Data Translation Between Taxonomies

Sérgio Luis Sardi Mergen and Carlos Alberto Heuser

Universidade Federal do Rio Grande do Sul, Av. Bento Gonalves,
9500 Porto Alegre - RS - Brasil
{mergen, heuser}@inf.ufrgs.br

Abstract. The task of translating data from one schema into another is usually performed with the help of information stating how the elements between two schemas correspond. Translation mechanisms can use this information in order to identify how instances of a source schema must be translated. We claim that a uniform matching approach, where instances of a source classes are always translated into the same target classes, may not represent the reality, specially when the schemas involved describe taxonomies. In this paper we demonstrate taxonomies that support our idea, and propose the usage of a conditional matching approach to improve the accuracy of taxonomical instances translation.

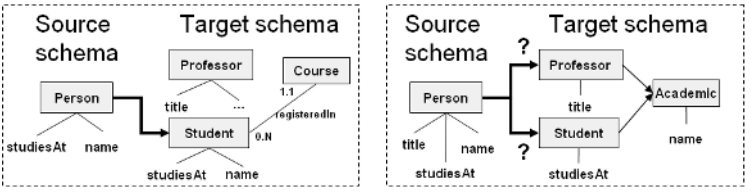
1 Introduction

Data translation plays a fundamental role in the information integration area. It is up to a data translation process the task of converting data from its native storage representation into another representation.

There are several applications where data translation mechanisms represent a crucial task. In a mediated system[18, 3], when a user submits a query, a usual approach would be to translate the results coming from the sources into a format expected by the user. When migrating databases from an old version to a new version, a commonly faced challenge involves data translation, where the arising question is how to covert data stored in the old schema into the new schema, specially when dealing with heterogeneous schemas, or even worse, when the schemas are represented in different models. Even in data warehouses, the incoming data must be processed in some way before storage; e.g. data may be filtered, and relations may be joined or aggregated. As the data is copied from the sources, it may need to be transformed in certain ways to make all data conform to the schema at the data warehouse.

Independently of the underlying mechanism that performs the data translation, this processing relies on a general idea that is to identify the matches between the schemas and apply transformation rules to translate instances of the source into instances of the target[14, 1, 4, 5] . Figure 1(a) illustrates an abstract example where two schemas are matched. The schemas are represented as classes with properties. The match is represented by the bold line, which indicates that instances of *Person* in the source should be translated into instances of *Student* in the target.

E. Dubois and K. Pohl (Eds.): CAiSE 2006, LNCS 4001, pp. 111–124, 2006.



(a) Non-taxonomical example (b) Taxonomical example

Fig. 1. Matching examples

At this time, there is no fully automatic semantic integration approach[10], which means the user is still responsible for identifying the matches and creating the translation rules. Nevertheless, this often laborious task can be aided with the help of schema matching techniques that suggest to the user the most probable ways to do the matching, and possibly, which transformations to apply on the instances.

However, when it comes to translate data between two taxonomies, there is a gap that neither schema matching techniques nor data translation mechanisms seem to support. Take for instance the taxonomies depicted in figure 1(b). The target schema presents the academic staff in a taxonomical model, where classes are described in terms of specialization and generalization of other classes. In the source schema, the $Person$ class is used to store both $Students$ and $Professors$ instances, undiscriminatingly. Observing the schema level alone it is not clear if an instance of $Person$ in the source is in fact a student or a professor.

On the other hand, the instance level may provide some relevant cues on how to translate the instances. For instance, the *studiesAt* and *title* properties of the person class are disjoint by nature. Thus, it is expected that every instance of $Person$ contains at most one of these two properties. Taking this reasoning one step further, we conclude that the presence (or absence) of some particular property in an instance may be strong indicators of that instance semantics.

Based on this reasoning, we present an approach to the translation of taxonomies, based on the adoption of a simple mechanism, that we call translation script. A translation script is generated based on pre-computed matches between the schemas.

A translation script is composed of alternative matches for every source class. Each alternative match is associated to a condition that indicates whether the match can be used in the translation of a particular instance or not. During translation, the script can be used to guide the translation process by determining which of the alternative matches should be used when translating a specific instance.

Our contribution is twofold. Firstly, the translation script is a novel method for data translation, that is specifically useful for translating data between taxonomies. Secondly, we demonstrate that the generation of this translation script is straightforward, once it simply requires that pre-computed matches between the schemas involved in the translation are provided.

The rest of the paper is organized as follows: In section 2 we show approaches related to the issue of matching taxonomies along with existing mechanisms used for data translation. Section 3 presents motivating examples that illustrate how alternative matches can be used to improve the translation accuracy. In section 4 we define a translation script and describe the algorithm used to generate a translation script based on pre-computed matches. Section 5 demonstrates a case study where we suggest the use of a translation script on real world ontologies. Section 6 briefly describes a different approach that is also based on a translation script to perform data translation. Final conclusions are presented in section 7.

2 Related Work

When studying translation of taxonomical data, related work can be classified in two groups: Schema matching approaches and data translation approaches.

In general, matching taxonomies can be seen as a broader problem, which is the matching of ontologies. Most of the integrated approaches for ontology matching are based on the idea of combining different strategies and measures of similarity. In the vast literature on ontology matching, we have found several strategies to match taxonomies[6, 11, 12, 15, 9], but none of them seems to worry about the existence of alterative matches to a source class(concept).

The S-Match system [9] takes the process of matching taxonomies one step further by allowing matches with semantic operators other than equivalence. Given two concepts, the algorithm assigns a semantic relation that can be of equivalence (=), generalization($\sqsupseteq$),specialization($\sqsubseteq$), mismatch($\perp$), or overlap($\sqcap$). The matches are computed by selecting the strongest semantic relation discovered by the algorithm between two ontology concepts. This kind of matching better describes the semantic of the correspondences between taxonomies, but still presents the drawback of allowing one single match per concept.

In [13] the authors also use several semantic operators when matching schemas. However, they claim that sometimes it is not possible to identify a single match between each pair of concepts of two schemas. They present an approach for schema integration that allows alternative matches for each pair of concepts, where a match is given a level of belief. The matches, referred to as uncertain matches, are computed in the schema matching process and propagated to a schema merging process that is responsible for generating an integrated schema. Similar work is presented in [8], with the limitation that matches can only be done through the equivalence operator. The usage of uncertain matches resembles the usage of the alternative matches suggested in our work, in a sense that both kind of matches are used to dynamically resolve semantic matching problems. However, uncertain matches are applicable for semantic reconciliation of schemas, while our approach is applicable for data translation.

The usage of uncertainty is also present in [17], only this time the uncertainty lies on the data level, instead of the schema level. Their approach, based on

the probability theory, allows the user to query for uncertain data in a data integration environment.

As for data translation approaches, most of them [14, 1, 4, 5] rely on the usage of matching rules to perform data translation. Despite some small variations among the approaches, they all share the same idea on what the matching rules should express, which are the matches between the schemas, and the transformations that must be performed when translating the instances. The instances can be transformed either by value modification or by schema restructuring. In [2] the authors introduce a middleware data model that supports the declaration of correspondences between two different representations (schemas). They also describe some practical cases where the correspondences are automatically turned into translation rules. One of the benefits of their approach is that the specification of correspondences can be used to perform translation of data in both directions.

For the best of our knowledge, none of the translation approaches consider the usage of alternative matches, where a match to a given source class may vary during translation. As far as we know, the existing approaches can use the absence of properties inside a given instance only to indicate that the instance should not be translated. Using the presence/absence of properties values of a source instance to decide whether one given class or another should be selected as the target is an entirely new idea.

We claim that our approach to data translation could be incorporated in data translation mechanisms, as a way to dynamically select one of the alternative matches, based on the presence or absence of property values in a given instance.

3 Running Examples of Taxonomy Translation

In order to perform data translation, it is necessary to understand how the source and target schema correspond to each other. Based on this understanding, generally expressed in the form of matchings, it is possible to translate instances of the source schema into instances of the target schema. In this context, we argue that the usage of a uniform matching - where one single match suffices for translating every instance of a source class - is questionable, specially when the schemas describe taxonomies. In such cases, the translation of the instances can vary, according to the values actually being translated.

Next, we show cases where the translation of taxonomical instances may result in an erroneous classification if the translation uses a uniform mapping strategy instead of a flexible one. We also demonstrate how the translation can be semantically improved if the idea of a translation script is used to guide the translation process.

To start, consider the academic staff taxonomies presented in figure 2. Consider a taxonomy-aware matcher able to realize that $\Gamma_2.Tenured$ is a better match to $\Gamma_1.Professor$, instead of its polysemous $\Gamma_2.Professor$. Additionally, the matcher was able to find the properties correspondences, where *nationality* matches *nationality*, *institution* matches *lecturesAt* and *tenuredSince* matches *isTenuredSince*.

This match solves the problem of translating the instances of professors in Γ_1 without information losses - since all properties of the source are translated - but it does not guarantee there will not be semantic losses. Suppose that some professors stored in Γ_1 are tenured, while others are not. During translation, if the match between $\Gamma_1.Professor$ and $\Gamma_2.Tenured$ is used, every instance of $professor$ in Γ_1 ends up translated into instances of $\Gamma_2.Tenured$.

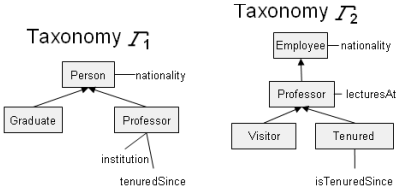


Fig. 2. Academic staff taxonomies

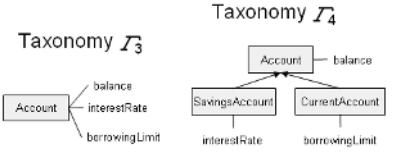


Fig. 3. Bank account taxonomies

However, this is not the expected classification, given that not every professor is tenured. One way to differentiate professors from tenured professors in Γ_2 is that professors lack the definition of properties that are exclusive to tenured professors, such as the property $isTenuredSince$.

In this context, we propose a solution that does not hardwire a match to a specific source class, but allows a range of alternative matches. During instance translation, it is possible to choose one among several alternative matches, by using some inference mechanism that analyzes the presence or absence of properties values in the instance being translated. For the example above, a $Professor$ instance in Γ_1 is translated into a $Tenured$ instance in Γ_2, when the property $tenuredSince$ has a value assigned to it, and is translated into a $Professor$ instance in Γ_2 otherwise. Note that the best that traditional translation mechanisms could do in this case is to invalidate the translation if some of the properties are missing.

We could go even further and say that a professor is a visitor, if we knew that all Γ_2 visitors come from abroad. This kind of information can be embedded in the schema itself, in the form of constraints, if the taxonomy supports this kind of constraint. Another way to obtain additional knowledge from the taxonomies is by discovering data patterns in some sample instances. A data pattern could state the fact that every visitor is actually a professor whose country of origin is different from the country in which the university is located.

Despite the presumable benefits from using existing or computed constraints to improve data translation, this kind of analysis is out of the scope of this paper. Our commitment is on demonstrating that the semantics of the translation can be improved by using an approach that is less sophisticated, but still effective, based solely on the presence/absence of property values in the instances.

Figure 3 shows another example where the presence/absence of property values can eliminate translation ambiguity. In this case the taxonomies describe

bank accounts. Notice that the computation of a match for the *Account* class in Γ_3 is even more dubious that the prior example, since the properties declared in the *Account* class in Γ_3 are split across the classes in Γ_4.

Again, exploiting the presence of properties values inside bank account instances could improve the quality of the translation, whereas a traditional translation approach would fail. For instance, an account in Γ_3 with no value for *interestRate* and *borrowingLimit* fits the description of an *Account* in Γ_4.

Moreover, accounts in Γ_3 with all properties defined but *borrowingLimit* could be translated into a *SavingsAccount* in Γ_4. Likewise, accounts in Γ_3 with all properties defined but *interestRate* could be translated into a *CurrentAccount* in Γ_4.

4 The Translation Script Approach

In this section we present the definition of a translation script, along with examples that demonstrate how a translation script can be used during the translation of taxonomies. We start this section with the definition of a taxonomy.

Definition 1 (Taxonomy). *Let* $\Gamma =< C, P, Prop(c_i), sup(c_i) >$ *be a taxonomy, where* C *is a set of classes* $\{c_i\}$ *and* P *is a set of properties* $\{p_i\}$.

Further, let $Prop(c_i)$ *be a function that returns the set of properties of* c_i. *Additionally, let* $sup(c_i)$ *be a function that returns the immediate super-class of* c_i.

Having defined this, we have that $if\ f\ c_i = sup(c_j)$, *then* $Prop(c_i) \subseteq Prop(c_j)$.

A taxonomy is represented as a hierarchy of classes, where each class contains a set of properties. A class shares its properties with each of its sub-classes. For our purposes, it suffices to define properties merely as being part of a class. Additional constraints of a class/property composition, such as the cardinality, are not defined since they are not exploited by our translation mechanism.

For the rest of the paper, we use Γ_s to refer to a source taxonomy, while Γ_t is used to denote a target taxonomy.

The translation script is generated based on an input matching that describes how classes of a source taxonomy correspond to classes of a target taxonomy. The computation of an input matching is out of the scope of this paper, but there are several approaches in the literature that handle class to class matching, as described in section 2. We define the input matching as follows:

Definition 2 (Input Matching). *The input matching* M *is a set of matches* $\{m_i\}$. *Let* m_i *be a tuple* $<c_s, c_t, \Psi>$, *where:*

- $c_s \in C_s$, *having* $C_s \in \Gamma_s$,
- $c_t \in C_t$, *having* $C_t \in \Gamma_t$,
- Ψ *is a set of property matchings* $\psi =<p_s, p_t>$, *where* $p_s \in Prop(c_s)$ *and* $p_t \in Prop(c_t)$. *Further, if* $\exists\ \psi_i =<p_{si}, p_{ti}> \in \Psi$, *than* $\nexists\ \psi_j =<p_{sj}, p_{tj}> \in \Psi$, *such that* $\psi_i \neq \psi_j$ *and* $(p_{si} = p_{sj}$ *or* $p_{ti} = p_{tj})$.

A match is composed by a source class, a target class and a set of correspondences between the classes' properties. Within a single class to class match, only one-to-one property matches are allowed. This restriction is, in fact, an expression of the

local one-to-one match cardinality restriction[16], that prevents the occurrence of matches where there is no direct correspondence between elements of the source and target.

This kind of match, also referred to as indirect match[7], occurs, for instance when a source property is actually a composition of two target properties (eg. *name* in the source and a concatenation of $firstName$ and $lastName$ in the target). Our translation script generation approach currently does not accept indirect matches as part of the input matching. The inability to handle such sort of match is a limitation that we expect to overcome in the near future.

The listing below shows an input matching used to match the taxonomies described in figure 2. Observe that the input matching is valid according to definition 2.

$$m_1 = \{\Gamma_1.Person, \Gamma_2.Employee, [(nationality, nationality)]\}$$
$$m_2 = \{\Gamma_1.Professor, \Gamma_2.Tenured, [(nationality, nationality), (institution, lecturesAt), (tenuredSince, isTenuredSince)]\}$$

Listing 1.1. Input matching for the taxonomies of figure 2

The definition 3 describes a translation script. Notice that a translation script is actually an extension of the input matching, where we have κ to express the condition that must be satisfied so that one particular match can be considered valid. A condition is expressed as a set of properties, where all properties belong to the source class. Since the matches are associated to a condition, they are referred to as conditional matches.

Definition 3 (Translation Script). *A translation script M' is a list of conditional matches $[m'_1, m'_2, ..., m'_n]$, where m'_l is a tuple on the form $m'_l = <m, \kappa>$, having κ as a set of properties $\{p_i\}$, such that $p_i \subseteq Prop(m.c_s)$ and $\exists <p_i, p_j> \in m.\Psi$.*

When translating instances, a conditional match can be interpreted as follows: if at least one of the properties within the match κ condition is present (has a value) in an instance, then the κ condition is satisfied, and the match can be used in the translation.

Inside a translation script, the conditional matches that refer to the same source class must be ordered according to definition 4. The matches are sorted with respect to the target class, and go from the more subsumed target class, as the first element, to the less subsumed target class, as the last element. The relation of subsumption between two classes is directly related to the hierarchical relation between the classes. The deeper a target class is in the hierarchy, the higher will be its sorting position.

Definition 4 (Property Presence Sorting). *Having $m'_i = <m_l, \kappa_i> \in M'$ and $m'_j = <m_n, \kappa_j> \in M'$, and having $m_l.c_s = m_n.c_s$, then $i < j$ if $m_l.c_t$ is a subsumption of $m_n.c_t$.*

Furthermore, the conditional matches that refer to the same source class have mutually exclusive κ conditions, as determined in definition 5. It is important that both definitions 4 and 5 are respected so the execution of the translation script may succeed.

Definition 5 (Mutually Exclusive Conditions). *Having* $m'_i = <m_l, \kappa_i> \in M'$ *and* $m'_j = <m_n, \kappa_j> \in M'$, *and having* $m_l.c_s = m_n.c_s$, *then* $\kappa_i \cap \kappa_j = \oslash$.

In listing 1.2 we show an example of a translation script that is based on part of the input matching presented in listing 1.1. The translation script is composed by some conditional matches that can be used for translating instances of the source class $\Gamma_1.Professor$. For clarity reasons, we omit the value of Ψ, whose content is [(tenuredSince, isTenuredSince), (lecturesAt, lecturesAt), (nationality, nationality)].

$$m'_1 = \{\{\Gamma_1.Professor, \Gamma_2.Tenured, \Psi\}, [tenuredSince]\}$$
$$m'_2 = \{\{\Gamma_1.Professor, \Gamma_2.Professor, \Psi\}, [lecturesAt]\}$$
$$m'_3 = \{\{\Gamma_1.Professor, \Gamma_2.Employee, \Psi\}, []\}$$

Listing 1.2. Translation script for instances of $\Gamma_1.Professor$

On the data translation phase, the decision on how to translate an instance can be performed using a rather simple approach. The algorithm starts from the first conditional match to the last and chooses the first match whose κ condition is satisfied.

For instance, the κ conditions expressed in the translation script of listing 1.2 can be used to decide whether a professor in Γ_1 should turn into a tenured professor, a regular professor or an employee in Γ_2. If the source instance has a value for the property *tenuredSince*, then this instance is translated into a $\Gamma_2.Tenured$ instance. Otherwise, if the property *lecturesAt* has a value, than this instance is translated into a $\Gamma_2.Professor$ instance. If the previous alternatives fail, the instance is translated into a $\Gamma_2.Employee$ instance, instead.

Note that the matches agree with definitions 4 and 5. If that was not the case, the processing of the translation script could lead to a misplaced translation, given situations where more than one κ condition can be satisfied regarding a given instance. As an example, if the matches position of listing 1.2 were inverted, tenured professors would be translated into a $\Gamma_2.Employee$ instance.

4.1 Translation Script Generation

In this section we present an algorithm that generates conditional matches based on an input matching. Recall that a translation script is actually the list of conditional matches derived from the matches used as input. This algorithm can be used, for instance, to transform the match for the source class $\Gamma_1.Professor$ specified in listing 1.1 into the translation script presented in listing 1.2.

The algorithm responsible for generating the translation script processes each original match at a time ($processMatches()$). To every original match, the

processMatches()
for $m_i \in M$ **do**
  conditionBuilder($m_i.c_s, m_i.c_t, m_i.c_t, m_i.\Psi$)
end for
conditionBuilder(c_s, c_t, c_χ, Ψ)
$c'_\chi \leftarrow sup(c_\chi)$
if $c'_\chi \neq NULL$ **and** $|MatchedProp(c_t)| > 0$ **then**
  $\kappa \leftarrow MatchedProp(c_\chi) - MatchedProp(c'_\chi)$
  if $|\kappa| > 0$ **then**
    $createMatch(c_s, c_t, \Psi, \kappa)$
    $conditionBuilder(c_s, c'_\chi, c'_\chi, \Psi)$
  else
    $conditionBuilder(c_s, c_t, c'_\chi, \Psi)$
  end if
end if
$createMatch(c_s, c_t, \Psi, \emptyset)$

Algorithm 4.1. Translation script generation algorithm

recursive algorithm *conditionBuilder*() is called. As a result of this processing, each original match ($m = \{c_s, c_t, \Psi\}$) generates at least one conditional match in the output. More details about the algorithm *conditionBuilder*() are given below:

The first and the second parameters represent respectively the source (c_s) and the target class (c_t) of the original match. The third parameter initially represents the target class (c_t), but its value is changed throughout the execution of the recursive calls. This parameter is used to help in identifying the properties of a condition. The fourth parameter is the set of property matches of the source and target class(Ψ). This parameter remains constant throughout the execution of the recursive calls. The function $MatchedProp(c)$ returns all properties of the class c that were matched (are part of Ψ). Observe that the generation of the conditional matches are represented by symbolic calls to the *createMatch* function.

Given a target class, two kinds of conditional matches can be generated:

Match with an empty condition. A conditional match with an empty κ condition is generated when the target class has no properties, or when it has the same properties as its root class in the taxonomy.

Match with a non empty condition. If the target class has more properties than its top-most parent, a conditional match is generated, where the κ condition is the difference of the properties between the target class and its closest predecessor that has fewer properties. Afterward, the processing starts over using this predecessor as the target class.

The algorithm finishes when the target class is one of the taxonomy roots. The algorithm assures that at least one conditional match is generated for every original match, which will be the match between the source and target class of the original match itself.

All target classes of the generated conditional matches belong to the same branch of the target class of the original match (they are super-classes of the original target class). The match derivation process is performed by climbing the nodes from the target class of the original match until a root is found. Since the algorithm assumes each target class has at most one direct parent, there is currently no support for taxonomies with multiple inheritance.

The set of property matches of a conditional match is equal to the set of property matches of the original match. If the source class of an original match has unmatched properties, such properties are not included in the derived conditional matches. During translation, these properties are ignored.

When more than one conditional match is generated for the same source class, the matches are sorted according to definition 4, except from one special case, when the input matching contains more than one match for the same source class. Take for instance, the listing below, that shows an input match for the taxonomies of figure 3. For clarity reasons, we omit the property matches, whose content is Ψ_α = [(balance, balance), (interestRate, interestRate)], for m_1, and Ψ_β = [(balance, balance), (borrowingLimit, borrowingLimit)], for m_2. Notice that the input match presents two matches to the source class *Account*.

$$m_1 = \{\Gamma_3.Account, \Gamma_4.SavingsAccount, \Psi_\alpha\}$$
$$m_2 = \{\Gamma_3.Account, \Gamma_4.CurrentAccount, \Psi_\beta\}$$

Listing 1.3. Input matching for the source class $\Gamma_3.Account$

When applying the algorithm to this input matching, we obtain the translation script showed in listing 1.4. Notice that two conditional matches (m'_2 and m'_4) match with the same target class ($\Gamma_4.Account$).

$$m'_1 = \{\{\Gamma_3.Account, \Gamma_4.SavingsAccount, \Psi_\alpha\}, [interestRate]\}$$
$$m'_2 = \{\{\Gamma_3.Account, \Gamma_4.Account, [(balance, balance)]\}, []\}$$
$$m'_3 = \{\{\Gamma_3.Account, \Gamma_4.CurrentAccount, \Psi_\beta\}, [borrowingLimit]\}$$
$$m'_4 = \{\{\Gamma_3.Account, \Gamma_4.Account, [(balance, balance)]\}, []\}$$

Listing 1.4. Translation script for instances of $\Gamma_3.Account$

For this kind of situation, it is necessary to remove redundant matches. In order to preserve the correct match ordering, it suffices to remove all redundant matches whose position within the translation script is higher. In the case of listing 1.4, it would mean to remove the m'_2 conditional match.

An interesting remark about this translation script comes from using it to perform data translation between taxonomies whose instances are inconsistent with respect to each other. For instance, account instances in Γ_3 with all properties defined are not consistent with respect to Γ_4, since in the latter taxonomy the properties *interestRate* and *borrowingLimit* are disjoint. In this case both conditional matches m'_1 and m'_3 are valid. Since there is no deterministic way

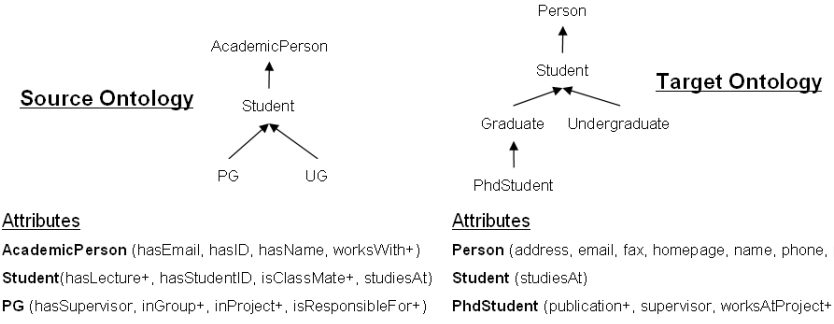


Fig. 4. Ontologies of a research community domain

of sorting these two matches (at least not by our sorting strategy), the instance are always translated into the valid match that is processed first, which is the $\Gamma_4.SavingsAccount$ for the case in question.

5 Case Study

During the explanation of our proposal, we have presented some examples in which conditional matches can be use to improve translation accuracy. In this section, we demonstrate how a translation script behaves when dealing with a translation situation where it is not clear how the instances of a source taxonomy should be translated.

This case study was held using parts of real ontologies we have extracted from the Web. The source ontology(Γ_s) and the target ontology (Γ_t) model a research community, including persons, organizations, and bibliographic metadata. Figure 4 describes the part of the ontologies that were actually used in the experiment. This part represents a taxonomical description of academic people.

Since the ontologies were modelled independently from each other, there is no common agreement on how the classes should match. Given this, we have conducted an exercise with the purpose of establishing a common sense on this subject. In this exercise we asked a group of students from a database research community to identify which classes of the source ontology correspond to which classes of the target ontology. We also asked them to identify the property matches inside each class to class match.

An interesting remark about this exercise is the divergence on the matching of $\Gamma_s.PG$. A significant amount of students (43%) relied more heavily on the semantics of the class names to deduce that $\Gamma_s.PG$ matches $\Gamma_t.Gradute$. The intuition behind the term "graduate" tells that it is a word used to designate students that are beyond the bachelors degree, which is indeed the case of a PG (Post-Graduate) student.

The most significant amount of students (53%) gave more attention to the classes properties to deduce the matches. The students observed that some of the properties of $\Gamma_s.PG(hasSupervisor, inProject)$ are similar to some properties of

$\Gamma_s.PhdStudent(supervisor, worksAtProject)$. This led them to the conclusion that $\Gamma_s.PG$ should match $\Gamma_t.phdStudent$.

One of the conclusions of this exercise was that the instance matching can be a rather subjective task. In the case of $\Gamma_s.PG$ there were two potential ways to translate instances of the source class PG. Given this subjectivity, the translation script approach can be used as an attempt to satisfy both groups.

Before building a translation script for the $\Gamma_s.PG$ class, we need to stipulate an input match for this class. The input match was not computed by a matcher, though. Instead, we took the most frequent answer of the exercise as the correct match. Additionally, we relied on the students opinion to stipulate the following property matches Ψ: [(hasEmail, email), (hasName, name), (studiesAt, studiesAt), (inProject, worksAtProject), (hasSupervisor, supervisor)]. Listing 1.5 shows the resulting input matching.

$$m_1 = \{\Gamma_s.PG, \Gamma_t.PhdStudent, \Psi\}$$

Listing 1.5. Input matching for source class $\Gamma_s.PG$

Given this input match, our translation script generation algorithm produces the following output:

$$m'_1 = \{\{\Gamma_s.PG, \Gamma_t.PhdStudent, \Psi\}, [inProject, hasSupervisor]\}$$
$$m'_2 = \{\{\Gamma_s.PG, \Gamma_t.Graduate, \Psi\}, [studiesAt]\}$$
$$m'_3 = \{\{\Gamma_s.PG, \Gamma_t.Person, \Psi\}, []\}$$

Listing 1.6. Translation script for instances of $\Gamma_s.PG$

Using this translation script, we have three translation possibilities: i) if the properties $\Gamma_s.PG.hasSupervisor$ or $\Gamma_s.PG.inProject$ have a value, the instance is translated into $\Gamma_t.PhDStudent$; ii) if the property $\Gamma_s.PG.studiesAt$ has a value, the instance is translated into $\Gamma_t.Graduate$; iii) the instance is translated into $\Gamma_t.Person$ otherwise.

6 Alternative for the Translation Script

We have devised two alternatives when using conditions for the translation of taxonomical data. So far we have presented the alternative that verifies whether at least one condition property is present in an instance. The second alternative does the opposite, and verifies whether all condition properties are absent. In this case, a match is considered valid only if all its condition properties are absent in a given instance. The listing below shows an example of how the translation script would look like, if the conditions expressed the absence of properties. This translation script regards conditional matches for the source class $\Gamma_1.Professor$, from figure 2.

$m'_1 = \{\{\Gamma_1.Professor, \Gamma_2.Employee, \Psi\}, [lecturesAt, tenuredSince]\}$
$m'_2 = \{\{\Gamma_1.Professor, \Gamma_2.Professor, \Psi\}, [tenuredSince]\}$
$m'_3 = \{\{\Gamma_1.Professor, \Gamma_2.Tenured, \Psi\}, []\}$

Listing 1.7. Translation script for instances of $\Gamma_1.Professor$

Again, during the translation phase, the matches must be analyzed in the correct order to prevent the wrong match from being chosen. As opposed to the approach where a condition tests the properties presence, in this alternative the matches are ordered from the less subsumed target class to the more subsumed target class, as indicated by definition 6.

Definition 6 (Property Absence Sorting). *Having* $m'_i = <m_l, \kappa_i> \in M'$ *and* $m'_j = <m_n, \kappa_j> \in M'$, *and having* $m_l.c_s = m_n.c_s$, *then* $j < i$ *if* $m_l.c_t$ *is subsumed by* $m_n.c_t$.

We believe that testing conditions as property presence is faster than testing the opposite. Our conviction is based on the fact that taxonomy instances are expected to have all its properties (or the majority of them) defined. Hence, testing if all properties of a condition are absent may take longer than testing if at least one property of a condition is present.

7 Conclusions

In this paper we address the problem of translating instances between taxonomies. Our approach encompasses a rule-base translation mechanism, that analyzes the presence of properties in the instances to identify the best translation.

Unlike most translations mechanisms, we are not concerned on designing a complete architecture that supports a broad range of translation needs, such as the specification of the transformation rules that must be applied when translating the instances. Instead, we focus our efforts on a simple rule specification, called translation script, that was conceived as a solution for the translating of taxonomical instances.

Since taxonomical instances are becoming increasingly popular with the advent of ontology models, we claim that the general idea hereby presented could be incorporated into more general/complete translation mechanisms, so the overall translation accuracy could be improved when the schemas involved in the translation describe taxonomies.

As future work, we intend to improve our translation mechanism in order to support indirect matches and taxonomies with multiple inheritance. Additionally, we intend to further explore the heuristic that perform data translation based on the absence of properties, and provide a comprehensive comparison between this heuristic and the one based on the presence of properties.

Acknowledgements. This paper was partially supported by projects FAPERGS PRONEX - 0408933, PETROGRAPHER - 360707 and CAPES.

References

1. Serge Abiteboul, Sophie Cluet, and Tova Milo. Correspondence and translation for heterogeneous data. In *Proceedings of the 6th International Conference on Database Theory*, Delphi, Greece, 1997. Springer, Berlin.
2. Serge Abiteboul, Sophie Cluet, and Tova Milo. Correspondence and translation for heterogeneous data. *Theor. Comput. Sci.*, 275(1-2):179–213, 2002.
3. Michael Boyd, Sasivimol Kittivoravitkul, Charalambos Lazanitis, Peter McBrien, and Nikos Rizopoulos. Automed: A bav data integration system for heterogeneous data sources. In *CAiSE*, pages 82–97, 2004.
4. Chen-Chuan K. Chang and Héctor García-Molina. Conjunctive constraint mapping for data translation. In *Proceedings of the Third ACM International Conference on Digital Libraries*, Pittsburgh, Pa., 1998. ACM Press, New York.
5. Sophie Cluet, Claude Delobel, Jérôme Siméon, and Katarzyna Smaga. Your mediators need data conversion! pages 177–188, 1998.
6. M. Ehrig and Y. Sure. Ontology mapping - an integrated approach.
7. David W. Embley, Li Xu, and Yihong Ding. Automatic direct and indirect schema mapping: experiences and lessons learned. *SIGMOD Rec.*, 33(4):14–19, 2004.
8. Avigdor Gal, Ateret Anaby-Tavor, Alberto Trombetta, and Danilo Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *The VLDB Journal*, 14(1):50–67, 2005.
9. Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. S-match: an algorithm and an implementation of semantic matching. In Y. Kalfoglou, M. Schorlemmer, A. Sheth, S. Staab, and M. Uschold, editors, *Semantic Interoperability and Integration*, number 04391 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany.
10. Sandra Heiler. Semantic interoperability. *ACM Comput. Surv.*, 27(2):271–273, 1995.
11. Y. Kalfoglou and M. Schorlemmer. If-map: An ontology-mapping method based on information-flow theory, 2003.
12. Alexander Maedche, Boris Motik, Nuno Silva, and Raphael Volz. Mafra - a mapping framework for distributed ontologies. In *EKAW '02: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, pages 235–250, London, UK, 2002. Springer-Verlag.
13. M. Magnani, N. Rizopoulos, P.J. McBrien, and D. Montesi. Schema integration based on uncertain semantic mappings. In *ER'05*, LNCS, pages XX–XX. Springer, 2005.
14. Yannis Papakonstantinou, Héctor García-Molina, and Jeffrey Ullman. Medmaker: A mediation system based on declarative specifications. In *Proceedings of the 12th International Conference on Data Engineering*, New Orleans, La., 1996.
15. Sushama Prasad, Yun Peng, and Tim Finin. A Tool For Mapping Between Two Ontologies Using Explicit Information. In *AAMAS 2002 Workshop on Ontologies and Agent Systems*, Bologna, Italy, July 2002.
16. Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, ???? 2001.
17. Maurice van Keulen, Ander de Keijzer, and Wouter Alink. A probabilistic xml approach to data integration. In *ICDE*, pages 459–470, 2005.
18. Gio Wiederhold. Mediators in the architecture of future information systems. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 185–196. Morgan Kaufmann, San Francisco, CA, USA, 1997.