Towards Automatic Generation of Application Ontologies

Eveline R. Sacramento[1,2], Vânia M. P. Vidal[1], José Antonio F. de Macêdo[1], Bernadette F. Lóscio[1], Fernanda Lígia R. Lopes[1], Marco A. Casanova[3]

[1] Department of Computing, Federal University of Ceará - Fortaleza, CE - Brazil
{eveline,vvidal,jose.macedo,bernafarias,fernanda.ligia}@lia.ufc.br
[2] Ceará State Foundation for Meteorology and Water Resources - FUNCEME - Fortaleza, CE - Brazil
eveline@funceme.br
[3] Department of Informatics, PUC-Rio - Rio de Janeiro, RJ - Brazil
casanova@inf.puc-rio.br

Abstract. In the context of the Semantic Web, a domain ontology may be used to provide the necessary support for linking together a large number of heterogeneous data sources pertaining to a same domain. In this paper, such data sources are first described as local ontologies. Then, each local ontology is rewritten as an application ontology, whose vocabulary is restricted to be a subset of the vocabulary of the domain ontology. Application ontologies enable the identification and the association of semantically corresponding concepts, and thereby help information discovery and retrieval, and also data integration. The main contribution of this paper is a strategy to automatically generate application ontologies, considering a set of local ontologies, a domain ontology and the result of the matching between each local ontology and the domain ontology. The proposed strategy also enables the automatic generation of mappings between the domain ontology and the application ontologies, and between each application ontology and its corresponding local ontology, which are used to query the data sources through the domain ontology.

Categories and Subject Descriptors: H. Information Systems [**H.m. Miscellaneous**]: Databases

Keywords: data integration, ontologies, ontology matching, rules, schema mappings, semantic heterogeneity

1. INTRODUCTION

The Web is a complex and vast repository of information, which is often stored in heterogeneous and distributed data sources. Problems that might arise due to heterogeneity of the data are already well known within the database community, and broadly classified as syntactic heterogeneity and semantic heterogeneity.

From a data integration perspective, ontologies provide a possible approach to address the problem of semantic heterogeneity. They have been used to formally describe the semantics of the data sources and to make their content explicit [Wache et al. 2001]. In the literature [Calvanese et al. 2007; Klien 2008; Lutz 2006], two architectures for ontology-based data integration can be identified: *two-level* and *three-level ontology-based architectures*.

The main components of the *two-level architecture* (Figure 1) are: the domain ontology (DO), which contains the basic terms of a domain; the local ontologies (LO), which describe the data sources using an ontology language; and the mapping that specifies the correspondences between the local ontologies and the domain ontology (LO-DO mappings). The systems described in [Calvanese et al. 2007; Klien 2008] adopt this architecture.

The main components of the *three-level architecture* (Figure 2) are: the domain ontology (DO);

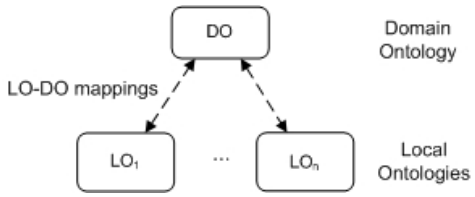

Fig. 1: Two-Level Ontology-based Architecture

the local ontologies (LO); the application ontologies (AO), which rewrite the local ontologies using a subset of the vocabulary of the domain ontology; the mapping that specifies the correspondences between the application ontologies and the domain ontology (mediated mappings); and the mapping that specifies the correspondences between the local ontologies and the application ontologies (LO-AO mappings). In this architecture, the application ontologies are used to facilitate tasks such as the discovery and retrieval of information, and also data integration. Lutz [Lutz 2006] adopts this architecture.

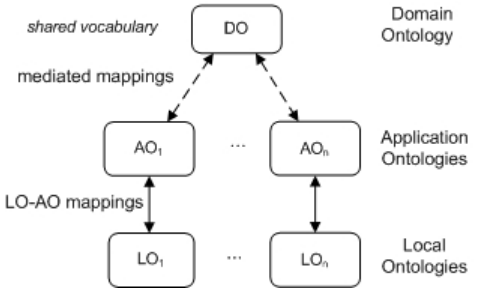

Fig. 2: Three-Level Ontology-based Architecture

Although the design of both two-level and three-level architectures is based on the matching between local ontologies and the domain ontology (explained latter on), the distinguishing point of the three-level architecture is that it introduces application ontologies, which are used to divide the definition of the mappings into two stages: mediated mappings and LO-AO mappings. We use the mediated mappings to define the classes and properties of the domain ontology in terms of the vocabularies of the application ontologies. Such mappings are used for unfolding a query submitted over the domain ontology into subqueries expressed in terms of the application ontologies. We use the LO-AO mappings to define the classes and properties of the application ontology in terms of the vocabulary of the corresponding local ontology. Such mappings are used for unfolding a query submitted over the application ontology into a subquery expressed in terms of its local ontology. In our approach, application ontologies also simplify the definition of the mediated mappings, thereby facilitating the query rewriting process [Vidal et al. 2009].

The major contribution of this paper is a strategy for generating application ontologies and mappings between ontologies at these three levels. The proposed strategy has three steps: (1) *vocabulary matching*, which generates a set of correspondences between entities of each local ontology and the domain ontology; (2) *post-matching*, which validates the correspondences obtained in Step (1) and also helps to adjust some correspondences, which cannot be captured by traditional matching algorithms; and (3) *generation of application ontologies and mappings*, which first induces a set of LO-DO mapping rules from the correspondences obtained in Step (2), and then use these rules to generate the application ontologies and a set of mappings (mediated mappings and LO-AO mappings). Note that we do not propose a new matching algorithm in Step (1), but we propose a *matching model* that captures the results of an existing ontology matching process. So, the matching algorithm is not a contribution of our work. Another contribution is an extended rule-based formalism that is able to

represent objects and include a suitable mechanism for building object identifiers, which is used to specify the mappings of Step (3).

Few current works address the problem of automatically generating application ontologies. Furthermore, the existing ontology matching tools only deal with homogeneous correspondences (correspondences between classes and correspondences between properties). Our approach allows validating or adjusting these correspondences, in order to capture new kinds of correspondences, resulting from the structural heterogeneity (called in our work *path correspondences*). Note that these path correspondences cannot be automatically obtained by existing matching tools, but they can be captured by our post-matching step. The resulting correspondences are then used to generate a set of *operational mappings*. Two important points are related to this problem: (i) how to generate and represent such mappings; and (ii) how to use these mappings to query the data sources. We address both points in our approach.

This paper is organized as follows. Section 2 introduces basic definitions and presents the example used in the rest of the paper. Section 3 introduces the matching model and discusses the post-matching step. Section 4 describes our strategy for generating application ontologies and mappings. Section 5 presents related work. Finally, Section 6 contains the conclusions and future research.

2. BASIC DEFINITIONS

2.1 A Brief Review of Description Logics

We adopt a family of *attributive languages* [Calvanese et al. 1998] defined as follows. A *language* L in the family is characterized by an *alphabet* A, consisting of a set of *atomic concepts* (unary predicates), a set of *atomic roles* (binary predicates), the *universal concept* and the *bottom concept*, denoted by $\top$ and $\bot$, respectively, and a set of *constants*. The universal concept denotes the set of all individuals (the set of all pairs of individuals), while the bottom concept indicates the empty set.

The set of *role descriptions* of L is inductively defined as:

- An atomic role is a role description;
- If S is a role description, then the following is also a role description:
S^- (the inverse of S).

The set of *concept descriptions* of L is inductively defined as:

- An atomic concept and the universal and bottom concepts are concept descriptions;
- If $a_1, ..., a_n$ are constants, then $\{a_1, ..., a_n\}$ is a concept description;
- If C and D are concept descriptions and S is a role description, then the following expressions are concept descriptions:

$\neg C$ (complement)	$\exists S.C$ (full existential quantification)
$C \sqcap D$ (intersection)	$\leq nS$ (at-most restriction)
$C \sqcup D$ (union)	$\geq nS$ (at-least restriction)

In this work, we use the following *terminological axioms* (where e and f are both concept descriptions):

- *inclusion axioms* of the form $e \sqsubseteq f$
- *disjunction axioms* of the form $e \mid f$

2.2 Rule-based Mapping Formalism

Let $\mathfrak{F}$ be a set of function symbols, $\mathfrak{P}$ be a set of atomic concepts and atomic roles, and $\mathfrak{V}$ be a set of variables. A *constant* is a 0-ary function symbol. The set of *terms* over $\mathfrak{F}$ and $\mathfrak{V}$ is recursively defined as follows:
(i) each variable v in $\mathfrak{V}$ is a term;
(ii) each constant c in $\mathfrak{F}$ is a term;
(iii) if $t_1, ..., t_n$ are terms, and f is an n-ary function symbol in $\mathfrak{F}$, then $f(t_1, ..., t_n)$ is a term.

An atom over $\mathfrak{F}$, $\mathfrak{P}$ and $\mathfrak{V}$ is an expression of the form $c(t)$, where c is a atomic concept and t is a term, or of the form $p(t, u)$, where p is an atomic role and t and u are terms.

Let O_S and O_T be two ontologies and R be a rule language. In our work, the relation between a concept of O_S and a concept of O_T (called *concept mapping* and explained latter on) is specified through a set of *mapping rules* of the form

$$\beta_1(w_1) \Leftarrow \alpha_1(v_1), ..., \alpha_m(v_m), \text{ where:}$$

—$\alpha_1(v_1), ..., \alpha_m(v_m)$, called the *body* of the mapping rule, is an atom or an atom conjunction, where $\alpha_i(v_i)$ is an atom whose atomic concept or atomic role occurs in the source ontology O_S;
—$\beta_1(w_1)$, called the *head* of the mapping rule, is an atom whose atomic concept or atomic role occurs in the target ontology O_T.

This rule-based formalism supports *Skolem functions* [Hull and Yoshikawa 1990] for the creation of *new object identifiers* of classes in O_T from one or more properties of O_S. In our work, the Skolem functions are simply used as URIref generators. So, these mapping rules allow the construction of URIrefs for new objects in O_T as terms of the form $f(t_1, ..., t_n)$, where f is an n-ary function symbol and $t_1, ..., t_n$ is a sequence of terms of O_S. Indeed, heterogeneous mappings [Ghidini and Serafini 2006], which use Skolem functions, are necessary to express the semantic relationships between two ontologies, when, for example, the information represented as a class in the former is represented as an object property in the latter, or vice versa.

2.3 Extralite Schemas

In this paper, we consider the family of *extralite ontologies* (or *extralite schemas* [Leme et al. 2009]). Using OWL jargon, an extralite ontology includes the definition of *classes*, *datatype properties* and *object properties*, and admits *domain* and *range* constraints, *minCardinality* and *maxCardinality* constraints, and *subset* and *disjointness* constraints with the usual meaning. Formally, an *extralite ontology* is a pair $\boldsymbol{s} = \boldsymbol{(A, C)}$ such that:

- $\boldsymbol{A}$ is an alphabet, called the *vocabulary* of $\boldsymbol{s}$, whose atomic concepts and atomic roles are called the *classes* and *properties* of $\boldsymbol{s}$, respectively.

- $\boldsymbol{C}$ is a set of formulas, called the *constraints* of $\boldsymbol{s}$, which must be one of the forms
  - *Domain Constraint*: $\exists P \sqsubseteq D$ (property P has domain D)
  - *Range Constraint*: $\exists P^- \sqsubseteq R$ (property P has range R)
  - *minCardinality constraint*: $D \sqsubseteq (\geq kP)$, where D is the domain of P
  (property P maps each individual in its domain D to at least k distinct individuals)
  - *maxCardinality constraint*: $D \sqsubseteq (\leq kP)$, where D is the domain of P
  (property P maps each individual in its domain D to at most k distinct individuals)
  - *Subset Constraint*: $C \sqsubseteq D$ (class C is a subclass of class D)
  - *Disjointness Constraint*: $C \mid D$ (class C is disjoint with class D)

The *minCardinality* and *maxCardinality* constraints are collectively called *cardinality constraints*, and the *subset* and *disjointness* constraints are called *class constraints*. As *property characteristic*, the dialect allows just the *InverseFunctionalProperty*, which captures simple keys. From now on, we will use the terms *class*, *property* and *vocabulary* interchangeably with *atomic concept*, *atomic role* and *alphabet*, respectively.

Finally, for later reference, we define that a class c *dominates* a class d in a schema $\boldsymbol{s}$ [Leme 2009] iff $c = d$, or there is a sequence $(c_1, c_2, ..., c_n)$ such that $c = c_1$, $d = c_n$ and c_{n-1} *subsumes* c_n and, for each $i \in [1, n-2)$, either

- c_{i+1} and c_i are classes and $c_i + 1$ is declared as a subclass of c_i in $\boldsymbol{s}$, or
- c_{i+1} is an object property in $\boldsymbol{s}$ whose domain is c_i, or
- c_i is an object property in $\boldsymbol{s}$ whose range is c_{i+1}.

We also say that $\pi = (c_1, c_2, ..., c_n)$ is a *dominance path* [Leme 2009] from c to d and $\theta = (pk_1, pk_2, ..., pk_n)$, the subsequence of π consisting of the object properties that occur in π, is the *property path* corresponding to π (note that θ may be the empty sequence).

2.4 Example

This section presents an example, adapted from [Casanova et al. 2009], of a virtual store mediating access to online booksellers. We assume that the user provides a domain ontology about virtual stores, and that we have two local ontologies modeling the Amazon and eBay virtual stores (see Figure3).

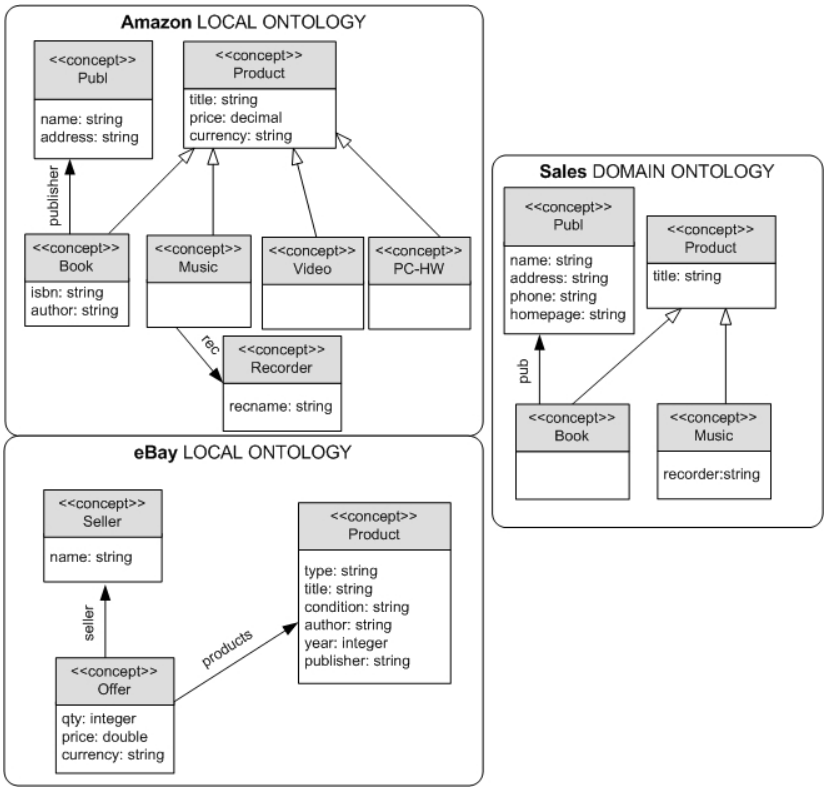

Fig. 3: Domain Ontology and Local Ontologies

We use the namespace prefix *"s:"* to refer to the vocabulary of *Sales* domain ontology. Figure 4 presents some constraints of the domain ontology: the first column shows the domain and range constraints; the second column, the cardinality constraints; and the third one, the class constraints.

We use the namespace prefixes *"a:"* and *"e:"* to refer to the vocabularies of *Amazon* and *eBay* local ontologies, respectively. Figures 5 and 6 formalize their constraints. Moreover, although not indicated, we assume that:

- In Figure 5, all properties, except *a:publisher*, *a:rec* and *a:name* have *maxCardinality* equal to 1, that is, they are single-valued; properties *a:title*, *a:isbn* and *a:name* are *inverse functional*, that is, they constitute simple keys to classes *a:Product*, *a:Book* and *a:Publ*, respectively.
- In Figure 6, all properties have *maxCardinality* equal to 1; and properties *e:name* and *e:title* are *inverse functional* of classes *e:Seller* and *e:Product*, respectively.

$\exists$ s:title $\sqsubseteq$ s:Product	s:Product $\sqsubseteq$ (= 1 s:title)	s:Book $\sqsubseteq$ s:Product
$\exists$ s:title$^-$ $\sqsubseteq$ string	s:Book $\sqsubseteq$ ($\geq$ 1 s:pub)	s:Music $\sqsubseteq$ s:Product
$\exists$ s:pub $\sqsubseteq$ s:Book	s:Music $\sqsubseteq$ (= 1 s:recorder)	
$\exists$ s:pub$^-$ $\sqsubseteq$ s:Publ ...		

Fig. 4: Formal definition of (some of) the constraints of the *Sales* domain ontology

$\exists$ a:title $\sqsubseteq$ a:Product	a:Product $\sqsubseteq$ (= 1 a:title)	a:Book $\sqsubseteq$ a:Product
$\exists$ a:title$^-$ $\sqsubseteq$ string	a:Product $\sqsubseteq$ (= 1 a:price)	a:Music $\sqsubseteq$ a:Product
...	a:Product $\sqsubseteq$ (= 1 a:currency)	a:Video $\sqsubseteq$ a:Product
$\exists$ a:publisher $\sqsubseteq$ a:Book	a:Book $\sqsubseteq$ (= 1 a:isbn)	a:PC-HW $\sqsubseteq$ a:Product
$\exists$ a:publisher$^-$ $\sqsubseteq$ a:Publ...	a:Book $\sqsubseteq$ ($\geq$ 2 a:publisher)	a:Book $\mid$ a:Music
$\exists$ a:rec $\sqsubseteq$ a:Music	a:Music $\sqsubseteq$ ($\geq$ 1 a:rec)	a:Book $\mid$ a:Video
$\exists$ a:rec$^-$ $\sqsubseteq$ a:Recorder	a:Recorder $\sqsubseteq$ (= 1 a:recname)	a:Book $\mid$ a:PC-HW
...	a:Publ $\sqsubseteq$ ($\geq$ 3 a:name)	a:Music $\mid$ a:Video
$\exists$ a:name $\sqsubseteq$ a:Publ	a:Publ $\sqsubseteq$ (= 1 a:address)	a:Music $\mid$ a:PC-HW
$\exists$ a:name$^-$ $\sqsubseteq$ string...		a:Video $\mid$ a:PC-HW

Fig. 5: Formal definition of (some of) the constraints of the *Amazon* local ontology

$\exists$ e:name $\sqsubseteq$ e:Seller	e:Seller $\sqsubseteq$ (= 1 e:name)	
$\exists$ e:name$^-$ $\sqsubseteq$ string...	e:Offer $\sqsubseteq$ (= 1 e:qty) ...	
$\exists$ e:title $\sqsubseteq$ e:Product	e:Product $\sqsubseteq$ (= 1 e:type)	(no class constraints)
$\exists$ e:title$^-$ $\sqsubseteq$ string	e:Product $\sqsubseteq$ (= 1 e:title)	
$\exists$ e:publisher $\sqsubseteq$ e:Product	...	
$\exists$ e:publisher$^-$ $\sqsubseteq$ string...	e:Product $\sqsubseteq$ ($\geq$ 1 e:publisher)	

Fig. 6: Formal definition of (some of) the constraints of the *eBay* local ontology

3. ONTOLOGY MATCHING

3.1 Vocabulary Matching

Let O_S and O_T be two ontologies, and V_S and V_T be their respective vocabularies. Let C_S and C_T be the sets of classes, and P_S and P_T be the sets of datatype or object properties in V_S and V_T, respectively. We extend the notion of a contextualized vocabulary matching previously defined in Leme et al. [Leme et al. 2009]. We consider the definition of restrictions over the domains, that is, we deal with other kinds of context besides classes and subclasses.

A *vocabulary matching from a source ontology O_S to a target ontology O_T* is represented by a finite set S of *sextuples* $(v_1, e_1, r_1, v_2, e_2, r_2)$ such that:

—If $(v_1, v_2) \in C_S \times C_T$, then e_1 and e_2 are the top class $\top$, and r_1 and r_2 are the *restrictions* of classes v_1 and v_2, respectively;
—If $(v_1, v_2) \in P_S \times P_T$, then e_1 and e_2 are classes in C_S and C_T, and r_1 and r_2 are their corresponding *restrictions*, which are allowed only for datatype properties. Note that e_1 and e_2 must be subclasses of the domains, the domains themselves, or *restricted domains* of properties v_1 and v_2, respectively.

In the first case, $(v_1, \top, r_1, v_2, \top, r_2)$ indicates that any individual x of v_1 that satisfies r_1 will be reinterpreted as an individual of v_2 and, furthermore, x will satisfy r_2. In the second case, $(v_1, e_1, r_1, v_2, e_2, r_2)$ indicates that any triple (x, v_1, a) such that x is an individual of e_1 that satisfies r_1, v_1 is a property of e_1, and a is the value of v_1, will be reinterpreted as a triple (x, v_2, a) if x is an individual of e_2 that satisfies r_2, v_2 is a property of e_2, and a is the value of v_2.

Informally, a class restriction r is a *numeric* or a *string* equality *comparison* expression that defines a *domain subset*. Whenever possible, we will use the informal notation "$S = c$", where S is an atomic role and c is a constant, to denote restrictions. Formally, a *restriction* is either the top concept $\top$ or an expression of the form $\exists S.\{c\}$.

If $(v_1, e_1, r_1, v_2, e_2, r_2) \in S$, we say that S *matches* v_1 with v_2 in the *context* of (e_1, r_1) and (e_2, r_2), respectively; (e_i, r_i) is the *context* of v_i; and (v_i, e_i, r_i) is a *contextualized concept*, for $i = 1, 2$. We use the term *context* as in [Köpcke and Rahm 2010].

We also say that a class v_1 in O_S belongs to the same *Semantic Partition(SP)* that a class v_2 in O_T, according to the following rules:

- If $(v_1, \top, r_1, v_2, \top, r_2) \in S$, then v_1 and v_2 belong to the same *SP*.
- If $(v_1, \top, r_1, v_2\text{'}, \top, r_2) \in S$ and v_2' is a subclass of v_2, then v_1 and v_2 belong to the same *SP*.
- If $(v_1, \top, r_1, v_2\text{'}, \top, r_2) \in S$ and $(v_2\text{'} \sqcap r_2 \sqsubseteq v_2 \sqcap r_2)$, then v_1 and v_2 belong to the same *SP*.

Finally, we say that a class v_1 is *semantically related with* a class v_2 represented by $(v_1 \sqsubseteq^{sr} v_2)$, iff v_1 and v_2 belong to the same *SP*. Otherwise, we say that v_1 is *not semantically related with* v_2, represented by $(v_1 \not\sqsubseteq^{sr} v_2)$.

In Table I, line 3 indicates that classes *a:Book* and *s:Book* match (a class correspondence); and line 1 indicates that properties *a:title* and *s:title* match (a property correspondence) in the context of classes *a:Book* and *s:Book*, respectively. It means that property *a:title* of an instance of *a:Book* has the same value as property *s:title* of a corresponding instance of *s:Book*. Note that, in both examples, the columns of class restrictions are the top class $\top$, as *a:Book* and *s:Book* are the domains of the properties *a:title* and *s:title*, respectively. Line 9 indicates that properties *a:recname* and *s:recorder* match, although their respective contexts (*a:Recorder and s:Music*) do not match, i.e., they are not semantically related, as they belong to different semantic partitions. This last example illustrates a case that needs a special matching condition: a path correspondence used to correctly relate the corresponding instances. As we will see in the next section, all of these correspondences can be validated or adjusted in the post-matching step.

In Table II, lines 1 and 2 indicate that properties *e:title* and *s:title* match, and also that the domain *e:Product*, restricted by a string comparison operation (*e:type = 'book'*), matches the domain *s:Book*. These examples illustrate *domain restrictions*.

3.2 Post-Matching

As we said before, even though the focus of this paper is not ontology matching, we are aware that the correspondences obtained using an existing vocabulary matching tool can be often incomplete, and sometimes even incorrect [Cappellari et al. 2010]. Therefore, a user interaction may be needed after the vocabulary matching. This interaction constitutes our post-matching step, which aims to

Table I: Vocabulary matching between the *Amazon* local ontology and the *Sales* domain ontology

#	***Amazon*** *Local Ontology*			***Sales*** *Domain Ontology*		
	v_1	e_1	r_1	v_2	e_2	r_2
1	a:title	a:Book	$\top$	s:title	s:Book	$\top$
2	a:publisher	a:Book	$\top$	s:pub	s:Book	$\top$
3	a:Book	$\top$	$\top$	s:Book	$\top$	$\top$
4	a:title	a:Music	$\top$	s:title	s:Music	$\top$
5	a:Music	$\top$	$\top$	s:Music	$\top$	$\top$
6	a:name	a:Publ	$\top$	s:name	s:Publ	$\top$
7	a:address	a:Publ	$\top$	s:address	s:Publ	$\top$
8	a:Publ	$\top$	$\top$	s:Publ	$\top$	$\top$
9	a:recname	a:Recorder	$\top$	s:recorder	s:Music	$\top$

Table II: Vocabulary matching between the *eBay* local ontology and the *Sales* domain ontology

#	***eBay*** *Local Ontology*			***Sales*** *Domain Ontology*		
	v_1	e_1	r_1	v_2	e_2	r_2
1	e:title	e:Product	e:type=´book´	s:title	s:Book	$\top$
2	e:Product	$\top$	e:type=´book´	s:Book	$\top$	$\top$
3	e:title	e:Product	e:type=´music´	s:title	s:Music	$\top$
4	e:Product	$\top$	e:type=´music´	s:Music	$\top$	$\top$
5	e:publisher	e:Product	e:type=´book´	s:name	s:Publ	$\top$

validate the correspondences obtained in the vocabulary matching step. This validation consists in: (i) to add correspondences that are missing; (ii) to remove correspondences that are wrong; and (iii) to adjust some property correspondences, when their respective contexts do not match. Intuitively, these adjusted correspondences reflect situations that cannot be captured by traditional matching algorithms, when the contexts of the properties are not semantically related, because they belong to different semantic partitions, as we explained before. Our approach allows automatically identifying such property correspondences, and it also allows finding one or more property paths that can be used to turn them into right path correspondences.

A correspondence *from a source ontology O_S to a target ontology O_T* can also be represented by an adjusted sextuple ("$p_1; v_1$", "$c_1; e_1$", r_1, "$p_2; v_2$", "$c_2; e_2$", r_2) resulting from the post-matching step, such that:

—If $(v_1, v_2) \in P_S \times P_T$, then e_1 and e_2 are classes in C_S and C_T, respectively; c_1 is a class in C_S such that there is a dominance path π from c_1 to e_1 in O_S; p_1 is the property path θ corresponding to π; and r_1 and r_2 are the restrictions of classes e_1 and e_2, respectively;

—If $(v_1, v_2) \in P_S \times P_T$, then e_1 and e_2 are classes in C_S and C_T, respectively; c_2 is a class in C_T such that there is a dominance path π from c_2 to e_2 in O_T; p_2 is the property path θ corresponding to π; and r_1 and r_2 are the restrictions of classes e_1 and e_2, respectively;

—If $(v_1, v_2) \in P_S \times P_T$, then e_1 and e_2 are classes in C_S and C_T, respectively; c_1 and c_2 are classes in C_S and C_T, respectively, such that there is a dominance path π from c_1 to e_1, in O_S, and also there is a dominance path π' from c_2 to e_2, in O_T; p_1 is the property path θ corresponding to π and p_2 is the property path θ' corresponding to π'; and r_1 and r_2 are the restrictions of classes e_1 and e_2, respectively.

Note that p_1 or p_2 (but not both) may be empty. In order to illustrate our post-matching step, consider the examples (a) and (b) described below.

(a) Let be the following adjusted sextuple, derived from the sextuple originally presented in line 9 of Table I:

(*"a:rec; a:recname"*, *"a:Music; a:Recorder"* , $\top$, *s:recorder, s:Music,* $\top$)

This means that, although the contexts e_1 = *a:Recorder* and e_2 = *s:Music* are not semantically related, there is a property path p_1 = *a:rec* from the class c_1 = *a:Music*, which is semantically related with e_2 = *s:Music*, to the class e_1 = *a:Recorder*. Such path allows correctly relate the instances of the properties v_1 = *a:recname* (that we can get through p_1) and v_2 = *s:recorder*.

(b) Let be the following adjusted sextuple, derived from the sextuple originally presented in line 5 of Table II:

(*e:publisher, e:Product, e:type='book', "s:pub; s:name", "s:Book; s:Publ",* $\top$)

This means that, although the contexts (e_1, r_1) = (*e:Product,e:type='book'*) and e_2 = *s:Publ* are not semantically related, there is a property path p_2 = *s:pub* from the class c_2 = *s:Book*, which is semantically related with the restricted class (*e:Product* $\sqcap \exists$*e:type.'book'*), to the class e_2 = *s:Publ*. Such path allows correctly relate the instances of the properties v_1 = *e:publisher* and v_2 = *s:name* (that we can get through p_2).

4. GENERATION OF APPLICATION ONTOLOGIES AND MAPPINGS

In general, a *concept mapping* from a *source* ontology O_S into a *target* ontology O_T is a set of expressions that define concepts of O_T in terms of concepts of O_S in such a way that the concepts semantically correspond to each other [Leme et al. 2009]. In this section, we first discuss how to generate a concept mapping from a contextualized vocabulary matching between each local ontology and the domain ontology, using a Datalog variant with OID-invention [Hull and Yoshikawa 1990], and also how to obtain a set of LO-DO mappings. Then we show how to generate the application ontologies and their mappings (mediated mappings and LO-AO mappings).

Note that our process of mapping generation was based in an existing concept mapping process described in Leme et al. [Leme et al. 2009]. However, their work did not allow generating all kinds of mapping rules that we generate, as we have a post-matching step that allows adjusting the set of correspondences automatically obtained.

4.1 Generating the LO-DO Mapping Rules

We show how a set of mapping rules are derived from a contextualized vocabulary matching. Then, we illustrate the definitions with a concrete example.

Let LO be a local ontology and DO be a domain ontology, whose vocabularies are identified by the namespace prefixes *"lo:"* and *"do:"*, respectively. For each sextuple in S, the set M of *LO-DO mapping rules derived from* S contains the following rules (we use the namespace prefix to clarify from which ontology the element belongs to):

Case 1. If $lo : v_1$ and $do : v_2$ are classes and $do : r_2$ is the top class $\top$, then M contains the rules:

$do : v_2(x) \Leftarrow lo : v_1(x), lo : r_1(x)$
$do : s(x) \Leftarrow lo : v_1(x), lo : r_1(x)$, for each superclass s of $do : v_2$

Case 2. If $lo : v_1$ and $do : v_2$ are classes and $do : r_2$ is a restriction of the form $do : p_2 = do : c_2$, where $do : p_2$ is a datatype property and $do : c_2$ is a constant, then M contains the rules:

$do : v_2(x) \Leftarrow lo : v_1(x), lo : r_1(x)$
$do : s(x) \Leftarrow lo : v_1(x), lo : r_1(x)$, for each superclass s of $do : v_2$

$do : p_2(x, do : c_2) \Leftarrow lo : v_1(x), lo : r_1(x)$

Case 3. If $lo : v_1$ and $do : v_2$ both are datatype properties (or both are object properties) such that the classes $lo : e_1$ and $do : e_2$ are semantically related, i.e., $(lo : e_1 \sqsubseteq^{sr} do : e_2)$, then M contains a single rule:

$do : v_2(x, y) \Leftarrow lo : v_1(x, y), lo : e_1(x), lo : r_1(x)$

Case 4. If $lo : v_1$ and $do : v_2$ both are datatype properties (or both are object properties) and there is a property path $\theta = lo : pk_1(x, x_1), lo : pk_2(x_1, x_2), ..., lo : pk_m(x_{m-1}, z)$ in the local ontology LO, then M contains a single rule:

$do : v_2(x, y) \Leftarrow lo : pk_1(x, x_1), lo : pk_2(x_1, x_2), ..., lo : pk_m(x_{m-1}, z), lo : v_1(z, y), lo : e_1(z), lo : r_1(z)$

Case 5. If $lo : v_1$ and $do : v_2$ both are datatype properties (or both are object properties) and there is a property path $\theta = do : pk_1, ..., do : pk_n, do : op_2$ in the domain ontology DO, then M contains the rules:

$do : v_2(f(y), y) \Leftarrow lo : v_1(x, y), lo : e_1(x), lo : r_1(x)$, where $do : v_2$ is a datatype property
$do : e_2(f(y)) \Leftarrow lo : v_1(x, y), lo : e_1(x), lo : r_1(x)$, where $do : e_2$ is a class
$do : op_2(x, f(y)) \Leftarrow lo : v_1(x, y), lo : e_1(x), lo : r_1(x)$, where $do : op_2$ is an object property

Case 6. If $lo : v_1$ and $do : v_2$ both are datatype properties (or both are object properties) and there is a property path θ in the local ontology LO and also a property path θ' in the domain ontology DO (as this case is a combination of Cases 4 and 5), then M contains the rules:

$do : v_2(f(y), y) \Leftarrow lo : pk_1(x, x_1), lo : pk_2(x_1, x_2), ..., lo : pk_m(x_{m-1}, z), lo : v_1(z, y), lo : e_1(z), lo : r_1(z)$, where $do : v_2$ is a datatype property
$do : e_2(f(y)) \Leftarrow lo : pk_1(x, x_1), lo : pk_2(x_1, x_2), ..., lo : pk_m(x_{m-1}, z), lo : v_1(z, y), lo : e_1(z), lo : r_1(z)$, where $do : e_2$ is a class
$do : op_2(x, f(y)) \Leftarrow lo : pk_1(x, x_1), lo : pk_2(x_1, x_2), ..., lo : pk_m(x_{m-1}, z), lo : v_1(z, y), lo : e_1(z), lo : r_1(z)$, where $do : op_2$ is an object property

The strategy for generating the mapping rules is deterministic, follows the order of the matching cases, and always stops, since the number of sextuples in S is finite. Figures 7 and 8 show the LO-DO rules induced by the vocabulary matching of Tables I and II, respectively.

#1:	$s : Book(b) \Leftarrow a : Book(b)$
#2:	$s : Product(b) \Leftarrow a : Book(b)$
#3:	$s : Music(m) \Leftarrow a : Music(m)$
#4:	$s : Product(m) \Leftarrow a : Music(m)$
#5:	$s : Publ(p) \Leftarrow a : Publ(p)$
#6:	$s : title(b, t) \Leftarrow a : title(b, t), a : Book(b)$
#7:	$s : pub(b, p) \Leftarrow a : publisher(b, p), a : Book(b)$
#8:	$s : title(m, t) \Leftarrow a : title(m, t), a : Music(m)$
#9:	$s : name(p, n) \Leftarrow a : name(p, n), a : Publ(p)$
#10:	$s : address(p, a) \Leftarrow a : address(p, a), a : Publ(p)$
#11:	$s : recorder(m, n) \Leftarrow a : rec(m, r), a : recname(r, n), a : Recorder(r)$

Fig. 7: Mapping rules from the *Amazon* local ontology to the *Sales* domain ontology

The rest of this section presents examples illustrating the matching cases.

Example 1 - Case 1. Consider the sextuple in line 2 of Table II

#1:	$s : Book(p) \Leftarrow e : Product(p), e : type(p) = 'book'$
#2:	$s : Product(p) \Leftarrow e : Product(p), e : type(p) = 'book'$
#3:	$s : Music(p) \Leftarrow e : Product(p), e : type(p) = 'music'$
#4:	$s : Product(p) \Leftarrow e : Product(p), e : type(p) = 'music'$
#5:	$s : title(p,t) \Leftarrow e : title(p,t), e : Product(p), e : type(p) = 'book'$
#6:	$s : title(p,t) \Leftarrow e : title(p,t), e : Product(p), e : type(p) = 'music'$
#7:	$s : Publ(fpubl(n)) \Leftarrow e : publisher(b,n), e : Product(b), e : type(b) = 'book'$
#8:	$s : name(fpubl(n),n) \Leftarrow e : publisher(b,n), e : Product(b), e : type(b) = 'book'$
#9:	$s : pub(b, fpubl(n)) \Leftarrow e : publisher(b,n), e : Product(b), e : type(b) = 'book'$

Fig. 8: Mapping rules from the *eBay* local ontology to the *Sales* domain ontology

(*e:Product,* $\top$*, e:type =‘book’, s:Book,* $\top$*,* $\top$)

This sextuple indicates that *e:Product*, restricted by a string comparison operation (*e:type* =´book´), matches *s:Book*. This matching induces the following rule from *e:Product* to *s:Book* (according to the type of the product in *e:type*), shown in line 1 of Figure 8:

$$s : Book(p) \Leftarrow e : Product(p), e : type(p) = 'book'$$

Also, for each superclass of *s:Book*, an additional mapping rule is generated. In this case, as *s:Book* has only one superclass, *s:Product*, we have the following rule, shown in line 2 of Figure 8:

$$s : Product(p) \Leftarrow e : Product(p), e : type(p) = 'book'$$

Example 2 - Case 3. Consider the sextuple in line 1 of Table II:

(*e:title, e:Product, e:type =‘book’, s:title, s:Book,* $\top$)

This sextuple indicates that properties *e:title* and *s:title* match in the restricted context of class *e:Product* and class *s:Book*, respectively (as explained in **Example 1**). In this example, *s:title* belongs to the superclass *s:Product*. This matching induces the following rule from *e:title* to *s:title* (according to the type of the product in *e:type*), shown in line 5 of Figure 8:

$$s : title(p,t) \Leftarrow e : title(p,t), e : Product(p), e : type(p) = 'book'$$

Example 3 - Case 4. Consider the sextuple in line 9 of Table I:

(*a:recname, a:Recorder,* $\top$*, s:recorder, s:Music,* $\top$)

This sextuple indicates that properties *a:recname* and *s:recorder* match, although their respective contexts, *a:Recorder* and *s:Music*, do not match. So, we cannot directly map *a:recname* into *s:recorder*. As we explained before, this sextuple was adjusted in the post-matching step:

(*"a:rec; a:recname", "a:Music; a:Recorder",* $\top$*, s:recorder, s:Music,* $\top$)

The following rule can be derived from this adjusted sextuple, and it is shown in line 11 of Figure 7:

$$s : recorder(m,n) \Leftarrow a : rec(m,r), a : recname(r,n), a : Recorder(r)$$

The body of this rule reflects a path defined from *a:Music*, the class that matches the context *s:Music* of *s:recorder*, to the class *a:Recorder*, the context of *a:recname*. Also, observing the body of the rule, we have that: (1) m stands for an instance of *a:Music*, which the object property *a:rec*

associates with an instance r of *a:Recorder*, and (2) the datatype property *a:recname* in turn associates r with a string n. Now, observing the head of the rule, the datatype property *s:recorder* associates m, an instance of *a:Music*, reinterpreted as an instance of *s:Music* (the domain of *s:recorder*) with n. This reinterpretation is consistent, since line 5 of Table I indicates that *a:Music* matches *s:Music*.

Example 4 - Case 5. Consider the sextuple in line 5 of Table II:

(*e:publisher, e:Product, e:type='book', s:name, s:Publ,* $\top$)

This sextuple indicates that properties *e:publisher* and *s:name* match, although their respective contexts, *e:Product* and *s:Publ*, do not match. So, we cannot directly map *e:publisher* into *s:name*. As we explained before, this sextuple was adjusted in the post-matching step:

(*e:publisher, e:Product, e:type='book', "s:pub; s:name", "s:Book; s:Publ",* $\top$)

The following rule can be derived from this adjusted sextuple, and it is shown in line 8 of Figure 8.

$$s : name(fpubl(n), n) \Leftarrow e : publisher(b, n), e : Product(b), e : type(b) = 'book'$$

Then, two other rules can be automatically deduced from the vocabulary matching and from the Sales domain ontology. These rules are shown in lines 7 and 9 of Figure 8. In these rules, ***fpubl*** is an URIref generator function, and n is the value of the inverse functional property *e:publisher* of the local ontology, passed as argument to f.

$$s : Publ(fpubl(n)) \Leftarrow e : publisher(b, n), e : Product(b), e : type(b) = 'book'$$
$$s : pub(b, fpubl(n)) \Leftarrow e : publisher(b, n), e : Product(b), e : type(b) = 'book'$$

4.2 Generating the Application Ontologies and their Mappings

Referring to Figure 2, let S be a contextualized vocabulary matching between a local ontology LO and a domain ontology DO, and let M be the set of LO-DO mapping rules induced by S. In this section, with the help of our running example, we illustrate how to use M and LO to automatically generate: (1) the application ontology AO corresponding to LO; (2) a set of LO-AO mapping rules; and (3) a set of mediated mapping rules between DO and AO.

Application Ontologies. The AO vocabulary consists of the classes and properties which are, intuitively, just the subset of DO that matches LO, with the additional classes and properties of DO used to define contexts in S. Furthermore, the constraints of AO are the translation of the constraints of LO using the LO-AO mapping rules.

Figure 9 shows the *Amazon* and *eBay* application ontologies generated from their corresponding local ontologies (see Figure 3), adopting the namespace prefixes "*ap:*" and "*ep:*" to refer to their vocabularies, respectively. Figures 10 and 11 show some of the constraints obtained for the application ontologies *Amazon* and *eBay*, respectively. Remember that the first column shows the *domain* and *range* constraints; the second column, the *cardinality* constraints; and the third one, the *class* constraints.

The constraints of the *Amazon* and the *eBay* application ontologies are obtained based on both the LO-AO rules (introduced latter on) and on the constraints of the respective local ontologies, shown in Figures 5 and 6. For example, the LO-AO rules for the *eBay* application ontology (introduced latter on) define *ep:Music* and *ep:Book* as restrictions of *ep:Product*. As a consequence, we have the two subset constraints and the disjointness constraint shown on the third column of Figure 11. Also note that the *eBay* local ontology has neither these classes nor these constraints (see Figure 6).

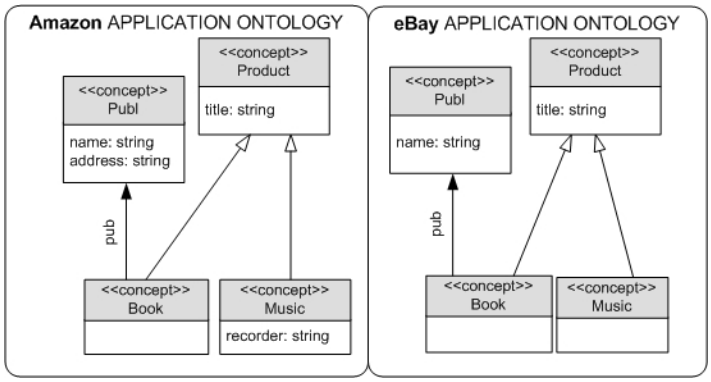

Fig. 9: Application Ontologies

$\exists$ ap:pub $\sqsubseteq$ ap:Book	ap:Product $\sqsubseteq$ (= 1 ap:title)	ap:Book $\sqsubseteq$ ap:Product
$\exists$ s:pub$^-$ $\sqsubseteq$ ap:Publ	ap:Book $\sqsubseteq$ ($\geq$ 2 ap:pub)	ap:Music $\sqsubseteq$ ap:Product
$\exists$ ap:recorder $\sqsubseteq$ ap:Music	ap:Music $\sqsubseteq$ (= 1 ap:recorder)	ap:Book | ap:Music
$\exists$ ap:recorder$^-$ $\sqsubseteq$ string ...	ap:Publ $\sqsubseteq$ ($\geq$ 3 ap:name) ...	

Fig. 10: Formal definition of some constraints of the *Amazon* application ontology

$\exists$ ep:pub $\sqsubseteq$ ep:Book	ep:Product $\sqsubseteq$ (= 1 ep:title)	ep:Book $\sqsubseteq$ ep:Product
$\exists$ ep:pub$^-$ $\sqsubseteq$ ep:Publ	ep:Book $\sqsubseteq$ ($\geq$ 1 ap:pub)	ep:Music $\sqsubseteq$ ep:Product
$\exists$ ep:name $\sqsubseteq$ ep:Publ		ep:Book | ep:Music
$\exists$ ep:name$^-$ $\sqsubseteq$ string ...		

Fig. 11: Formal definition of some constraints of the *eBay* application ontology

LO-AO mapping rules. Since, by construction, the vocabulary of an application ontology is just a subset of the vocabulary of the DO, the LO-AO mapping rules are similar to the LO-DO mapping rules, except for the namespace prefixes that must refer to the AO vocabulary. For example, the LO-AO rules for the *eBay* application ontology include the following two rules, obtained from rules #1 and #3 in Figure 8 by replacing "*s:*", the Sales domain ontology namespace, by "*ep:*", the *eBay* application ontology namespace:

$$ep : Book(p) \Leftarrow e : Product(p), e : type(p) = 'book'$$
$$ep : Music(p) \Leftarrow e : Product(p), e : type(p) = 'music'$$

Mediated mapping rules. First observe that the contextualized vocabulary matching S may have more than one sextuple for the same concept $do : v_2$ of the domain ontology, which implies that the process described in Section 4.1 may generate more than one rule for $do : v_2$. Therefore, as a last step in the construction of the concept mapping M, we collect together all rules for $do : v_2$ as a single rule with a *disjunctive body* (where a *disjunction* is a list of conjunctions separated by semi-colons). For example, suppose that $do : v_2$ is a class and the process generates the following rules:

$$do : v_2(x) \Leftarrow B_i[x], \text{ for } i \in [1, n]$$

Then, we replace all such rules by a single rule of the form:

$$do : v_2(x) \Leftarrow B_1[x]; ...; B_n[x]$$

and likewise, if $do : v_2$ is a property.

Figures 12 and 13 show the rules generated from the rules presented in Figures 7 and 8, respectively, using disjunction in the body of the rules.

```
from #2 and #4:
s : Product(p) ⇐ a : Book(p); a : Music(p)
from #6 and #8:
s : title(p, t) ⇐ (a : title(p, t), a : Book(p)); (a : title(p, t), a : Music(p))
```

Fig. 12: Mapping rules from the *Amazon* local ontology to the *Sales* domain ontology with disjunctive bodies

```
from #2 and #4:
s : Product(p) ⇐ (e : Product(p), e : type(p, 'book')); (e : Product(p), e : type(p, 'music'))
from #5 and #6:
s : title(p, t) ⇐ (e : title(p, t), e : Product(p), e : type(p, 'book')); (e : title(p, t), e : Product(p), e : type(p, 'music'))
```

Fig. 13: Mapping rules from the *eBay* local ontology to the *Sales* domain ontology with disjunctive bodies

The mediated mapping rules then follow directly from the LO-DO rules in disjunctive form and from the LO-AO rules. Figure 14 shows the mediated mapping rules for our example. Finally, we observe that the mediated mapping rules can be used for unfolding a query submitted over the domain ontology into one or more sub-queries over the application ontologies [Vidal et al. 2009].

Class Mapping rules:	Property Mapping rules:
$s : Product \Leftarrow ap : Product;\ ep : Product$	$s : title \Leftarrow ap : title;\ ep : title$
$s : Book \Leftarrow ap : Book;\ ep : Book$	$s : name \Leftarrow ap : name;\ ep : name$
$s : Music \Leftarrow ap : Music;\ ep : Music$	$s : pub \Leftarrow ap : pub;\ ep : pub$
...	...

Fig. 14: Some of the mediated mapping rules

5. RELATED WORK

Comprehensive surveys of ontology matching can be found in [Kalfoglou and Schorlemmer 2003; Euzenat and Shvaiko 2007]. Rahm and Bernstein [Rahm and Bernstein 2001] survey schema matching, and Bernstein and Melnik [Bernstein and Melnik 2007] list the requirements for model management systems that support the matching process. Köpcke and Rahma [Köpcke and Rahm 2010] comparatively analyze eleven frameworks for entity matching. In general, schema matching techniques can be classified as *syntactic*, *semantic* (or *instance-based*) and *hybrid* [Rahm and Bernstein 2001].

Melnik et al. [Melnik et al. 2002] describe syntactic techniques based on modeling the schemas as graphs. Bilke and Naumann [Bilke and Naumann 2005] propose a semantic technique based on an analysis of duplicated instances. Leme et al. [Leme et al. 2009] introduced the notion of a contextualized vocabulary matching between a source ontology and a target ontology using a finite set of quadruples as the specification model; and also proposed a semantic schema matching technique based on similarity functions. Our strategy for generating application ontologies is based on the result of an existing ontology matching process.

Few works address the problem of generating application ontologies. In the geospatial area, recent research [Klien 2008; Lutz 2006] uses ontology-based architectures for enhancing the discovery and retrieval of geographic information. In [Lutz 2006], for example, each feature type schema offered via WFS is described by application concepts that are built from a shared vocabulary. However, in his work, the application ontologies and the mappings are both manually computed by the service providers.

Casanova et al. [Casanova et al. 2009] address the problem of revising the constraints of a mediated schema to accommodate the constraints of a new local schema, after the appropriated translation to a common vocabulary. However, their work just considers homogeneous mappings between ontologies, expressed in DL.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we first proposed a model for specifying ontology vocabulary matching, whose objective is to describe correspondences between the concepts of the local ontologies and concepts of the domain ontology. We then proposed a strategy for the automatic generation of the application ontologies, considering a set of local ontologies, a domain ontology and the result of the matching between each local ontology and the domain ontology. This strategy also enabled the automatic generation of mappings between ontologies at the three levels. These mappings are used for unfolding a query submitted over the domain ontology into one or more sub-queries expressed in terms of the data sources.

We are now implementing the process of generation of the LO-DO mappings, and of generation of the application ontologies and their corresponding mappings. We are also implementing the query rewriting process between a source ontology and a target ontology using the LO-AO mappings.

As a future work, we intend to prove that a vocabulary matching which is structurally correct induces a correct concept mapping, and that a correct concept mapping is also a consistent concept mapping. Our proof will extend that of [Leme 2009] to accommodate our extended definition of contextualized vocabulary matching and our increased set of induced rules.

REFERENCES

Bernstein, P. A. and Melnik, S. Model management 2.0: Manipulating richer mappings. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Beijing, China, pp. 1–12, 2007.

Bilke, A. and Naumann, F. Schema matching using duplicates. In *Proceedings of the International Conference on Data Engineering*. Tokyo, Japan, pp. 69–80, 2005.

Calvanese, D., Giacomo, G. D., andMaurizio Lenzerini, D. L., Poggi, A., and Rosati, R. MASTRO-I: Efficient Integration of Relational Data through DL Ontologies. In *Proceedings of the International Workshop on Description Logic*. Brixen-Bressanone, Italy, pp. 227–234, 2007.

Calvanese, D., Lenzerini, M., and Nardi, D. Description Logics for Conceptual Data Modeling. In J. Chomicki and G. Saake (Eds.), *Logics for Databases and Information Systems*. Kluwer Academic Publishers, pp. 229–263, 1998.

Cappellari, P., Barbosa, D., and Atzeni, P. A framework for automatic schema mapping verification through reasoning. In *Proceedings of the International Workshop on Data Engineering meets the Semantic Web*. Long Beach CA, USA, pp. 245–250, 2010.

Casanova, M. A., Lauschner, T., Leme, L. A., Breitman, K. K., Furtado, A. L., and Vidal, V. M. A strategy to revise the constraints of the mediated schema. In *Proceedings of the International Conference on Conceptual Modeling*. Gramado, Brazil, pp. 265–279, 2009.

Euzenat, J. and Shvaiko, P. *Ontology Matching*. Springer-Verlag New York Inc, 2007.

Ghidini, C. and Serafini, L. Reconciling concepts and relations in heterogeneous ontologies. In *Proceedings of the European Semantic Web Conference*. Budva, Montenegro, pp. 50–64, 2006.

Hull, R. and Yoshikawa, M. Ilog: Declarative creation and manipulation of object identifiers. In *Proceedings of the International Conference on Very Large Databases*. Brisbane, Australia, pp. 455–468, 1990.

Kalfoglou, Y. and Schorlemmer, M. Ontology mapping: the state of the art. *The Knowledge Engineering Review* 18 (1): 1–31, 2003.

Klien, E. *Semantic Annotation of Geographic Information*. Ph.D. thesis, University of Muenster, Germany, 2008.

Köpcke, H. and Rahm, E. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering* 69 (2): 197–210, 2010.

Leme, L. A. P. P. *Conceptual Schema Matching based on Similarity Heuristics*. Ph.D. thesis, PUC - Rio, Brazil, 2009.

Leme, L. A. P. P., Casanova, M. A., Breitman, K. K., and Furtado, A. L. Instance-based OWL schema matching. In *Proceedings of the International Conference on Enterprise Information Systems*. Milan, Italy, pp. 14–26, 2009.

Lutz, M. *Ontology-based discovery and composition of geographic information services*. Ph.D. thesis, Institut für Geoinformatik, Germany, 2006.

Melnik, S., Garcia-Molina, H., and Rahm, E. Similarity flooding: A versatile graph matching algorithm and its application. In *Proceedings of the International Conference on Data Engineering*. San Jose, USA, pp. 117–128, 2002.

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

Vidal, V. M., Sacramento, E. R., Macêdo, J. A., and Casanova, M. A. An ontology-based framework for geographic data integration. In *Proceedings of the International Workshop on Semantic and Conceptual Issues in Geographic Information Systems*. Gramado, Brazil, pp. 337–346, 2009.

Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., and Hübner, S. Ontology-based integration of information - a survey of existing approaches. In *Proceedings of the International Workshop on Ontologies and Information Sharing*. Seattle, USA, pp. 108–117, 2001.