Specifying Schema Mappings for Query Reformulation in Data Integration Systems

Tadeusz Pankowski[1,2]

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

Abstract. In data integration systems there is a problem of answering queries through a target schema, given a set of mappings between source schemas and the target schema, and given that the data is at the sources. This is of special importance when integrated sources, e.g. from Web data repositories, have overlapping data and its merging is necessary. We propose a language for specifying a class of mappings between source and target schemas, and design rewriting rules which reformulate a target query to a query over data sources based on the mappings.

1 Introduction

The main problem of data integration systems is to combine data from different sources, and to provide the user with a unified view of these data [2, 3, 9]. We assume that there is a set of *source schemas* describing real data, and a *target schema* (or *mediated schema*) that is a virtual and reconciled view of the underlying sources. This allows the user for uniformly querying many different sources through one common interface (target schema). There are two approaches to answer queries in such environment: *view materialization* and *query reformulation*. The former assumes that a materialized instance of the target schema is available. In the latter, a query is reformulated in such a way that it can be processed against underlying sources, and partial answers are merged to obtain the final result. To realize query reformulation, relationships or *schema mappings* must first be established between the source schemas and the target schema.

In this paper we propose a method for specifying schema mappings and show how this specification can be used for query reformulation in data integration systems. We follow the GAV (*global-as-view*) approach and assume that both source and target are XML data. A schema mapping is specified by a high-level language XDMap which is an extension of XDTrans [5]. We propose four rewriting rules which reformulate a target query to a query over data sources. This reformulation is based on the mappings and some target constraints.

The rest of the paper is organized as follows: In Section 2 we define XDMap as a language for specifying mappings between schemas. In Section 3 we show how mapping between schemas can be defined and used for query reformulation. Section 4 concludes the paper.

2 XDMap - a language for schema mapping specification

To define schema mapping we will use a language called XDMap, which is an extended version of our XML data transformation language XDTrans [5]. We will consider mappings over XML data, so the syntax and semantics of the language is oriented towards the tree nature of XML. According to W3C standard, any XML document can be represented as a data tree [**?**], where a node conforms to one of the seven node types: *root*, *element*, *attribute*, *text*, *namespace*, *processing instruction*, and *comment*. In this paper, we restrict our attention to four first of them. Every node has a unique *node identifier* (*nid*) - to obtain an unique node identifier we will use Skolem functions.

Definition 1. *A data tree is an expression defined by the syntax:*

$$\begin{array}{lll} \textit{data tree} ::= nid(tree), & & \\ \textit{tree} ::= \textit{e-tree} \mid \textit{a-tree} \mid \textit{t-tree}, & & \\ \textit{e-tree} ::= \langle e, nid\rangle(tree, ..., tree), & & \textit{(element tree),} \\ \textit{a-tree} ::= \langle a, nid\rangle(s), & & \textit{(attribute tree),} \\ \textit{t-tree} ::= nid(s), & & \textit{(text tree),} \end{array}$$

where nid, e, a, and s are from, respectively, a set $\mathcal{N}$ of node identifiers, a set Σ_E of element labels, a set Σ_A of attribute labels, and a set $\mathcal{S}$ of string values. By $\mathcal{D}_{\Sigma,\mathcal{S}}(\mathcal{N})$, where $\Sigma = \Sigma_E \cup \Sigma_A$, will be denoted a set of all data trees over Σ and $\mathcal{S}$ with node identifiers from $\mathcal{N}$. □

Further on we assume that $\mathcal{C}$ is a set of *non-terminal symbols* called *concepts*, $\mathcal{P}$ is a set of *XPath expressions* in which *variables* can appear, and $\mathcal{F}$ is a set of 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 are distinct.

The goal of *schema mapping* is to define a conversion of a set of source schemas (data tree types) into an expected set of target schemas. A mapping can be specified by a set of *mapping rules*. Every rule determines a type of expected *final* or *intermediate* result tree in a form of a *tree expression*.

Definition 2. *A tree expression over alphabets Σ, $\mathcal{C}$, $\mathcal{S}$, $\mathcal{P}$ and $\mathcal{F}$ conforms to the following syntax:*

$\tau ::= s \mid E \mid SF(...)(s) \mid \langle a, SF(...)\rangle(s) \mid \langle e, SF(...)\rangle(\tau, ..., \tau) \mid C(E, ..., E)$,

where: $s \in \mathcal{S}$, $E \in \mathcal{P}$, $SF \in \mathcal{F}$, $a \in \Sigma_A$, $e \in \Sigma_E$, $C \in \mathcal{C}$. The set of all tree expressions will be denoted by $\mathcal{T}_{\Sigma,\mathcal{S}}(\mathcal{C}, \mathcal{P}, \mathcal{F})$. □

Definition 3. *A mapping specification language is a system*

$$XDMap = (\Sigma, \mathcal{C}, \mathcal{S}, \mathcal{F}, \texttt{START}, \mathcal{P}, \mathcal{R}),$$

where $\texttt{START} \in \mathcal{C}$ is the initial concept, and $\mathcal{R}$ is a finite set of rules of the form:

$$(C, (\$v_1 : E_1, ..., \$v_p : E_p)) \rightarrow \tau, ..., \tau,$$

where $C \in \mathcal{C}$, any E (possibly with subscripts) is from $\mathcal{P}$, $\$v$ (possibly with subscripts) is a variable, $\tau \in \mathcal{T}_{\Sigma,\mathcal{S}}(\mathcal{C}, \mathcal{P}, \mathcal{F})$, and every variable occurring in the body occurs also in the head of the rule. □

The head of a rule includes a concept C which will be rewritten by the body of the rule. A rule with concept C in the head *defines* this concept. We assume that any concept in a given set of rules must be defined, and that every concept has exactly one definition. Thus, our system is deterministic. Recursive definitions for concepts are also allowed. There must be exactly one rule, the *initialization rule*, defining the initial concept `START`. In order to refer to the root of a document we use "@doc/".

3 Schema mapping and rewriting rules

In Figure 1 we have a source schema, two target schemas and mappings between them. Constraints in Figure 1 can easily be deduced from the mapping specification. Target schemas are defined as views over the source, so the GAV approach is applied. The mapping explicitly tells the system how to retrieve the data when one wants to evaluate the various elements of the target schema. This idea is effective whenever the data integration system is based on a set of sources that is stable.

Source and target schemas:

```
src:                tgt:                tgt:
  Students            Students            Evals
    Student*            Student*            Eval*
      Id                  StId                EvalId
      Name                StName              Course
      Course              CourEvals           Grade
      Grade                 EvalId*
```

Mappings:

$$(\texttt{START}, \$x : @\texttt{src/Students}) \rightarrow \langle @\texttt{tgt/Students}, STS()\rangle(\texttt{STUDENT}(\$x)), \langle @\texttt{tgt/Evals}, EVS()\rangle(\texttt{EVAL}(\$x))$$

$$(\texttt{STUDENT}, \$x : \texttt{Student}) \rightarrow \langle \texttt{Student}, STU(\$x/\texttt{Id})\rangle(\langle \texttt{StId}, STI(\$x/\texttt{Id})\rangle(\$x/\texttt{Id}), \langle \texttt{StName}, STN(\$x/\texttt{Id}, \$x/\texttt{Name})\rangle(\$x/\texttt{Name}), \langle \texttt{CourEvals}, CES(\$x/\texttt{Id}, \$x/\texttt{Course})\rangle(\texttt{EVALID}(\$x)))$$

$$(\texttt{EVALID}, \$x : .) \rightarrow \langle \texttt{EvalId}, EVI(\$x/\texttt{Id}, \$x/\texttt{Course})\rangle(\texttt{value}(\$x/\texttt{Id}, \$x/\texttt{Course}))$$

$$(\texttt{EVAL}, \$x : \texttt{Student}) \rightarrow \langle \texttt{Eval}, EVA(\$x/\texttt{Id}, \$x/\texttt{Course})\rangle(\texttt{EVALID}(\$x), \langle \texttt{Course}, COU(\$x/\texttt{Id}, \$x/\texttt{Course})\rangle(\$x/\texttt{Course}), \langle \texttt{Grade}, GRA(\$x/\texttt{Id}, \$x/\texttt{Course}, \$x/\texttt{Grade})\rangle(\$x/\texttt{Grade}))$$

Constraints:

STS = @tgt/Students EVI = CES/EvalId EVS = @tgt/Evals
STU = STS/Student EVA = EVS/Eval
STI = STU/StId EID = EVA/EvalId
STN = STU/StName COU = EVA/Course
CES = STU/CourEvals GRA = EVA/Grade

Fig. 1. Source and target schemas, mappings and constraints

The first rule tells that the start non-terminal concept `START` denoting the expected target, is to be replaced by two target trees. The first target tree starts with the outermost element "<students>...</students> and the `students` node is uniquely determined by the Skolem function *STS*(). The subtree pointed to by the node is denoted by non-terminal tree expression `STUDENT($x)`, where the current context node is bound to variable `$x` and is passed to the rule defining the concept `STUDENT` (i.e. to the second rule). The precise semantics for transformation rules where Skolem functions are not given explicitly is defined in [5].

In Figure 2 we define rewriting rules for reformulating the target query into a source query (or a set of source queries) based on the mapping rules and constraints. The reformulation is achieved by iteratively applying rewriting rules to a target query. We use the following notations: *tgtE* and *srcE* denote XPath expressions on target and source schema, respectively; $SF_{tgtE}(L_{\$x})$ denotes the Skolem function associated to path expression *tgtE* in mapping specification, and $L_{\$x}$ denotes a list of arguments of the function. For example, *CES*(`$x/Id,$x/Course`) is associated to `@tgt/students/student/CourEvals`; $[\$z \mapsto \$x]$ is a *replacement operation* replacing occurrences of target variable $\$z$ with source variable $\$x$; $L_{\$x}[\$x \to \$x']$ denotes the result of substitution all occurrences of $\$x$ in $L_{\$x}$ with $\$x'$. An expression of the form $SF_{tgtE}(L_{\$x}),\ value(tgtE) = L_{\$x}.k$ means, that the text value associated to a path *tgtE* is equal to the *k-th* component of the argument list $L_{\$x}$ of the Skolem function $SF_{tgtE}(L_{\$x})$ associated to *tgtE*.

$$(R1)\ \frac{\$z : tgtE, SF_{tgtE}(L_{\$x}), \$x : srcE}{\$x : srcE, [\$z \mapsto \$x]}$$

$$(R2)\ \frac{\$z : \$y/tgtE, SF_{\$y/tgtE}(L_{\$x}, L'_{\$x}) = SF_{\$y}(L_{\$x})/tgtE, \$x : srcE, [\$y \mapsto \$y']}{\$x : srcE \wedge L_{\$x} = (L_{\$x}[\$x \to \$y']),\ [\$z \mapsto\ \$x]}$$

$$(R3)\ \frac{\$x/tgtE_1 = \$y/tgtE_2}{R5(\$x/tgtE_1) = R5(\$y/tgtE_2)}$$

$$(R4)\ \frac{\$x/tgtE, SF_{\$x/tgtE}(L_{\$x}), value(\$x/tgtE) = L_{\$x}.k}{R5(\$x/tgtE).k}$$

$$(R5)\ \frac{\$y/tgtE, SF_{\$y/tgtE}(L_{\$x}) = SF_{\$y}(L'_{\$x})/tgtE, [\$y \mapsto \$y']}{L_{\$x}[\$x \to \$y']}$$

Fig. 2. Rewriting rules

The two first rules are used to reformulate variable definitions from the **for** clause of the query. (R3) is used to rewrite an atomic equality from the **where** clause, and (R4) is applied to rewrite terms from the **return** part of the query. Rule (R5) is an auxiliary rule invoked by (R3) and (R4).

In Figure 3 we show application of rewriting rules to reformulate a query.

Target query Q:

```
(1) for    $s in @tgt/Students/Student
(2)        $c in $s/CourEvals
(3)        $e in @tgt/Evals/Eval
(4) where  $c/EvalId=$e/EvalId
(5) return {$s/StName}
(6)        {$e/Course}
(7)        {$e/Grade}
```

Application of rewriting rules to query Q:

$$(1)\quad \frac{\$s : @tgt/Students/Student,\ STU(\$s'/Id),\ \$s' : @src/Students/Student}{\$s' : @src/Students/Student,\ [\$s \mapsto\ \$s']}$$

$$(2)\quad \frac{\$c : \$s/CourEvals,\ CES(\$c'/Id, \$c'/Course) = STU(\$c'/Id)/CourEvals,\ \$c' : @src/Students/Student,\ [\$s \mapsto\ \$s']}{\$c' : @src/Students/Student \wedge c'/Id = s'/Id, [\$c \mapsto\ \$c']}$$

$$(3)\quad \frac{\$e : @tgt/Evals/Eval, EVA(\$e'/Id, \$e'/Course), \$e' : @src/Students/Student}{\$e' : @src/Students/Student,\ [\$e \mapsto\ \$e']}$$

$$(4)\quad \frac{\$c/EvalId = \$e/EvalId}{(\$c'/Id, \$c'/Course) = (\$e'/Id, \$e'/Course)}$$

$$(5)\quad \frac{\$s/StName}{s'/Name} \qquad (6)\quad \frac{\$e/Course}{e'/Course} \qquad (7)\quad \frac{\$e/Grade}{e'/Grade}$$

For (5), we have the following inference performed by invocation of (R5):

$$(R5)\quad \frac{\$s/StName, STN(\$x/Id, \$x/Name) = STU(\$x/Id)/StName, [s \mapsto s']}{(\$s'/Id, \$s'/Name)}$$

Reformulated query Q:

```
for    $s' in @src/Students/Student
       $c' in @src/Students/Student
       $e' in @src/Students/Student
where  $c'/Id=$e'/Id
       ∧ $c'/Course=$e'/Course
       ∧ $c'/Id=$s'/Id
return {$s'/Name}
       {$e'/Course}
       {$e'/Grade}
```

Fig. 3. Reformulation of a target query to a source query

4 Conclusion

The problem of schema mapping and query reformulation has received considerable attention in recent years especially in the context of data integration, where data from various heterogeneous sources has to be transformed into data structured under a target (or mediated) schema. In relational data integration systems mappings have been defined in the form of GAV (*global-as-view*) or LAV (*local-as-view*) [3, 8]. Schema mappings have been also investigated in more general data integration scenarios [1, 4, 7]. Mappings are often specified in a high-level declarative way that state how groups of related elements in a source schema correspond to groups of related elements in the target schema [9].

Novelty of this paper is as follows: (1) we propose a language XDMap for specifying schema mappings, the language involves Skolem functions and allows for expressing a rich set of data restructuring operations (among other grouping and merging); (2) we propose rewriting rules for reformulating target queries into queries over source data, rewriting algorithm is illustrated by an example.

The proposed approach can be used when the source schemas are known. When these are not available some other methods based on schema discovery and domain ontologies should be applied. Such a case we discuss in [6].

References

1. Fagin, R., Popa, L., Kolaitis, P., Tan, W.-C.: Composing Schema Mappings: Second-Order Dependencies to the Rescue, *Proc. of the 23th ACM SIGMOD Symposium on Principles of Database Systems (PODS 2004)*, 2004, 83–94.
2. Halevy, A. Y.: Answering queries using views: A survey, *The VLDB Journal*, **10**(4), 2001, 270–294.
3. Lenzerini, M.: Data integration: a theoretical perspective, *ACM SIGMOD Symposium on Principles of Database Systems (PODS 2002)*, 2002, 233–246.
4. Madhavan, J., Halevy, A. Y.: Composing mappings among data sources, *Proc. of the 29th International Conference on Very Large Data Bases, VLDB 2003, Berlin, Germany*, 2003, 572–583.
5. Pankowski, T.: A High-Level Language for Specifying XML Data Transformations, In: Advances in Databases and Information Systems, ADBIS 2004, *Lecture Notes in Computer Science*, **3255**, 2004, 159–172.
6. Pankowski, T., Hunt, E.: 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, 2005.
7. Tatarinov, I., Halevy, A.: Efficient query reformulation in peer data management systems, *Proc. of the 2004 ACM SIGMOD International Conference on Management of Data*, 2004, 539–550.
8. Ullman, J. D.: Information Integration Using Logical Views. In: Database Theory - ICDT 1997, *Lecture Notes in Computer Science*, **1186**, 1997, 19–40.
9. Yu, C., Popa, L.: Constraint-based XML query rewriting for data integration, *Proc. of the 2004 ACM SIGMOD International Conference on Management of Data*, 2004, 371–382.