivery`, and `Discount` represent, respectively, supplier identifier, part identifier, price of a part being supplied, information about delivery time, and about discount. The attribute `OId` is used to link suppliers with their offers (in documents `@S2` and `@O2`) or to link parts with offers concerning these parts (in documents `@P3` and `@O3`).

Note that information in two sources, i.e. in (`@S1`), and in (`@S2`,`@O2`), may overlap. If an offer appears only in one source, then we can have incomplete information in the target (the lack of values for `Delivery` or `Discount`). In the rest of the paper instances of `@P3` and `@O3` will be treated as canonical virtual instances derived from real instances of `@S1`, `@S2` and `@O2`. It means that these instances are not materialized. Further on it will be assumed that there are three peers, P_1, P_2, and P_3, in a P2P system. Peers P_1 and P_2 are *source peers* that store data with schemas, respectively, {`@S1`} and {`@S2`,`@O2`}. P_3 does not store any data but provides the schema {`@P3`,`@O3`}.

2.2 Schema mapping specification based on Skolem functions

The basic idea of our approach to schema mapping is shown in Figure 3. A source schema consists of two document schemas `@S2` and `@O2`, and a target schema has also two document schemas `@P3` and `@O3` (see Figure 1). There are four mapping functions shown in Figure 3, the total number of functions is equal to the number of mapped nodes in the target schema trees, i.e. 13, 2 for root nodes (root nodes are not visible) and 11 for ordinary labeled nodes (the node `Delivery` will be not mapped for this source schema).

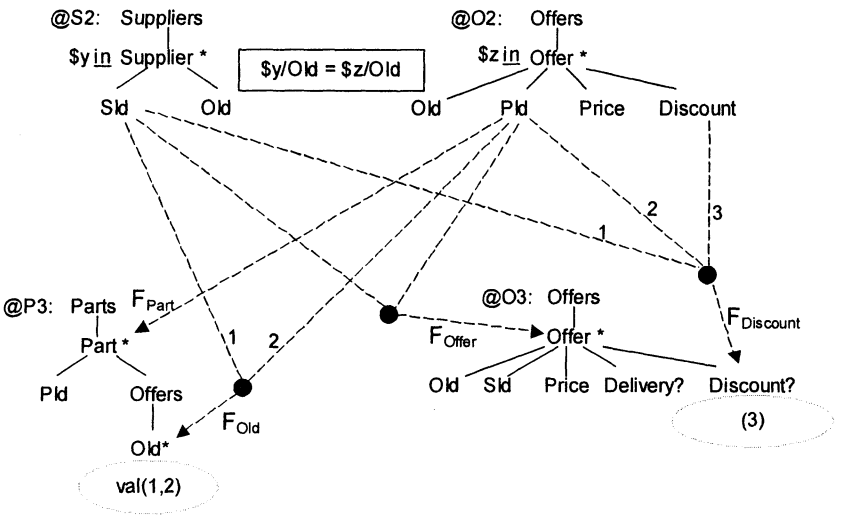

Figure 3. Graphical illustration of schema mapping based on Skolem functions

Variables `$y` and `$z` are defined over the source schema, and their ranges are sets of nodes determined by `@S2/Suppliers` and `@O2/Offers/Offer`, respectively. A constraint can be imposed on variable values (in the box). We assume that there is a Skolem function for any node type from a target schema tree. The function generates instances of this node type, i.e. nodes of this type in a target data tree. In our approach, we are interested in arguments of the function and not in how the function works. It is only important that a Skolem function returns different nodes (node identifiers) for different arguments, and the same values for equal arguments. It means that the value is uniquely determined by the function name and the value of argument list. Moreover, if two functions have different names they will never return the same value [Abiteboul et al., 2000]. In our approach, if a list of arguments is not empty then every argument is the text value of a leaf node from a source data tree. This

value is determined by a path expression specified over source schema tree. In Figure 3, we have:

- F_{Part}(`$z/PID`) generates instances of `@P3/Parts/Part`,
- F_{OId}(`$y/SId,$z/PID`) generates instances of `@P3/Parts/Part/Offers/OId`, and so on.

Additionally, if a target node is a leaf node we also specify the text value for it. The value is a text-valued function over arguments of the Skolem function generating this node. Arguments of a text-valued function will be denoted by indices of arguments in the argument list of the corresponding Skolem function. If the text-valued function is the identity function its name will be omitted. In Figure 3 specifications of text values for leaf nodes are depicted in ovals. If a Skolem function has more arguments then one, we index them by integers labeling appropriate edges in Figure 3.

The following definition lays out the notational convention used for specifying schema mappings.

DEFINITION 1 *A mapping between a source schema* **S** *and a target schema* **T** *is an expression conforming to the following syntax*

$\mathcal{M}_{\mathbf{S},\mathbf{T}}$ ::=
foreach $f_1, ..., f_n$
where Φ
exists sk_1 **in** P_1 [**with** $val_1(idx\text{-}seq_1)$],
...,
sk_m **in** P_m [**with** $val_m(idx\text{-}seq_m)$]

where f_i *is a range over* **S**, Φ *is a conjunction of atomic formulas,* sk_i *is a Skolem function expression over* **S**, P_i *is an absolute path in* **T**, val_i *is a text-valued function,* $idx\text{-}seq_i$ *is a sequence of indices of arguments of the Skolem function occurring in* sk_i *(applicable only if* P_i *leads to a leaf node).*

To explain our approach to schema mapping, we will use schemas from Figure 1. Let us assume the following notations:

$\mathbf{S}_1$={`@S1`}, and $\mathbf{S}_2$={`@S2`,`@O2`} – source schemas,
$\mathbf{S}_3$={`@P3`,`@O3`} – target schema,
$\mathbf{I}_1 = \{I^{@S1}\}$, and $\mathbf{I}_2 = \{I^{@S2}, I^{@O2}\}$ – source instances,
$\mathcal{M}_{1,3}$, $\mathcal{M}_{2,3}$ – mappings from $\mathbf{S}_1$ and $\mathbf{S}_2$, respectively, to $\mathbf{S}_3$.

Mappings $\mathcal{M}_{1,3}$ and $\mathcal{M}_{2,3}$ are specified in Figure 4 and Figure 5, respectively. The mappings specify how elements from source XML data relate to elements in the target.

According to these specifications, the root node $@P3/$ is obtained by invocation of the Skolem function $F_{P3}()$ (with the empty list of arguments). The unique node of type `@P3/Parts` is obtained by invocation of $F_{Parts}()$.

$\mathcal{M}_{1,3}$:
foreach $x in @S1/Suppliers/Supplier
exists
F_{P3}() **in** @P3/
F_{Parts}() **in** @P3/Parts
F_{Part}($x/PId) **in** @P3/Parts/Part
F_{PId}($x/PId) **in** @P3/Parts/Part/PId **with** (1)
F_{Offers}($x/PId) **in** @P3/Parts/Part/Offers
F_{OId}($x/SId,$x/PId) **in** @P3/Parts/Part/Offers/OId **with** val(1,2)
F_{O3}() **in** @O3/
F'_{Offers}() **in** @O3/Offers
F_{Offer}($x/SId,$x/PId) **in** @O3/Offers/Offer
F'_{OId}($x/SId,$x/PId) **in** @O3/Offers/Offer/OId **with** val(1,2)
F_{SId}($x/SId,$x/PId) **in** @O3/Offers/Offer/SId **with** (1)
F_{Price}($x/SId,$x/PId,$x/Price) **in** @O3/Offers/Offer/Price **with** (3)
$F_{Delivery}$($x/SId,$x/PId,$x/Delivery) **in** @O3/Offers/Offer/Delivery
with (3)

Figure 4. Mapping from source schema @S1 to target schema (@P3,@O3)

Note that in @S1/Suppliers/Supplier/PId there can be many nodes of type PId corresponding to the same real world *part*. In @P3, however, we want to have exactly one node of type PId for one *part*. Thus, we assign the Skolem function $F_{PId}(\$x/PId)$ with the path @P3/Parts/Part/PId. The function will return as many new nodes as many different values the expression $\$x/PId$ has. In this way we *merge* source nodes with the same PId. In general, in our approach criteria for merging are specified by appropriate construction of path expressions determining arguments of Skolem functions.

The mapping specification allows for partial specification, i.e. some target absolute path might not be constraint in a mapping from a source schema that does not have any corresponding information. For example, there is no node generating dependency for the path @O3/Offers/Offer/Discount in the mapping $\mathcal{M}_{1,3}$, because there is no Discount element in @S1. Similarly for @O3/Offers/Offer/Delivery in mapping $\mathcal{M}_{2,3}$. The advantage of these approach is that we can add, or remove the mapping constraints, when the source schema is changed [Yu and Popa, 2004].

A **with** clause specifies text value for an absolute path leading to a leaf node. The value is obtained from arguments of the Skolem function indicated by a sequence of integers written on the right hand side of the keyword **with**. If the sequence is $(k_1, ..., k_h)$ it means that the value is obtained by $value(E_{k_1}, ..., E_{k_h})$, where E_i is the *i-th* argument of a corresponding Skolem function, and $value()$ is a text-valued function. For $h = 1$ we assume that $value()$ is the identity function and its name will be omitted. For example:
F_{OId}($x/SId,$x/PId) **in** @P3/Parts/Part/Offers/OId **with** val(1,2)

specifies that the text value of the path `@O3/Offers/Offer/OId` is equal to $val(\$x/\texttt{SId}, \$x/\texttt{PId})$, so, it is determined by the first and the second arguments of the Skolem function F_{OId}.

Value constraints can impose equalities between some target values. For example (see Figure 5), values of nodes in

`@P3/Parts/Part/Offers/OId`, and in

`@O3/Offers/Offer/OId`

are determined by the expression $val(\$y/\texttt{SId}, \$z/\texttt{PId})$. Thus, the value equality between leaf nodes from these two different paths is imposed. That allows for joining corresponding target elements.

$\mathcal{M}_{2,3}$:

foreach `$y:@S2/Suppliers/Supplier`
`$z:@O2/Offers/Offer`
where
`$y/OId=$z/Oid`
exists
F_{P3}`()` **in** `@P3/`
F_{Parts}`()` **in** `@P3/Parts`
F_{Part}`($z/PId)` **in** `@P3/Parts/Part`
F_{PId}`($z/PId)` **in** `@P3/Parts/Part/PId` **with** `(1)`
F_{Offers}`($z/PId)` **in** `@P3/Parts/Part/Offers`
F_{OId}`($y/SId,$z/PId)` **in** `@P3/Parts/Part/Offers/OId` **with** `val(1,2)`
F_{O3}`()` **in** `@O3/`
F'_{Offers}`()` **in** `@O3/Offers`
F_{Offer}`($y/SId,$z/PId)` **in** `@O3/Offers/Offer`
F'_{OId}`($y/SId,$z/PId)` **in** `@O3/Offers/Offer/OId` **with** `val(1,2)`
F_{SId}`($y/SId,$z/PId)` **in** `@O3/Offers/Offer/SId` **with** `(1)`
F_{Price}`($y/SId,$z/PId,$z/Price)` **in** `@O3/Offers/Offer/Price` **with** `(3)`
$F_{Discount}$`($y/SId,$z/PId,$z/Discount)` **in** `@O3/Offers/Offer/Discount`
with `(3)`

Figure 5. Mapping from source schema (`@S2,@O2`) to target schema (`@P3,@O3`)

3. Rewriting rules

Rewriting rules determine how a target query (on a target schema **T**) should be translated into a source query (on a target schema **S**) taking into account a mapping between these source and target schemas. For our running example we have the following interpretation.

Let Q be a target query over $\mathbf{S}_3$. We want to find rewritings $\tau_{\mathcal{M}_{1,3}}$, and $\tau_{\mathcal{M}_{2,3}}$ such that: $\tau_{\mathcal{M}_{1,3}}(Q)$ is a source query over $\mathbf{S}_1$, $\tau_{\mathcal{M}_{2,3}}(Q)$ is a source query over $\mathbf{S}_2$, and for any instances I_1 and I_2 of schemas $\mathbf{S}_1$ and $\mathbf{S}_2$, respectively, the following equality holds:

$$Q(\mathcal{M}_{1,3}(\mathrm{I}_1) \cup \mathcal{M}_{2,3}(\mathrm{I}_2)) = \tau_{\mathcal{M}_{1,3}}(Q)(\mathrm{I}_1) \cup \tau_{\mathcal{M}_{2,3}}(Q)(\mathrm{I}_2),$$

where $\mathcal{M}(I)$ denotes the canonical target instance corresponding to I with respect to $\mathcal{M}$, and $\cup$ denotes an operation of merging two XML documents.

We will consider queries that have the following XQuery-like form [XQuery 1.0: An XML Query Language. W3C Working Draft, 2002]:

DEFINITION 2 *A query is an expression of the form*

$q ::=$ <tag> **for** $f_1, ..., f_2$
where $w_1 \wedge ... \wedge w_m$
return r</tag>

where each f_i is a range, the **where** *clause is a conjunction of atomic formulas, and* **return** *clause is an expression defined by the grammar $r ::=$ <tag>r</tag>$\mid e \mid e\ r \mid q$, where $e ::=$ <tag>$\$x/P$</tag>, and the path $\$x/P$ ends at a leaf node. Each variable occurring in q is defined either in a range f_i or in a superquery in which q is nested.*

In Figure 6 we define rewriting rules for reformulating a target query into a source query based on a mapping specification. A rewriting rule is a collection of *premises* and *conclusions*, written respectively above and below a dividing line. The premise part of a rule is a set of conditions which implies rewriting actions connected with the conclusion part.

There are three type of conditions: *target query conditions*, *mapping conditions*, and *variable mapping conditions* (by A we denote an absolute path).

1 *A target query condition* is an expression of one of the following forms $F_t : e$, $W_t : e$, or $R_t : e$, and asserts that e is an expression within, respectively, the **for**, **where**, or **return** part of the target query. The e expression is to be rewritten by a (sequence of) source query expression(s) defined in the conclusion of the rule.

2 *A mapping condition* is an expression of one of the following forms $\mathcal{M}^f : d, \mathcal{M}^w : d, \mathcal{M}^e : d$ or $\mathcal{M}^v : d$, and asserts that d is an expression occurring in the mapping specification under consideration. The expression d may be: a source range occurring in the **foreach** part of $\mathcal{M}$, a conjunction of atomic formulas occurring in the **where** part of $\mathcal{M}$, a node generating dependency $SF(...)$ **in** P, occurring in the **exists-in** part of $\mathcal{M}$ or a value dependency P **with** $val(...)$ occurring in the **exists-with** part of $\mathcal{M}$. The superscript of $\mathcal{M}$ denotes the kind of d.

3 *A variable mapping condition* is an expression of the form $\omega\ [\$t \mapsto (\$s_1, ..., \$s_n)]$, where $\$t$ is a target variable defined in the target query Q_t, and $\$s_1, ..., \$s_n$ are source variables invented for the source query Q_s. The premise is valid if the mapping $[\$t \mapsto (\$s_1, ..., \$s_n)]$ has been defined in a conclusion part of a preceding rewriting rule.

$$(R1)\ \frac{\begin{array}{l} F_t : \$t \text{ in } A \\ \mathcal{M}^e : SF(\$x_1/P_1, ..., \$x_n/P_n) \textbf{ in } A \\ \mathcal{M}^f : \$x_1 \text{ in } A_1, ..., \$x_n \text{ in } A_n \\ \mathcal{M}^w : \Phi_{\$x_1,...,\$x_n} \end{array}}{\begin{array}{l} \omega : [\$t \mapsto (\$s_1, ..., \$s_n)] \\ F_s : \mathcal{M}^f[\$x_1 \rightarrow \$s_1, ..., \$x_n \rightarrow \$s_n] \\ W_s : \mathcal{M}^w[\$x_1 \rightarrow \$s_1, ..., \$x_n \rightarrow \$s_n] \end{array}}$$

$$(R2)\ \frac{\begin{array}{l} F_t : \$t \text{ in } \$t'/P \\ \mathcal{M}^e : SF_1(\$x_1/P_1, ..., \$x_n/P_n) \textbf{ in } type(\$t) \\ \mathcal{M}^e : SF_2(\$x_1/P_1, ..., \$x_{n-k}/P_{n-k}) \textbf{ in } type(\$t') \\ \mathcal{M}^f : \$x_1 \text{ in } A_1, ..., \$x_n \text{ in } A_n \\ \mathcal{M}^w : \Phi_{\$x_1,...,\$x_n} \\ \omega : [\$t' \mapsto (\$s'_1, ..., \$s'_{n-k})] \end{array}}{\begin{array}{l} \omega : [\$t \mapsto (\$s_1, ..., \$s_n)] \\ F_s : \mathcal{M}^f[\$x_1 \rightarrow \$s_1, ..., \$x_n \rightarrow \$s_n] \\ W_s : \mathcal{M}^w[\$x_1 \rightarrow \$s_1, ..., \$x_n \rightarrow \$s_n] \wedge \\ \quad \wedge \$s_1/P_1 = \$s'_1/P_1 \wedge \ ... \ \wedge \$s_{n-k}/P_{n-k} = \$s'_{n-k}/P_{n-k} \end{array}}$$

$$(R3)\ \frac{W_t : \$t_1/P_1 = \$t_2/P_2}{\begin{array}{l} F_s : R5(\$t_1/P_1).F;\ F_s : R5(\$t_2/P_2).F \\ W_s : R5(\$t_1/P_1).W \wedge R5(\$t_2/P_2).W \\ W_s : R5(\$t_1/P_1).R = R5(\$t_2/P_2).R \end{array}}$$

$$(R4)\ \frac{\begin{array}{l} R_t : \$t/P \\ \mathcal{M}^e : SF(\$x_1/P_1, ..., \$x_n/P_n) \textbf{ in } type(\$t/P) \\ \mathcal{M}^v : type(\$t/P) \textbf{ with } value(k_1, ..., k_h) \end{array}}{\begin{array}{l} F_s : R5(\$t/P).F \\ W_s : R5(\$t/P).W \\ R_s : value((R5(\$t/P).R)[k_1, ..., k_h]) \end{array}}$$

$$(R5)\ \frac{\begin{array}{l} Val : \$t/P \\ \mathcal{M}^e : SF_1(\$x_1/P_1, ..., \$x_n/P_n) \textbf{ in } type(\$t/P) \\ \mathcal{M}^e : SF_2(\$x_1/P_1, ..., \$x_{n-k}/P_{n-k}) \textbf{ in } type(\$t) \\ \mathcal{M}^f : \$x_1 \text{ in } A_1, ..., \$x_n \text{ in } A_n \\ \mathcal{M}^w : \Phi_{\$x_1,...,\$x_n} \\ \omega : [\$t \mapsto (\$s_1, ..., \$s_{n-k})] \end{array}}{\begin{array}{l} F : \$s'_{n-k+1} \text{ in } A_{n-k+1}, ..., \$s'_n \text{ in } A_n \\ W : \mathcal{M}^w[\$x_1 \rightarrow \$s_1, ..., \$x_{n-k} \rightarrow \$s_{n-k}, \\ \quad \$x_{n-k+1} \rightarrow \$s'_{n-k+1}, ..., \$x_n \rightarrow \$s'_n] \\ R : (\$s_1/P_1, ..., \$s_{n-k}/P_{n-k}, \$s'_{n-k+1}/P_{n-k+1}, ..., \$s'_n/P_n) \end{array}}$$

Figure 6. Rewriting rules for query reformulation

The conclusion part of a rule consists of a variable mapping and a set of source query elements that are of the form $F_s : e_f$, $W_s : e_w$, or $R_s : e_r$, and represent the **for**, **where**, or **return** parts of a source query, respectively.

4. Example of rewriting

Now, we will show how rewriting rules from Figure 6 can be used to reformulate a target query. The query Q from Figure 7 is a target query over target schema (@P3,@O3). Some steps of reformulating Q according to schema mapping $\mathcal{M}_{1,3}$ using rewriting rules (R1)-(R4) and the auxiliary rule (R5), are given in Figure 10. The result of rewriting is the query in Figure 9.

```
Q :
<Result>
   for $p in @P3/Parts/Part
       $s in $p/Offers
       $o in @O3/Offers/Offer
   where $s/OId = $o/OId
   return
   <Part>
      <PartId>$p/PId</PartId>
      <Supplier>$o/SId</Supplier>
      <Price>$o/Price</Price>
      <Discount>$o/Discount</Discount>
   </Part>
</Result>
```

Figure 7. Example of a target query

In Figure 8 we show three kinds of data integration scenarios that can be realized within our approach to P2P data sharing systems.

We assume that P_3 provides a schema $\mathbf{S}_3$. There are two other peers, P_1 and P_2, that store local data with schemas $\mathbf{S}_1$ and $\mathbf{S}_2$, respectively. We also assume that there are schema mappings $\mathcal{M}_{1,3}$, $\mathcal{M}_{2,3}$ and $\mathcal{M}_{1,2}$ globally available to all peers.

When P_3 receives a query Q formulated over its schema (a target query) then the following situations may occur [Tatarinov and Halevy, 2004]:

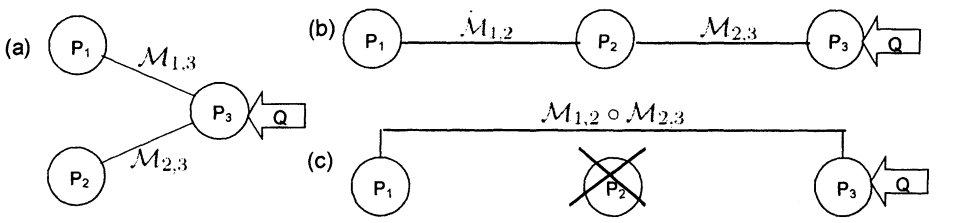

Figure 8. Three scenarios of data integration in P2P -system

(a) P_3 starts from the querying its own data (if it stores any) and reformulates Q over its immediate neighbors, Figure 8(a). This reformulation can be performed using available schema mappings. In our case $\mathcal{M}_{1,3}$ and $\mathcal{M}_{2,3}$ are used to obtain Q' (a source query over P_1) and Q'' (a source query over P_2). The reformulation processes for Q with respect to mapping $\mathcal{M}_{1,3}$ is illustrated in Figure 10.

(b) P_3 reformulates Q over some neighbor peers (P_2 in our case). Each such a peer can also play the role of mediator and reformulates obtained query Q' over its neighbor peers (P_1 in our case), and so on, until all the relevant data sources are reached, Figure 8(b). In this way the initial query Q is reformulated across all peers participating in the data sharing system, and answers are in turn merged and sent back to the starting peer.

(c) It may happen that instead of successive reformulation we have to perform reformulation based on a composition of mappings, as was illustrated in Figure 8(c), where peer P_2 is not available. Composing schema mappings is a new challenging problem formulated recently in [Fagin et al., 2004; Madhavan and Halevy, 2003]: given a schema mapping $\mathcal{M}_{1,2}$ from schema $\mathbf{S}_1$ to schema $\mathbf{S}_2$, and a schema mapping $\mathcal{M}_{2,3}$ from schema $\mathbf{S}_2$ to schema $\mathbf{S}_3$, derive a schema mapping $\mathcal{M}_{1,3}$ from schema $\mathbf{S}_1$ to schema $\mathbf{S}_3$ that is "equivalent" to the successive application of $\mathcal{M}_{2,3}$ and $\mathcal{M}_{1,2}$, i.e. $\mathcal{M}_{1,3} = \mathcal{M}_{1,2} \circ \mathcal{M}_{2,3}$. (Schema composition is not addressed in this paper.)

Q' :

```
<Result>
  for $p' in @S1/Suppliers/Supplier
      $s' in @S1/Suppliers/Supplier,
      $o' in @S1/Suppliers/Supplier
  where $p'/PId = $s'/PId and $s'/OId = $o'/OId and $s'/SId = $o'/SId
  return
  <Part>
      <PartId>$p'/PId</PartId>
      <Supplier>$o'/SId</Supplier>
      <Price>$o'/Price</Price>
  </Part>
</Result>
```

Figure 9. Reformulated query Q with respect to mapping $\mathcal{M}_{1,3}$

$$(R1\text{–}1)\ \frac{\begin{array}{l}F_t : \$p \underline{\text{in}}\ @P3/Parts/Part\\ \mathcal{M}^e : F_{Part}(\$x/\texttt{PId})\ \textbf{in}\ @P3/Parts/Part\\ \mathcal{M}^f : \$x \underline{\text{in}}\ @S1/Suppliers/Supplier\end{array}}{\omega : [\$p \mapsto \$p'];\ F_s : \$p' \underline{\text{in}}\ @S1/Suppliers/Supplier}$$

$$(R2\text{–}2)\ \frac{\begin{array}{l}F_t : \$s \underline{\text{in}}\ \$p/Offers\\ \mathcal{M}^e : F_{Offers}(\$x/\texttt{PId})\ \textbf{in}\ @P3/Parts/Part/Offers\\ \mathcal{M}^e : F_{Part}(\$x/\texttt{PId})\ \textbf{in}\ @P3/Parts/Part\\ \mathcal{M}^f : \$x \underline{\text{in}}\ @S1/Suppliers/Supplier;\ \omega : [\$p \mapsto \$p']\end{array}}{\begin{array}{l}\omega : [\$s \mapsto \$s'];\ F_s : \$s' \underline{\text{in}}\ @S1/Suppliers/Supplier\\ W_s : \$p'/PId = \$s'/PId\end{array}}$$

$$(R1\text{–}3)\ \frac{\begin{array}{l}F_t : \$o \underline{\text{in}}\ @O3/Offers/Offer\\ \mathcal{M}^e : F_{Offer}(\$x/\texttt{SId}, \$x/\texttt{PId})\ \textbf{in}\ @O3/Offers/Offer\\ \mathcal{M}^f : \$x \underline{\text{in}}\ @S1/Suppliers/Supplier\end{array}}{\omega : [\$o \mapsto \$o'];\ F_s : \$o' \underline{\text{in}}\ @S1/Suppliers/Supplier}$$

$$(R3\text{–}4)\ \frac{W_t : \$s/OId = \$o/OId}{W_s : \$s'/PId = \$o'/PId \wedge \$s'/SId = \$o'/SId}$$

$$(R5\text{–}4)\ \frac{\begin{array}{l}Val : \$s/OId\\ \mathcal{M}^e : F_{OId}(\$x/SId, \$x/PId)\ \textbf{in}\ @P3/Parts/Part/Offers/OId\\ \mathcal{M}^e : F_{Offers}(\$x/PId)\ \textbf{in}\ @P3/Parts/Part/Offers\\ \mathcal{M}^f : \$x \underline{\text{in}}\ @S1/Suppliers/Supplier;\ \omega : [\$s \mapsto \$s']\end{array}}{R : (\$s'/SId, \$s'/PId)}$$

$$(R4\text{–}5)\ \frac{\begin{array}{l}R_t : \$p/PId\\ \mathcal{M}^e : F_{PId}(\$x/PId)\ \textbf{in}\ @P3/Parts/Part/PId\\ \mathcal{M}^v : @P3/Parts/Part/PId = (1)\end{array}}{R_s : \$p'/PId}$$

Figure 10. Reformulation of query Q using mapping $\mathcal{M}_{1,3}$ and rewriting rules from Figure 6

5. Conclusion

This paper presents a novel approach to XML query reformulation based on Skolem functions and its application to data integration and data sharing in the context of P2P e-commerce systems. We propose a new method for schema mapping specification between heterogeneous data sources and rewriting rules for query reformulation in an environment of cooperating peers. Such advanced data management features provide a new facility in the field of integration, interchange and transformation of data across global marketplace. The described method is a part of our work on data integration [Pankowski and Hunt, 2005] and transformation of heterogeneous data sources [Pankowski, 2004]. Current work is devoted to evaluate algorithms in real word scenarios.

References

Abiteboul, S., Buneman, P., and Suciu, D. (2000). *Data on the Web. From Relational to Semistructured Data and XML*. Morgan Kaufmann, San Francisco.

Bernstein, P. A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., and Zaihrayeu, I. (2002). Data management for peer-to-peer computing: A vision. In *Proc. of the 5th International Workshop on the Web and Databases (WebDB 2002)*, pages 1–6.

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

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

Halevy, A. Y. (2001). Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294.

Lenzerini, M. (2002). Data integration: a theoretical perspective. In *Proc. of the 21th ACM SIGMOD Symposium on Principles of Database Systems (PODS 2002)*, pages 233–246.

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

Miller, R. J., Haas, L. M., and Hernandez:, M. A. (2000). Schema mapping as query discovery. In *Proc. of the 26th International Conference on Very Large Data Bases, VLDB 2000, Cairo, Egypt*, pages 77–88.

Pankowski, T. (2004). A High-Level Language for Specifying XML Data Transformations, In: Advances in Databases and Information Systems, ADBIS 2004. *Lecture Notes in Computer Science*, 3255:159–172.

Pankowski, T. (2005). Specifying Schema Mappings for Query Reformulation in Data Integration Systems. In *Proc. of the 3-rd Atlantic Web Intelligence Conference - AWIC'2005*, Lecture Notes in Artificial Intelligence 3528, Springer-Verlag, pages 361–365.

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

Popa, L., Velegrakis, Y., Miller, R. J., Hernandez, M. A., and Fagin, R. (2002). Translating web data. In *Proc. of the 28th International Conference on Very Large Data Bases, VLDB 2002, Hong Kong, China*, pages 598–609.

Quix, C., Schoop, M., and Jeusfeld, M. A. (2002). Business Data Management for B2B Electronic Commerce. *SIGMOD Record*, 31(1):49–54.

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

Sheth, A. P. and Larson, J. A. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236.

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

Ullman, J. D. (1997). Information integration using logical views. in: Database Theory - ICDT 1997. *Lecture Notes in Computer Science*, 1186:19–40.

XQuery 1.0: An XML Query Language. W3C Working Draft (2002). www.w3.org/TR/ xquery.

XQuery 1.0 and XPath 2.0 Data Model. W3C Working Draft (2002). www.w3.org/TR/ query-datamodel.

Yu, C. and Popa, L. (2004). Constraint-based XML query rewriting for data integration. In *Proc. of the 2004 ACM SIGMOD Conference*, pages 371–382.