Managing XML Schema Mappings and Annotations in P2P Data Integration Systems

Tadeusz Pankowski and Magdalena Niwińska

Institute of Control and Information Engineering,
Poznań University of Technology, Poland
tadeusz.pankowski@put.poznan.pl

Abstract. The crucial problem in semantic data integration is creating and maintaining mappings between heterogeneous, independently designed data sources. To deal with the problem we can enrich XML schemas with semantic information from a domain ontology by annotating the schema. In this paper we discuss how the annotation establishing matches between XML schema components and the ontology, and the ontological knowledge itself, can be used to (quasi)automatic creation of mappings between schemas. A special attention is paid to the original concept of conditional annotations which occur in modeling of specialization.

1 Introduction

Schema matching and schema mapping are two crucial steps in developing data integration and data exchange systems, especially when schemas evolve and the data sources are considered in dynamic P2P data integration systems. In this paper we discuss the problem of automatic creation of schema mappings based on matches provided by XML schema annotations into a domain ontology, and on the ontology itself. We identify a class of XML schemas which model *specialization*, the data modeling abstraction known in conceptual database modeling. The specialization can be *disjoint* or *overlapping*. These kinds of specialization can be modeled by variety of XML schemas. To cope with the problem, so called *conditional annotations* are needed. We formalize the notion of XML schema annotation (conditional and unconditional) and propose a formalism for specifying schema mappings. Based on these notations we propose rules for representing information provided by the annotation in a form of RDFS triples. A pair of sets of RDFS triples (of the source and the target schemas) are then the input to the $GenMap$ algorithm that generates the mapping between the source and the target schemas.

Annotations are commonly used to enrich semantics of XML schemas [2,13]. Schema matching techniques received a great deal of attention and were reported in many papers and surveys ([11,5,7]). Matches create a key input to the creation of schema mappings. A formal foundation of mapping specification for relational data was proposed in [6]. An adaptation of this concept to XML data integration was discussed in [1].

R. Meersman, P. Herrero, and T. Dillon (Eds.): OTM 2009 Workshops, LNCS 5872, pp. 29–38, 2009.

In this paper we discuss an approach to manage schema mappings using a domain ontology that is used for annotating XML schemas. The information provided by the mapping, as well as the ontological knowledge, are used to infer mappings for a class of XML schemas. The existence of semantic annotation of schemas can be extensively used for many purposes concerning reorganization of mappings in response to changes in the system. New mappings can be inferred and the consistency of the set of mappings can be checked.

The contribution of this paper is the following: (1) we propose and discuss conditional annotation in the context of XML schema integration (the approach is rooted in ideas proposed in [3]), we study XML schema annotation as a way for enriching semantics of XML schemas and a way of organizing and using this knowledge; (2) we design and describe algorithms for creating XML schema mappings using the information provided by the annotation and the ontological knowledge.

The paper is organized as follows: In Section 2 we discuss and illustrate by examples the ideas underlying the research in particular the need of conditional annotations. Rules for developing RDFS triples representing the annotation are given in Section 3. The main algorithm for generating XML schema mappings is proposed in Section 4. Section 5 concludes the paper.

2 Ontology-Based XML Schema Matching – Motivating Examples

The idea of using a global domain ontology to support creation of mappings between XML schemas is depicted in Figure 1. This approach is applied in implementation of SixP2P system [4,10,9]. We assume that there is a global domain ontology containing all the information of interest to the user. Each of the sources in a P2P data integration system is understood as a local view over this ontology. The matching between XML schema elements in local sources is defined by the *annotation*. In Figure 1 there are two XML schema trees, S_1 and S_2, located at different peers, and examples of their annotations.

We assume that the global domain ontology is defined in OWL Lite [8] and consists of: *class names*, *object property names*, *datatype property names*, and *axioms*. Simple type names are primitive and user-defined simple types of XSD [12]. In SixP2P we extend the ontology with two additional construct: (1) signatures of value-conversion functions ($fullN$, $firstN$, and $lastN$ in Figure 1), and (2) definition of composed properties (not shown in Figure 1).

In Figure 1 there is an example of *unconditional annotation*. However, there is often necessity for *conditional annotation*. Consider the problem of finding a mapping between schemas S_3 and S_4 (with instances I_3 and I_4, respectively) in Figure 2. In both we have information about papers which are divided (specialized) into two classes: *conference papers* and *journal papers*. In I_3 the *type* subelement of *paper* indicates whether the *paper* element is a conference paper ($type = "c"$) or a journal paper ($type = "j"$). In I_4, information about conference and journal papers are organized in separate subtrees ($cPaper$ and $jPaper$,

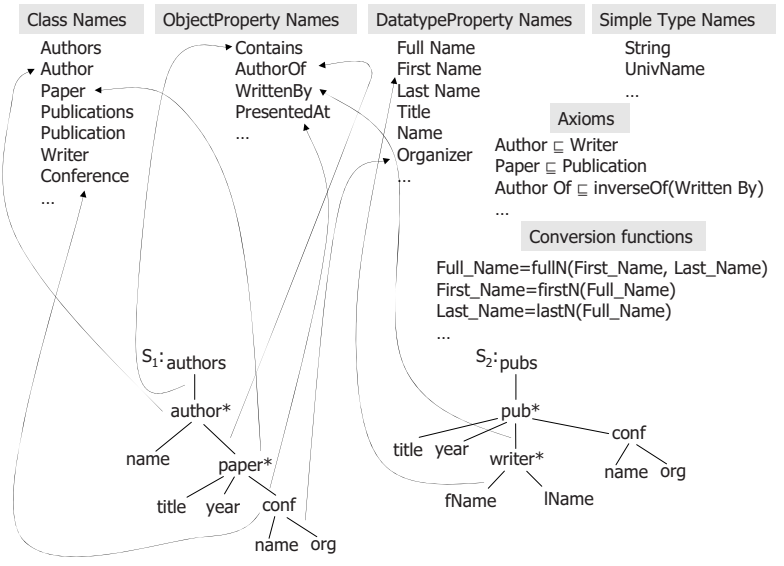

Fig. 1. Unconditional annotation of XML schema trees S_1 and S_2 with terms from a global domain ontology

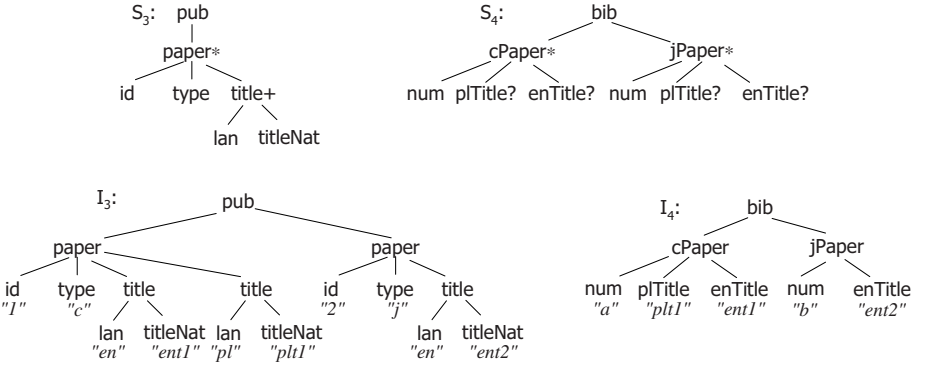

Fig. 2. Sample XML schemas and their instances

respectively). A paper can have the title written in many languages. In I_3 the value of *lan* child of *title* indicates the language ("*pl*" for Polish and "*en*" for English), and *titleNat* stores the title in this language. In S_4 there is a separate element (i.e. *plTitle* and *enTitle*) for the title written in the corresponding language. Papers in S_3 are identified by integer values of *id* while in S_4 by character strings being values of the *num* element.

In Figure 3 there are annotations of S_3 and S_4. Note that annotations of S_3 are conditional. The *paper* label is unconditionally annotated with the class

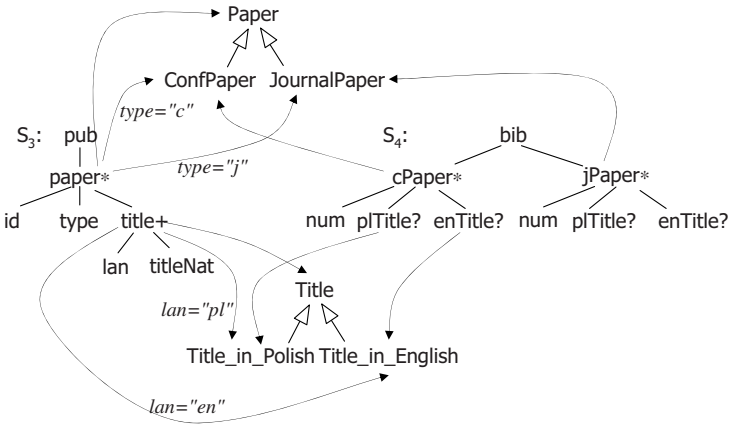

Fig. 3. Annotation of schemas S_3 and S_4

name *Paper*, and additionally may have many conditional annotations (e.g. $ConfPaper$ under the condition $type = "c"$). The *paper* element models *disjoint specialization*, since $ConfPaper$ and $JournalPaper$ are names of disjoint classes. The *title* element in S_3 models *overlapping specialization*, since a title can belong to many subclasses of the class $Title$, e.g. to $Title\_in\_Polish$, $Title\_in\_English$, etc. A national representation of the title is stored in separate instance of the tree rooted in *title* (in I_3) or within the same instance of the subtree $cPaper$ (or $jPaper$) (in I_4).

3 Generating RDFS Triples from Schema Annotation

An XML schema can be defined by DTD (*Document Type Definition* or by XSD (*XML Schema Definition*) proposed by W3C [12]. In this paper, we assume that attributes are represented by so called *terminal elements* labeled with *terminal labels* and having *simple types*. Simple types are primitive types (e.g. *xsd:string* or simple user-defined types, e.g. *titleType*). Let Lab be a set of non-terminal labels, Ter be a set of terminal labels, $Lab \cap Ter = \emptyset$, and $root \in Lab$ be a root label. Let $STypes$ be a set of simple type names.

Definition 1. *A tuple $S = (root, Lab, Ter, STypes, \rho)$ is an XML schema, where ρ is a function assigning regular expressions over $Lab - \{root\} \cup Ter$ to non-terminal labels, and simple type names to terminal labels, i.e.*

- $\rho : Lab \to Reg$,
- $\rho : Ter \to STypes$,
- *the set Reg of regular expressions is defined by the grammar:*
  $e ::= A \mid l \mid e? \mid e^* \mid e^+ \mid e, e \mid e + e$, *where $l \in Lab - \{r\}$, $A \in Ter$.*

An XML schema can be annotated in a domain ontology O. We will use names of three OWL categories: *classes*, *objectProperties*, and *datatypeProperties*. The following rules are obeyed:

- Non-terminal labels are annotated with OWL *class* names; the annotation may be constrained by a simple condition of the form: $A = const$ or $A \neq const$, built out from a terminal label going out from the annotated label and a constant *const*.
- Edges between two non-terminal labels are annotated with OWL *object property* names, where the predecessor is the domain and the successor – the range of the property.
- Edges between non-terminal and terminal labels are annotated with OWL *datatype property* names, where the domain is a non-terminal label and the range is a terminal label.

Let S be an XML schema, $CNames$, $OPNames$ and $DTPNames$ be sets of, respectively, class names, object property names and datatype property names in O.

Definition 2. *A (conditional) annotation of S with ontology O is a tuple $\mathcal{A}_{S,O} = (Cond, \{\lambda_\alpha\}_{\alpha \in Cond})$, where: (1) $Cond$ is a set of simple conditions over S, ($\texttt{TRUE} \in Cond$), (2) λ_α is a conditionally annotating function ($\lambda_{\texttt{TRUE}}$ is referred to as the unconditional annotation) i.e.:*

- $\lambda_\alpha : Lab \rightarrow CNames$,
- $\lambda_{\texttt{TRUE}} : Lab \times Lab \rightarrow OPNames$,
- $\lambda_{\texttt{TRUE}} : Lab \times Ter \rightarrow DTPNames$.

For $Lab = \{l_1, ..., l_N\}$, let $Cond(l_i)$ be a set of conditions for annotating l_i, $1 \leq i \leq N$. Then $\sigma = (\sigma[l_1], ..., \sigma[l_N])$ consisting of single conditions (possibly TRUE) for any label, is called *conditional tuple*. Let σ be a conditional tuple. By $\mathcal{T}_S(\sigma)$ we will denote the set of all RDFS triples created by means of rules given in Figure 4. The sets $\mathcal{T}_{S_1}$ and $\mathcal{T}_{S_2}$ of RDFS triples in Figure 5 are produced by means of rules in Figure 4 for schemas S_1 and S_2 from Figure 1.

For annotation of S_3 we have: (1) *paper* is annotated with disjoint classes, and $Cond(paper) = \{type = "c", type = "j"\}$; (2) *title* is annotated with overlapping classes, and $Cond(title) = \{lan = "pl", lan = "en"\}$.

In general, if there are N conditionally annotated non-terminal labels $l_1, ..., l_N$ in a schema S, and the set $Cond(l_i)$ contains k_i conditions, then the number of all unconditional annotations is

$$K = \prod_{i=1}^{N} k_i.$$

Thus, there are four sets of triples, $\mathcal{T}_{S_3}(\alpha, \beta)$, derived from annotation of S_3, where $\alpha \in Cond(paper)$, and $\beta \in Cond(title)$. For example, first of them can be:

$\mathcal{T}_{S_3}(type = "c", lan = "pl")$:

(1) $\{(Publications, Contains, ConfPaper)$ $(T1)$
(2) $(ConfPaper, PaperId, String)$ $(T3)$
(3) $(ConfPaper, PaperType, "c")$, $(T4)$
(4) $(ConfPaper, TitleInPolish, Title\_in\_Polish)$, $(T2)$
(5) $(Title\_in\_Polish, LanguageOfTitle, "pl")$, $(T4)$
(6) $(Title\_in\_Polish, Value, String)$, $(T3)$
(7) $(ConfPaper, Title\_in\_Polish, String)$ $(T5)\}$

(T1) $(Root, R, C) \in \mathcal{T}_S(\sigma)$, if
$\lambda_{\sigma[root]}(root) = Root$ and $\exists l \in Lab\ (\lambda_{\texttt{TRUE}}(root, l) = R$ and $\lambda_{\sigma[l]}(l) = C)$;
(T2) $(C, R, C') \in \mathcal{T}_S(\sigma)$, if $\exists(\_, \_, C) \in \mathcal{T}_S(\sigma)$ and
$\exists l, l' \in Lab\ (\lambda_{\sigma[l]}(l) = C$ and $\lambda_{\sigma[l']}(l') = C'$ and $\lambda_{\texttt{TRUE}}(l, l') = R)$;
(T3) $(C, D, t) \in \mathcal{T}_S(\sigma)$, if $(C = Root$ or $\exists(\_, \_, C) \in \mathcal{T}_S(\sigma))$ and
$\exists l \in Lab, A \in Ter\ (\lambda_{\sigma[l]}(l) = C$ and $\lambda_{\texttt{TRUE}}(l, A) = D$ and $\rho(A) = t)$;
(T4) $(C, D, "a") \in \mathcal{T}_S(\sigma)$, if $(C = Root$ or $\exists(\_, \_, C) \in \mathcal{T}_S(\sigma))$ and
$\exists l \in Lab, A \in Ter, A = "a" \in \sigma[l]\ (\lambda_{\sigma[l]}(l) = C$ and $\lambda_{\texttt{TRUE}}(l, A) = D)$;
(T5) $(C, D, t) \in \mathcal{T}_S(\sigma)$, if
$(C, R, C') \in \mathcal{T}_S(\sigma)$ and $(C', D', t) \in \mathcal{T}_S(\sigma)$ and $D = R \circ D' \in O$.

Fig. 4. Rules deriving the set of RDFS triples from an annotated XML schema

$\mathcal{T}_{S_1}$:	$\mathcal{T}_{S_2}$:
$(Authors, Contains, Author)$	$(Publications, Contains, Publication)$
$(Author, Full\_Name, String)$	$(Publication, Title, String)$
$(Author, AuthorOf, Paper)$	$(Publication, Year, String)$
$(Paper, Title, String)$	$(Publication, WrittenBy, Writer)$
$(Paper, Year, String)$	$(Writer, First\_Name, String)$
$(Paper, PresentedAt, Conference)$	$(Writer, Last\_Name, String)$
$(Conference, Name, String)$	$(Publication, PresentedAt, Conference)$
$(Conference, Organizer, String)$	$(Conference, Name, String)$
	$(Conference, Organizer, String)$

Fig. 5. RDFS triples derived from unconditional annotation of S_1 and S_2

Along with the triples, we write identifiers of the inferring rules, $(T1) - (T5)$. The triple (7) can be derived in force of $(T5)$ if in ontology O the datatype property $Title\_in\_Polish$ is defined as the composition of the object property $TitleInPolish$ and the datatype property $Value$, i.e. $Title\_in\_Polish = TitleInPolish \circ Value \in O$.

4 Generating Schema Mappings from Annotations and Ontology

4.1 Tree Pattern-Based XML Schema Mappings

A schema mapping is a specification describing how data structured under a source schema is to be transformed into data structured according to a target

schema. To define such transformation we will use *tree patterns* (TPs) and *tree-pattern formulas* (TPFs) [1,10]. In fact, both TPs and TPFs are formulas in XPath so their standard semantics is precisely defined.

We will say that a tree-pattern formula $\phi(u_1, ..., u_n)$, where u_i is a variable or a constant, is defined over XML schema S, if defines a subtree or a set of subtrees conforming to S. Mappings will be written in the datalog-like style.

Definition 3. *A* schema mapping *(or* mapping*)* $\mathcal{M}_{S,T}$ *from a source XML schema* S *to a target XML schema* T *is a set of mapping rules, where a mapping rule is an expression of the form:*

$$\psi(\mathbf{u}) :- [\neg]\phi_1(\mathbf{u}_1), [\neg]\phi_2(\mathbf{u}_1), ..., [\neg]\phi_k(\mathbf{u}_k), \chi(\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_k),$$

where:

- $\psi(\mathbf{u})$*, and* $\phi_i(\mathbf{u}_i)$ *are TPFs over* T *and* S*, respectively,* $1 \leq i \leq k, k \geq 0$*;*
- $var(\mathbf{u}) \subseteq var(\mathbf{u}_1) \cup, ... \cup var(\mathbf{u}_k)$ *– each variable occurring in* $\mathbf{u}$ *must occur in at least one of* $\mathbf{u}_1, ..., \mathbf{u}_k$*;*
- $\chi(\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_k)$ *is a conjunction of atomic formulas over variables and constants,*
- *the comma sign (,) between formulas denotes conjunction.*

Example 1. The mapping from S_3 to S_4 (Figure 2) includes among others the following two mapping rules:

$$\begin{aligned} m^1_{S_3,S_4} : & \, bib[cPaper[num = funNum(x_1), plTitle = x_2, enTitle = x_3]] :- \\ & \, pub[paper[id = x_1, type = "c", title[lan = "pl", titleNat = x_2]], \\ & \, pub[paper[id = x_1, type = "c", title[lan = "en", titleNat = x_3]] \\ m^2_{S_3,S_4} : & \, bib[cPaper[num = funNum(x_1), plTitle = x_2, enTitle = x_3]] :- \\ & \, pub[paper[id = x_1, type = "c", title[lan = "pl", titleNat = x_2]], \\ & \, \neg pub[paper[id = x_1, type = "c", title[lan = "en"]], x_3 = \bot \end{aligned}$$

Function $funNum(x_1)$ in heads of the rules converts value of x_1 into the value of the type assigned to *num*. In the second rule, it is tested whether the conference paper has the title written in English, if not then the null ($\bot$) value is assigned to *enTitle* in the target instance.

Semantics of a mapping is defined as the union of all sets of tuples of values produced by mapping rules constituting the mapping [10].

4.2 Mapping Generating Algorithm

The following algorithm generates a mapping between two unconditionally annotated schemas S and T.

Algorithm 1. (*Generating a mapping,* $GenMap(\mathcal{T}_S, \mathcal{T}_T)$)
Input: $\mathcal{T}_S$ – a set of RDFS triples for a source schema S, $\mathcal{T}_T$ – a set of RDFS triples for a target schema T, functions λ_S and λ_T annotating S and T in O.
Output: A mapping $\mathcal{M}_{S,T}$, initially empty.

(1) If all triples in $\mathcal{T}_T$ are resolved then return $\mathcal{M}_{S,T}$ and stop, otherwise go to step (2).
(2) If all terminal triples in $\mathcal{T}_T$ are resolved then go to step (3), otherwise get the first not resolved terminal triple and denote it τ. Mark τ as resolved.
  (2.1) if $\tau = (C, D, t)$, where $C = \lambda_T(l)$ and $D = \lambda_T(l, A)$, and t is a type, then
    (2.1.1) if $(C', D', t') \in \mathcal{T}_S$, $C' = \lambda_S(l') \sqsubseteq C$ and $D' = \lambda_S(l', A') \sqsubseteq D$ then
      $m := l[A = u] :- l'[A' = u]$,
    where u if a variable name if t' is a type equal to t, and a constant "a" if t' is "a";
    (2.1.2) else if $(C', D'_1, t_1), \ldots, (C', D'_n, t_n) \in \mathcal{T}_S$, where $C' = \lambda_S(l') \sqsubseteq C$ and $\lambda_S(l', A'_1) = D'_1, \ldots \lambda_S(l', A'_n) = D'_n$, $D = f(D'_1, \ldots, D'_n) \in O$ then
      $m := l[A = f(u_1, \ldots, u_n)] :- l'[A'_1 = u_1, \ldots, A'_n = u_n]$,
    where u_i is as was explained in (2.1.1);
    (2.1.3) else go to step (2);
    insert m into $\mathcal{M}_{ST}$ and go to step (2).
  (2.2) if $\tau = (C, D, "a")$, where $C = \lambda_T(l)$ and $D = \lambda_T(l, A)$, then
    $m := l[A = "a"] :- \texttt{TRUE}$,
    insert m into $\mathcal{M}_{ST}$ and go to step (2).
(3) If all elements in $\mathcal{M}_{S,T}$ have been processed then go to step (1), otherwise:
  (3.1) if $m_1 = l[\psi_1] :- l'[\phi_1] \in \mathcal{M}_{S,T}$ and $m_2 = l[\psi_2] :- l'[\phi_2] \in \mathcal{M}_{S,T}$ then
    $m := l[\psi_1, \psi_2] :- l'[\phi_1, \phi_2]$;
  (3.2) else if $m_1 = l[\psi] :- l'[\phi] \in \mathcal{M}_{S,T}$ and $(C'_1, R', C') \in \mathcal{T}_S$ and $(C_1, R, C) \in \mathcal{T}_T$ and $R' = \lambda_S(l'_1, l') \sqsubseteq R = \lambda_T(l_1, l) \in O$ then
    $m = l_1[l[\psi]] :- l'_1[l'[\phi]]$;
  (3.3) else if $m_1 = l_1[\psi_1] :- l'_1[\phi_1] \in \mathcal{M}_{S,T}$, $m_2 = l_2[\psi_2] :- l'_2[\phi_2] \in \mathcal{M}_{S,T}$ and $(C'_1, R', C') \in \mathcal{T}_S$ and $(C_1, R, C) \in \mathcal{T}_T$ and $R' = \lambda_S(l'_1, l'_2)$ and $R = \lambda_T(l_1, l_2)$ and $R' \sqsubseteq insertOf(R) \in O$ then
    $m := l_1[\psi_1, l_2[\psi_2]] :- l'_2[\phi_2, l'_1[\phi_1]]$.
  (3.4) else if $m_1 = l[\psi] :- l'[\phi] \in \mathcal{M}_{S,T}$ and $\lambda_T(l_1, l) = "Contains" \in O$
    $m := l_1[l[\psi]] :- l'[\phi]$.
  (3.5) else if $m_1 := l[\psi] :- l'[\phi] \in \mathcal{M}_{S,T}$ and $\lambda_S(l'_1, l') = "Contains" \in O$
    $m := l[\psi] :- l'_1[l'[\phi]]$.
  (3.6) else if $m_1 = l[\psi_1] :- \phi_1 \in \mathcal{M}_{S,T}$ and $m_2 := l[\psi_2] :- \phi_2 \in \mathcal{M}_{S,T}$ then
    $m := l[\psi_1, \psi_2] :- \phi_1, \phi_2$;
  insert m to $\mathcal{M}_{S,T}$, remove m_1 or also m_2 from $\mathcal{M}_{S,T}$, go to step (3).

Let S be a conditionally annotated schema and $\mathcal{A}_{S,O} = (Cond, \{\lambda_\alpha\}_{\alpha \in Cond})$ be an annotation of S, and T be a schema annotated unconditionally, i.e. $\mathcal{A}_{T,O} = (\lambda)$. Then:

1. For any selection of conditions $\sigma = (\alpha_1, ..., \alpha_n)$, $\alpha_i \in Cond(l_i)$, the set $\mathcal{T}_S(\sigma)$ of RDFS triples for the σ-selection of S is produced. There are $\prod_{i=1}^{N} m_i$ such selections, where $m_i = count(Cond(l_i))$.
2. The set $\mathcal{T}_T$ of RDFS triples for T is computed.

3. The algorithm $GenMap(\mathcal{T}_S(\sigma), \mathcal{T}_T)$ is used to generate N mapping rules from S to T.
4. The mapping rules are used to create the final mapping $\mathcal{M}_{S,T}$.

The final mapping depends on the kinds of specializations determined by classes assigned to labels of S by the conditional annotations. We will define the mapping for the case when two labels in S are conditionally annotated: one label (say l_1) defines a disjoint, and the other (say l_2) an overlapping specialization. Let: $Cond(l_1) = \{\alpha_1, \alpha_2\}$, $Cond(l_2) = \{\beta_1, \beta_2\}$, and

- $m_{S,T}(\alpha, \beta) := \psi_{\alpha,\beta} :- \phi_{\alpha,\beta}$, where $\alpha \in Cond(l_1)$, $\beta \in Cond(l_2)$, and $m_{S,T}(\alpha, \beta) = GenMap(\mathcal{T}_S(\alpha, \beta), \mathcal{T}_T)$.

Then

$$\begin{aligned}\mathcal{M}_{S,T} = \{&\psi_{\alpha_1,\beta_1,\beta_2} :- \phi_{\alpha_1,\beta_1}, \phi_{\alpha_1,\beta_2},\\ &\psi_{\alpha_1,\beta_1,\beta_2} :- \phi_{\alpha_1,\beta_1}, \neg\phi_{\alpha_1,\beta_2}, x_{\beta_2} = \bot,\\ &\psi_{\alpha_1,\beta_1,\beta_2} :- \neg\phi_{\alpha_1,\beta_1}, \phi_{\alpha_1,\beta_2}, x_{\beta_1} = \bot,\\ &\psi_{\alpha_2,\beta_1,\beta_2} :- \phi_{\alpha_2,\beta_1}, \phi_{\alpha_2,\beta_2},\\ &\psi_{\alpha_2,\beta_1,\beta_2} :- \phi_{\alpha_2,\beta_1}, \neg\phi_{\alpha_2,\beta_2}, x_{\beta_2} = \bot,\\ &\psi_{\alpha_2,\beta_1,\beta_2} :- \neg\phi_{\alpha_2,\beta_1}, \phi_{\alpha_2,\beta_2}, x_{\beta_1} = \bot\}\end{aligned}$$

Every mapping rule involves one condition used in the annotation with a class name from a disjoint specialization, and all conditions applied in annotations with class names from an overlapping specialization. At least one component in the body is positive (indicating that an object can be specialized in at least one from all overlapping classes), and the rest can be negative (a negative component indicates that the corresponding class might not contain the object). If $\phi_{\alpha,\beta}$ is negative and β corresponds to an overlapping specialization, then the conjunct $x_\beta = \bot$ is added to the body. If β is of the form $A =$ "a", then x_β is the variable associated with the path ending with A.

The mapping from S_3 to S_4 considered in Example 1, consists of six mapping rules, i.e.: $\mathcal{M}_{S_3,S_4} = \{m^1_{S_3,S_4}, m^2_{S_3,S_4}, m^3_{S_3,S_4}, m^4_{S_3,S_4}, m^5_{S_3,S_4}, m^6_{S_3,S_4}\}$. Two first of them were given in Example 1, and the third is:

$$\begin{aligned}m^3_{S_3,S_4} = {}& bib[cPaper[num = funNum(x_1), plTitle = x_2, enTitle = x_3]] :-\\ &\neg pub[paper[id = x_1, type = \text{"}c\text{"}, [title[lan = \text{"}pl\text{"}]]]], x_2 = \bot,\\ &pub[paper[id = x_1, type = \text{"}c\text{"}, [title[lan = \text{"}en\text{"}, titleNat = x_3]]]]\\ &\ldots\end{aligned}$$

In the similar way the remainder three mapping rules corresponding to the condition $type =$ "j" can be created.

The mapping rules can be implemented by translating into XQuery programs. Such the translation was proposed, for example, in our previous papers [4,10].

5 Conclusion

In the paper we propose a method for managing XML schema mappings and annotations in data integration systems. The method is of special importance

when the integration involves P2P connected sources and new peers can enter the system with new schemas and old schemas can evolve. We identified an important class of XML schemas for which a conditional annotation is necessary. We discussed constructs in a domain ontology which are needed to annotation of XML schemas in the context of data integration. Next, we used the ontological knowledge and the matches between schema components and terms in the ontology to derive conditional matches between schemas. The conditional and unconditional schema matches are the base for automatic generation of schema mappings.

Acknowledgement. The work was supported in part by the Polish Ministry of Science and Higher Education under Grant 3695/B/T02/2009/36.

References

1. Arenas, M., Libkin, L.: XML Data Exchange: Consistency and Query Answering. In: PODS Conference, pp. 13–24 (2005)
2. Beneventano, D., Bergamaschi, S.: The MOMIS methodology for integrating heterogeneous data sources. IFIP Congress Topical Sessions, 19–24 (2004)
3. Bohannon, P., Elnahrawy, E., Fan, W., Flaster, M.: Putting Context into Schema Matching. In: Proc. of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, pp. 307–318. ACM, New York (2006)
4. Brzykcy, G., Bartoszek, J., Pankowski, T.: Schema Mappings and Agents Actions in P2P Data Integration System. Journal of Universal Computer Science 14(7), 1048–1060 (2008)
5. Doan, A., Halevy, A.Y.: Semantic Integration Research in the Database Community: A Brief Survey. AI Magazine 26(1), 83–94 (2005)
6. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.C.: Composing Schema Mappings: Second-Order Dependencies to the Rescue. In: PODS, pp. 83–94 (2004)
7. Madhavan, J., Bernstein, P.A., Doan, A., Halevy, A.Y.: Corpus-based Schema Matching. In: Proceedings of the 21st International Conference on Data Engineering, ICDE, pp. 57–68. IEEE Computer Society, Los Alamitos (2005)
8. OWL Web Ontology Language Overview (2004), `http://www.w3.org/TR/owl-ref`
9. Pankowski, T.: Query propagation in a P2P data integration system in the presence of schema constraints. In: Hameurlain, A. (ed.) Globe 2008. LNCS, vol. 5187, pp. 46–57. Springer, Heidelberg (2008)
10. Pankowski, T.: XML data integration in SixP2P – a theoretical framework. In: EDBT Workshop Data Management in P2P Systems (DAMAP 2008), pp. 11–18. ACM Digital Library, New York (2008)
11. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. The VLDB Journal 10(4), 334–350 (2001)
12. W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes (2009), `www.w3.org/TR/xmlschema11-2`
13. Xiao, H., Cruz, I.F.: Integrating and Exchanging XML Data Using Ontologies. In: Spaccapietra, S., Aberer, K., Cudré-Mauroux, P. (eds.) Journal on Data Semantics VI. LNCS, vol. 4090, pp. 67–89. Springer, Heidelberg (2006)