

TOWARDS AUTOMATIC GENERATION OF APPLICATION ONTOLOGIES

Eveline R. Sacramento¹, Vânia M. P. Vidal¹, José A. F. de Macêdo¹, Bernadette F. Lóscio¹, Fernanda L. R. Lopes¹
Marco A. Casanova²

¹Department of Computing, Federal University of Ceará – Fortaleza, CE – Brazil
{eveline, vvidal, jose.macedo, bernafarias, fernanda.ligia}@lia.ufc.br

²Department of Informatics – PUC-Rio – Rio de Janeiro, RJ – Brazil
{casanova}@inf.puc-rio.br

Abstract. In the context of the Semantic Web, domain ontologies can provide the necessary support for linking together a large number of heterogeneous data sources. In our proposal, these data sources are described as local ontologies using an ontology language. 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, so they are useful for enhancing information discovery and retrieval, and also data integration. The main contribution of this work is a strategy to automatically generate such 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 inference of mappings between ontologies, which are used to query the data sources through the domain ontology.

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

Additional Keywords and Phrases: semantic heterogeneity, ontologies, ontology matching, data integration, schema mappings, rules.

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 et al. 2007; 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 et al. 2007] adopt this architecture.

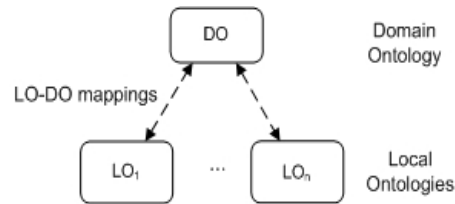


Fig. 1. Two-Level Ontology-based Architecture.

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

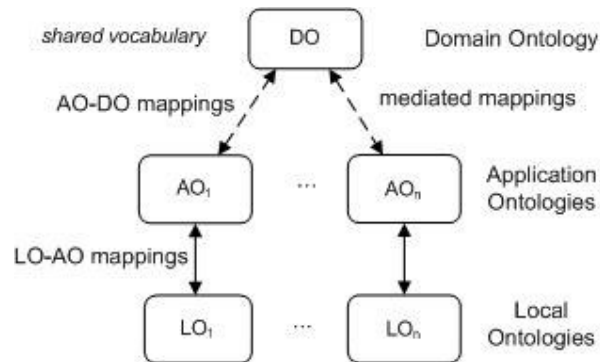


Fig. 2. Three-Level Ontology-based Architecture.

The main problems raised by both architectures are: (i) how to specify the mappings between ontologies; and (ii) how to use the mappings to correctly answer the queries posed on the domain ontology.

In the two-level architecture, the domain ontology is only used for specifying the mediated schema. So, to allow the discovery, retrieval and data integration, the user has to define mappings, which we call *heterogeneous mappings*, between the local ontologies and the domain ontology, as such ontologies do not share the same vocabulary and they are structurally heterogeneous.

In the three-level architecture, the domain ontology is used both for specifying the mediated schema and as a shared vocabulary. As the application ontologies are fragments of the domain ontology, the user can define mappings, which we call *homogeneous mappings*, between these two ontologies.

Although the two-level and three-level architectures are based on the matching between local ontologies and the domain ontology, the three-level architecture introduces application ontologies, which simplify the definition of the mediated mappings, thereby facilitating the query rewriting process [Vidal et al. 2009]. In our approach, application ontologies are used to divide the definition of the mappings into two stages: AO-DO mappings and LO-AO mappings. Furthermore, we use mediated mappings to define the classes and properties of the domain ontology in terms of the vocabularies of the application ontologies. The mediated mappings are used for *unfolding* a query submitted over the domain ontology into sub-queries expressed in terms of the application ontologies. Our approach takes advantage of ontology reasoning services to discard the sub-queries that are not consistent, as detailed in [Vidal et al. 2009].

The major contributions of this paper are two-fold: (1) a model for specifying ontology vocabulary matching; and (2) a strategy for generating application ontologies and their respective mappings. The objective of our matching model is to describe correspondences between local ontologies and the domain ontology. We assume that vocabulary correspondences result from an ontology matching process. The proposed strategy uses these correspondences to automatically generate the application ontologies. Specifically, the AO-DO and the mediated mappings are represented using a Description Logics (DL) formalism to take advantage of ontological reasoning tasks (e.g. subsumption and classification). Reasoning services can be used, for example, to determine whether existing application concepts are a match for a user’s query. Since we need to represent object restructuring mappings, the LO-AO mappings are expressed in an extended rule-based formalism, to overcome DL limitations when defining suitable mechanisms for building object identifiers.

This paper is organized as follows. Section 2 introduces basic definitions and presents the example used in the rest of the paper. Section 3 presents general information about ontology matching since our approach is based on the result of this process. Section 4 describes our strategy for generating application ontologies and mappings. Section 5 presents related work. Finally, Section 6 contains the conclusions.

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 \perp , respectively, the *universal role* and the *bottom role*, also denoted by \top and \perp , respectively, and a set of *constants*.

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

- An atomic role and the universal and bottom roles are role descriptions;
- If *S* and *T* are role descriptions, then the following are role descriptions:

$$S^{-} \text{ (the inverse of } S) \qquad S \circ T \text{ (the composition of } S \text{ and } T)$$

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, \dots, a_n are constants, then $\{a_1, \dots, a_n\}$ 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)	$(\leq n S)$ (at-most restriction)
$C \sqcup D$ (union)	$(\geq n S)$ (at-least restriction)

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

- *inclusions axioms* of the form $e \sqsubseteq f$
- *disjunction axioms* of the form $e \sqcup f$
- *equivalence axioms* of the form $e \equiv f$

2.2 Extralite Schemas

In this paper, the schemas of all ontologies are represented as extralite schemas [Leme et al. 2009] that, using OWL jargon, support *classes*, *datatype properties* and *object properties*, and that admit *domain* and *range* constraints, *minCardinality* and *maxCardinality* constraints, and *subset* and *disjoint* constraints with the usual meaning. Formally, an *extralite schema* is a pair $s = (A, C)$ such that:

- A is an alphabet, called the *vocabulary* of s , whose atomic concepts and atomic roles are called the *classes* and *properties* of s , respectively
- C is a set of formulas, called the *constraints* of s , which must be of one the forms
 - *Domain Constraint*: $\exists P. \top \sqsubseteq D$ (property P has domain D)
 - *Range Constraint*: $\exists P^{-}. \top \sqsubseteq R$ (property P has range R)
 - *minCardinality constraint*: $D \sqsubseteq (\geq k P)$, where D is the domain of P (property P maps each individual in its domain D to at least k distinct individuals)
 - *maxCardinality constraint*: $D \sqsubseteq (\leq k P)$, 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)
 - *Disjoint Constraint*: $C \sqcup D$ (class C is disjoint with class D)

The *minCardinality* and *maxCardinality* constraints are collectively called *cardinality constraints*, and the *subset* and *disjoint* 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.

2.3 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 Fig. 3).

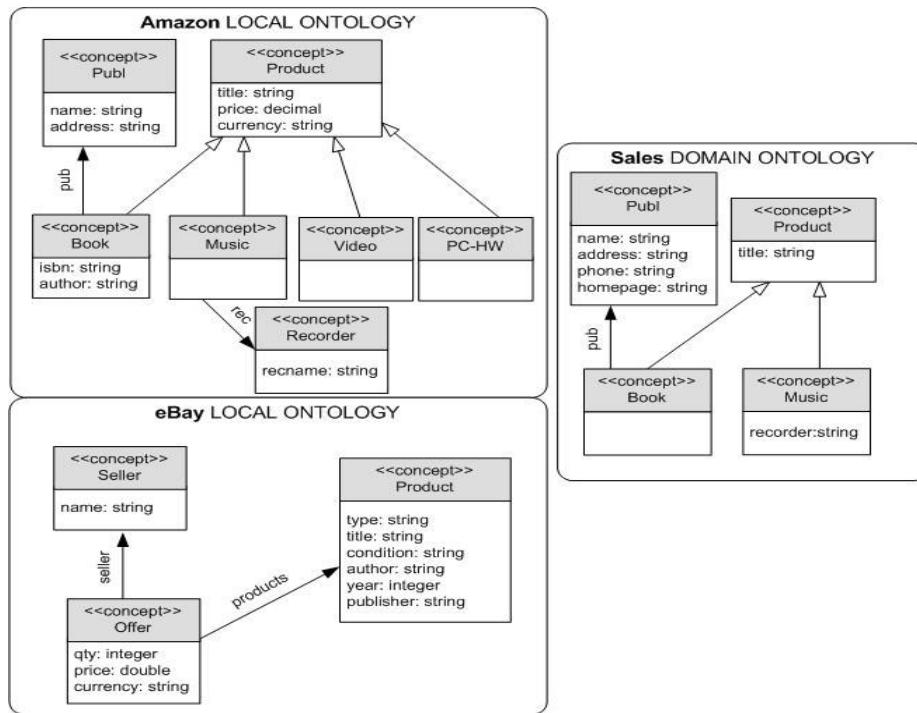


Fig. 3. Domain Ontology and Local Ontologies.

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

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

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

Fig. 4(b). Formal definition of (some of) the constraints of the *Amazon* local ontology.

Figure 4(a) 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 prefix “s:” to refer to the vocabulary of *Sales* domain ontology.

We use the namespace prefixes “*a:*” and “*e:*” to refer to the vocabularies of *Amazon* and *eBay* local ontologies, respectively. Figures 4(b) and 4(c) formalize their constraints. Moreover, although not indicated, we assume that:

- In Figure 4(b), all properties, except *a:pub* 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 4(c), 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.

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

Fig. 4(c). Formal definition of (some of) the constraints of the *eBay* local ontology.

3. VOCABULARY MATCHING

Ontology matching is the process of finding *relationships* or *correspondences* between semantically related entities of different ontologies [Euzenat and Shvaiko 2007; Kalfoglou and Schorlemmer 2003]. Such process is a fundamental issue in many database applications, including query mediation and database integration. Ontology matching tools usually deal with ontologies in the same domain, but with different vocabularies. In this context, the main problem consists in detecting when classes (or properties) from distinct ontologies correspond to the same real-world class (or property) and, therefore, should match.

As mentioned before, our strategy for generating application ontologies [Sacramento et al 2010] is based on the result of an ontology matching process. In the following, we present the two main steps of our strategy, adapted from [Leme et al. 2009]: (1) *vocabulary matching*, which generates the alignment between entities of two different ontologies; and (2) *concept mapping*, which induces a set of mapping rules from the ontology alignment.

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/object properties in V_S and V_T , respectively. We extend the notion of a contextualized vocabulary matching defined in Leme et al. [Leme et al. 2009] and redefine the concept of a *contextualized vocabulary matching* between a *source* ontology O_S and a *target* ontology O_T . In our work, the correspondences obtained in this matching are 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 that must be subclasses of the domains, the domains themselves, or *restricted domains*, of properties v_1 and v_2 , respectively; and r_1 and r_2 are the *restrictions* of classes e_1 and e_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 will be reinterpreted as a triple (x, v_2, a) if x is an individual of e_2 that satisfies r_2 .

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_1, e_1, r_1) is a *contextualized concept*, for $i = 1, 2$.

Intuitively, a vocabulary matching expresses equivalences between properties and classes in a given context. In our work, we consider restrictions over the domains, that is, we deal with other kinds of context besides classes and subclasses. These restricted domains can be captured by a *numeric* or a *string* equality *comparison* operation. These operations permit defining a *domain subset*.

When the domain is restricted by a numeric (string) comparison operation, the user identifies a numeric (string) property for filtering the domain class using the operator “equal to” ($=$). These operations allow the generation of new sextuples in S . Note that the values of the restrictions can be \top , indicating that there are no restrictions defined over the classes.

Let S be a contextualized vocabulary matching between O_S and O_T . The *dependency graph* of S , $DG[S] = (N, E)$, is such that N is the set of classes and properties of O_T that occur in S and $(e_2, f_2) \in E$ iff there is $(v_1, e_1, r_1, v_2, e_2, r_2) \in S$ such that r_2 is a restriction of the form $f_2 = c$. That is, the context of e_2 depends on f_2 . We say that a contextualized vocabulary matching S is *acyclic* iff its dependency graph $DG[S]$ is acyclic.

Furthermore, S is *closed* iff, for every $(v_1, e_1, r_1, v_2, e_2, r_2) \in S$,

- if $e_2 \neq \top$, then there is $(v_3, e_3, r_3, v_4, e_4, r_4) \in S$ such that $e_2 = v_4$, and
- if r_2 is a restriction of the form $f_2 = c$, then there is $(v_5, e_5, r_5, v_6, e_6, r_6) \in S$ such that $f_2 = v_6$

Intuitively, S is closed iff S contains matches for all classes and properties that occur in S .

Finally, we say that S is a *perfect contextualized vocabulary matching* iff S is an acyclic, closed contextualized vocabulary matching.

In Table 1, lines 1 and 3 indicate that properties $a:title$ and $s:title$ match in the context of classes $a:Book$ and $s:Book$, respectively. Note that, in both lines, 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. These lines illustrate “trivial” matching cases. In Table 2, 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 $s:Book$. This example illustrates *domain restrictions*.

Table 1. Vocabulary matching between the *Amazon* local ontology and the *Sales* domain ontology.

#	<i>Amazon Local Ontology</i>			<i>Sales Domain Ontology</i>		
	v_1	e_1	r_1	v_2	e_2	r_2
1	a:title	a:Book	T	s:title	s:Book	T
2	a:pub	a:Book	T	s:pub	s:Book	T
3	a:Book	T	T	s:Book	T	T
4	a:title	a:Music	T	s:title	s:Music	T
5	a:Music	T	T	s:Music	T	T
6	a:name	a:Publ	T	s:name	s:Publ	T
7	a:address	a:Publ	T	s:address	s:Publ	T
8	a:Publ	T	T	s:Publ	T	T
9	a:recname	a:Recorder	T	s:recorder	s:Music	T

Table 2. Vocabulary matching between the *eBay* local ontology and the *Sales* domain ontology.

#	<i>eBay Local Ontology</i>			<i>Sales Domain Ontology</i>		
	v_1	e_1	r_1	v_2	e_2	r_2
1	e:title	e:Product	$e:type='book'$	s:title	s:Book	T
2	e:Product	T	$e:type='book'$	s:Book	T	T
3	e:title	e:Product	$e:type='music'$	s:title	s:Music	T
4	e:Product	T	$e:type='music'$	s:Music	T	T
5	e:publisher	e:Product	$e:type='book'$	s:name	s:Publ	T

4. INDUCING A CONCEPT MAPPING FROM A VOCABULARY MATCHING

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 discuss how to generate a concept mapping from a contextualized vocabulary matching between ontologies.

Currently, many systems still do not consider structural heterogeneities. This way, such systems only need mapping formalisms that are able to represent *homogeneous mappings*. DL, for example, can be used for inferring implicit taxonomic relationships between concepts or between concepts and individuals. However, it presents some limitations, for example, DL cannot express ternary predicate (without resorting to reification). Furthermore, DL does not define suitable mechanisms for the explicit building of object identifiers (OIDs). As both features are important in our approach, we use a Datalog variant with OID-invention [Hull et al 1990] to represent concept mappings.

4.1 Rule-based Mapping Formalism

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

Let O_S and O_T be two ontologies and R be a rule language. A concept mapping is specified through a set of *mapping rules*, each one of the form

$$\beta_1(w_l) \Leftarrow \alpha_1(v_l), \dots, \alpha_m(v_m)$$

where $\alpha_1(v_l), \dots, \alpha_m(v_m)$, called the *body* of the mapping rule, is an atom or an atom conjunction, where an *atom* α_i can be an atomic concept or an atomic role occurring in the source ontology O_S , and v_i is a sequence of terms, and $\beta_1(w_l)$, called the *head* of the mapping, is an atom that can be an atomic concept or an atomic role occurring in the target ontology O_T , and w_l is a sequence of terms.

This rule-based formalism supports *Skolem functions* [Hull et al 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 $f(t_1, \dots, t_n)$, where f is an n -ary function symbol and t_1, \dots, t_n is a sequence of terms of O_S . Other examples of Datalog mapping rules can be found in [Leme et al. 2009]. However, their work does not support extended Datalog rules as we do.

Note that our induced mapping rules can be *homogeneous* or *heterogeneous*. Heterogeneous mappings [Ghidini and Serafini 2006] 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. With respect to *directionality*, a mapping rule can be classified as *unidirectional* or *bidirectional*. An *unidirectional* mapping rule specifies how to map elements in O_T using elements from O_S , and is usually not invertible. A *bidirectional* mapping rule specifies how to map elements O_T using elements from O_S , and vice-versa. We consider only unidirectional mapping rules.

4.2 Generating the Mapping Rules

Let S be a contextualized vocabulary matching between a local ontology LO and a domain ontology DO. In this section, we show how S induces a set of LO-DO mapping rules. Section 4.2.1 describes the strategy to generate the mapping rules according to a classification of the sextuple in S ; the strategy is deterministic, follows the order of the cases, and always stops, since the number of sextuples in S is finite. Figures 5(a) and 5(b) show the LO-DO rules induced by the vocabulary matching of Tables 1 and 2, respectively. Section 4.2.2 shows a detailed derivation of some of these rules.

```

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

```

Fig. 5(a). Mapping rules from the *Amazon* local ontology to the *Sales* domain ontology.

```

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

```

Fig. 5(b). Mapping rules from the *eBay* local ontology to the *Sales* domain ontology.

4.2.1 Matching Cases and the Induced Mapping Rules

We now present each case considered in our work, with their corresponding induced rules. To guarantee that the rules are not mutually recursive and form a complete translation, we assume that S is a perfect contextualized vocabulary matching in what follows. Note that in **Case 1**, $lo:v_1$ and $do:v_2$ are *classes*, while in **Case 2** they are *properties*. In both cases, $(lo:e_1, lo:r_1)$ and $(do:e_2, do:r_2)$ are their corresponding *contexts*, which are explicitly shown in the body of the induced rules. Remember that, in **Case 1**, such contexts are constituted by *class restrictions* or *null values* (as $lo:e_1$ and $do:e_2$ are the top class).

Case 1: $lo:v_1$ and $do:v_2$ are classes.

Case 1.1: Assume that $do:r_2$ is the top class \top .

Then, the LO-DO mapping rules for v_2 are:

$$\begin{aligned}
 do:v_2(x) &\leftarrow lo:v_1(x), lo:r_1(x) \\
 do:s(x) &\leftarrow lo:v_1(x), lo:r_1(x) \quad \text{for each superclass } s \text{ of } do:v_2
 \end{aligned}$$

Case 1.2: Assume that $do:r_2$ is a restriction of the form $do:p_2 = do:c_2$. Then, the LO-DO mapping rules for v_2 would be as follows:

$$\begin{aligned}
 do:v_2(x) &\leftarrow lo:v_1(x), lo:r_1(x) \\
 do:s(x) &\leftarrow lo:v_1(x), lo:r_1(x) \quad \text{for each superclass } s \text{ of } do:v_2 \\
 do:p_2(x, do:c_2) &\leftarrow lo:v_1(x), lo:r_1(x)
 \end{aligned}$$

The first rule indicates that each individual x in $lo:v_1$ that satisfies $lo:r_1$ should be reinterpreted as an individual of $do:v_2$. The second rule says that this reinterpretation propagates up to the superclasses of

$do:v_2$. The third rule tells us that each individual in $lo:v_1$ that satisfies $lo:r_1$ will have the value of property $do:p_2$ equal to $do:c_2$.

Case 2: $lo:v_1$ and $do:v_2$ are properties.

Case 2.1: The contexts of the properties match. Then, the LO-DO mapping rule for v_2 is:

$$do:v_2(x, y) \Leftarrow lo:v_1(x, y), lo:e_1(x), lo:r_1(x), do:e_2(x), do:r_2(x)$$

Note that $do:e_2$ and $do:r_2$ will be further processed until the body of the rule contains only classes and properties of the local ontology. However, since S is a perfect contextualized vocabulary matching, this process is always possible. (A similar observation also applies to the other cases).

Case 2.2: The contexts of the properties do not match.

Case 2.2.1: The user proposes a *restructuring of objects* between the enrolled ontologies that allows the properties' alignment. This restructuring creates in DO instances of a class ($do:e_2$), an object property ($do:p_2$), and a datatype property ($do:v_2$): all of them from a property of LO. The LO-DO mapping rule for v_2 would be as follows (this rule can be directly derived from the vocabulary matching):

$$do:v_2(f(y), y) \Leftarrow lo:v_1(x, y), lo:e_1(x), lo:r_1(x), do:e_2(x), do:r_2(x)$$

Then, these rules are generated from the schemas and the vocabulary matching:

$$do:e_2(f(y)) \Leftarrow lo:v_1(x, y), lo:e_1(x), lo:r_1(x), do:e_2(x), do:r_2(x)$$

$$do:p_2(x, f(y)) \Leftarrow lo:v_1(x, y), lo:e_1(x), lo:r_1(x), do:e_2(x), do:r_2(x)$$

Case 2.2.2: The user identifies a *property path* in the local ontology that allows the properties' alignment. Then, the LO-DO mapping rule for v_2 is:

$$do:v_2(x, y) \Leftarrow lo:pk_1(x, x_1), lo:pk_2(x_1, x_2), \dots, lo:pk_m(x_{m-1}, z), lo:v_1(z, y), lo:e_1(z), lo:r_1(z), do:e_2(x), do:r_2(x)$$

Case 2.2.3: The user proposes a *restructuring of objects* between the enrolled ontologies and identifies a *property path* in the local ontology that allow the properties' alignment. This case is a combination of **Cases 2.2.1** and **2.2.2**. The LO-DO mapping rule for v_2 would be as follows (this rule can be directly derived from the vocabulary matching and the property path):

$$do:v_2(f(y), y) \Leftarrow lo:pk_1(x, x_1), lo:pk_2(x_1, x_2), \dots, lo:pk_m(x_{m-1}, z), lo:v_1(z, y), lo:e_1(z), lo:r_1(z), do:e_2(x), do:r_2(x)$$

Then, these rules are generated from the schemas, the vocabulary matching and the property path:

$$do:e_2(f(y)) \Leftarrow lo:pk_1(x, x_1), lo:pk_2(x_1, x_2), \dots, lo:pk_m(x_{m-1}, z), lo:v_1(z, y), lo:e_1(z), lo:r_1(z), do:e_2(x), do:r_2(x)$$

$$do:p_2(x, f(y)) \Leftarrow lo:pk_1(x, x_1), lo:pk_2(x_1, x_2), \dots, lo:pk_m(x_{m-1}, z), lo:v_1(z, y), lo:e_1(z), lo:r_1(z), do:e_2(x), do:r_2(x)$$

4.2.2 Illustrating the Matching Cases

In this section, we present examples illustrating the matching cases of Section 4.2.1..

Example 1 - Case 1:

Consider the sextuple in line 2 of Table 2:

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

This sextuple indicates that $e:Product$, restricted by a string comparison operation ($e:type = \text{'book'}$), 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 5(b):

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

Then, 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 5(b):

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

Example 2 - Case 2.1:

Consider the sextuple presented in line 1 of Table 2:

$$(e:title, e:Product, e:type = \text{'book'}, s:title, s:Book, \top).$$

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 5(b):

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

Example 3 - Case 2.2.1:

Consider the sextuple in line 9 of Table 1:

$$(a:rename, a:Recorder, \top, s:recorder, s:Music, \top).$$

This sextuple indicates that properties $a:rename$ and $s:recorder$ match, although their respective contexts, $a:Recorder$ and $s:Music$, do not match. So, we cannot directly map $a:rename$ into $s:recorder$. In this case, only part of the following rule can be directly derived from the vocabulary matching. This rule is shown in line 11 of Figure 5(a):

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

The body of rule reflects a *path* that is generated from the schemas and from the vocabulary matching. This path is defined from $a:Music$, the class that matches the context $s:Music$ of $s:recorder$, to the class $a:Recorder$, the context of $a:rename$. Also, observing the body, we have that m stands for an instance of $a:Music$, which the object property $a:rec$ associates with an instance r of $a:Recorder$, and the datatype property $a:rename$ 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 1 indicates that $a:Music$ matches $s:Music$.

Example 4 - Case 2.2.2:

Consider the sextuple in line 5 of Table 2:

$$(e:publisher, e:Product, e:type = \text{'book'}, s:name, s:Publ, \top).$$

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 $s:name$. This matching induces the following rule, shown in lines 8 of Figure 5(b).

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

Then, two other rules can be automatically deduced from the domain ontology DO. These rules are shown in lines 7 and 9 of Figure 5(b). In these rules, *fpubl* is an URIref generator function, created by the user to express the restructuring of objects, and *n* is an inverse functional property of the local ontology passed as argument to *f*.

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

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

4.3 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 \mathcal{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 \mathcal{M}' and a local ontology LO to automatically generate the corresponding application ontology AO, which intuitively is just the subset of DO that matches LO, with the additional classes and properties of the DO used to define contexts in *S*. We also show to generate a set of LO-AO mapping rules and a set of AO-DO mappings.

Application Ontologies. For each rule in \mathcal{M}' , a class or property is created in AO from the rule head. Also, the constraints of the local ontology LO are translated to the constraints of the corresponding application ontology in view of the LO-DO mapping rules and of the constraints of the domain ontology.

Figure 6 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 7(a) and 7(b) 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.

Note that, the constraints of the *eBay* application ontology are obtained based on both the LO-DO rules of Figure 5(b) and on the constraints of the *Sales* domain ontology of Figure 4(a). Note in Figure 7(b) that *ep:Music* and *ep:Book* are defined as restrictions of *ep:Product*. As a consequence, we have two subset constraints, shown on the third column of this figure. Also note that the *eBay* local ontology does not have these classes neither these constraints (see Figure 4(c)). So, the constraints of the *Sales* domain ontology were used to model the constraints of these new classes in the *eBay* application ontology.

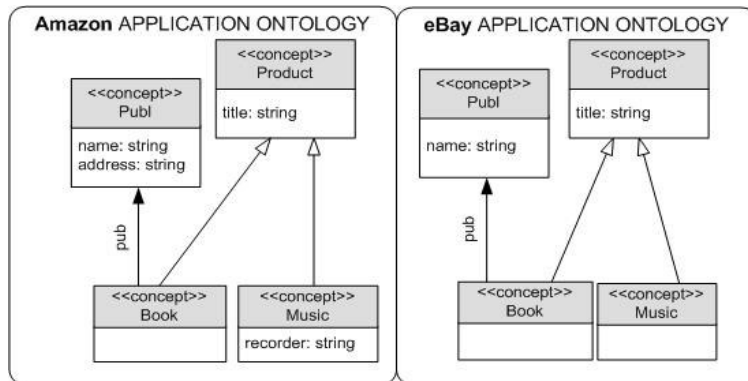


Fig. 6. Application Ontologies.

$\exists \text{ ap:pub}.T \sqsubseteq \text{ ap:Book}$ $\exists \text{ ap:pub}^{\neg}.T \sqsubseteq \text{ ap:Publ}$ $\exists \text{ ap:recorder}.T \sqsubseteq \text{ ap:Music}$ $\exists \text{ ap:recorder}^{\neg}.T \sqsubseteq \text{ string}$...	$\text{ ap:Product} \sqsubseteq (=1 \text{ ap:title})$ $\text{ ap:Book} \sqsubseteq (\geq 2 \text{ ap:pub})$ $\text{ ap:Music} \sqsubseteq (=1 \text{ ap:recorder})$ $\text{ ap:Publ} \sqsubseteq (\geq 3 \text{ ap:name})$...	$\text{ ap:Book} \sqsubseteq \text{ ap:Product}$ $\text{ ap:Music} \sqsubseteq \text{ ap:Product}$ $\text{ ap:Book} \mid \text{ ap:Music}$
---	---	--

Fig. 7(a). Formal definition of some constraints of the *Amazon* application ontology.

$\exists \text{ ep:pub}.T \sqsubseteq \text{ ep:Book}$ $\exists \text{ ep:pub}^{\neg}.T \sqsubseteq \text{ ep:Publ}$ $\exists \text{ ep:name}.T \sqsubseteq \text{ ep:Publ}$ $\exists \text{ ep:name}^{\neg}.T \sqsubseteq \text{ ep:string}$...	$\text{ ep:Product} \sqsubseteq (=1 \text{ ep:title})$ $\text{ ep:Book} \sqsubseteq (\geq 1 \text{ ep:pub})$	$\text{ ep:Book} \sqsubseteq \text{ ep:Product}$ $\text{ ep:Music} \sqsubseteq \text{ ep:Product}$ $\text{ ep:Book} \mid \text{ ep:Music}$
---	---	--

Fig. 7(b). Formal definition of some constraints of the *eBay* application ontology.

LO-AO mapping rules. Since the application ontology is just a subset of 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.

AO-DO mappings. These mappings specify the correspondences between the application ontologies AOs and the domain ontology DO. Figures 8(a) and 8(b) show some of the inferred AO-DO mappings in DL.

Mediated Mappings. They allow the definition of a class (property) of a domain ontology through a *unique axiom*, composed of unions of classes (properties) of one or more application ontologies. As such mappings express homogeneous mappings between ontologies; they 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]. Figure 9 shows the mediated mappings for our example in DL.

$\text{ ap:Product} \sqsubseteq \text{ s:Product}$ $\text{ ap:title} \sqsubseteq \text{ s:title}$ $\text{ ap:Book} \sqsubseteq \text{ s:Book} \dots$	$\text{ ep:Product} \sqsubseteq \text{ s:Product}$ $\text{ ep:title} \sqsubseteq \text{ s:title}$ $\text{ ep:Book} \sqsubseteq \text{ s:Book} \dots$
--	--

Fig. 8(a). Some of the inferred mappings between Amazon AO and Sales DO.

Fig. 8(b). Some of the inferred mappings between eBay AO and Sales DO.

Class Mappings: $\text{ s:Product} \equiv \text{ ap:Product} \sqcup \text{ ep:Product}$ $\text{ s:Book} \equiv \text{ ap:Book} \sqcup \text{ ep:Book} \dots$	Property Mappings: $\text{ s:title} \equiv \text{ ap:title} \sqcup \text{ ep:title}$ $\text{ s:name} \equiv \text{ ap:name} \sqcup \text{ ep:name} \dots$
---	--

Fig. 9. Some of the mediated mappings.

5. RELATED WORK

Research on the ontology matching was already addressed by many relevant works. Euzenat and Shvaiko [Euzenat and Shvaiko 2007] present a comprehensive survey of ontology matching. 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 Rahma 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; Shvaiko and Euzenat 2004]. For example, 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.

In relation to the generation of application ontologies, few works have addressed this problem so far. In the geospatial area, recent researches [Klien et al. 2007; Lutz 2006] use 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 this work, the application ontologies and the mappings are manually computed by the service providers.

Casanova et al. [Casanova et al. 2009] addresses 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. Furthermore, although we translate some constraints as they do, our approach also allows using the constraints of the domain ontology to model some constraints of the application ontologies.

6. CONCLUSIONS

In this paper, we first proposed a model for specifying ontology vocabulary matching, whose objective is to describe correspondences between local ontologies and 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 inference of mappings between ontologies, which are used to query the data sources through the domain ontology.

In 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 the proof presented in [Leme 2009] according to our extended definition of contextualized vocabulary matching and our increased set of induced rules.

REFERENCES

- Bilke, A.; Naumann, F. 2005. "Schema matching using duplicates". In: Proc. 21st Int'l. Conf. on Data Engineering (Apr 2005), pp. 69–80.
- Bernstein, P.; Melnik, S. 2007. "Model management 2.0: manipulating richer mappings". In: Proc. 27th ACM SIGMOD Int'l. Conf. Management of Data, Beijing, China, pp. 1–12.
- Calvanese, D., De Giacomo, G., Lenzerini, M., Lembo, D., Poggi, A., Rosati, R. 2007. "MASTRO-I: Efficient Integration of Relational Data through DL Ontologies". In: Proc. Description Logic Workshop (DL'07), pp. 227–234.

- Calvanese, D., Lenzerini, M., Nardi, D. 1998. "Description Logics for Conceptual Data Modeling". In: *Logics for Databases and Information Systems*. Kluwer Academic Publisher.
- Casanova, M.A., Lauschner, T., Leme, L.A.P., Breitman, K.K; Furtado, A.L., Vidal, V. M. P. 2009. "A Strategy to Revise the Constraints of the Mediated Schema". In: Proc. 28th Conf. on Conceptual Modeling, pp. 265-279, Gramado, Brazil.
- Euzenat, J., Shvaiko, P. 2007. "*Ontology Matching*". Springer, Heidelberg.
- Ghidini C., Serafini L. 2006. "Reconciling Concepts and Relations in Heterogeneous Ontologies". In: Proc. ESWC 2006, pp. 50-64.
- Hull, R., Yoshikawa, M. 1990. "ILOG: Declarative Creation and Manipulation of Object Identifiers". In: Proc. VLDB 1990, pp. 455-468.
- Kalfoglou, Y., Schorlemmer, M. 2003. "Ontology Mapping: the State of the Art". In: *The Knowledge Engineering Review*, vol. 18, n. 1, pp.1-31. Cambridge University Press.
- Klien, E., Fitzner, D. I., Maué, P. 2007. "Baseline for Registering and Annotating Geodata in a Semantic Web Service Framework". In: Proc. 10th Conf. on Geographic Information Science, Aalborg, Denmark.
- Köpcke, H.; Rahma, E. 2010. "Frameworks for entity matching: A comparison", *Data & Knowledge Engineering*, Volume 69, Issue 2, February 2010, pp. 197-210.
- Leme, L. A. P. 2009. "*Conceptual Schema Matching based on Similarity Heuristics*". Phd Thesis, PUC-Rio, Brazil.
- Leme, L. A. P., Casanova, M. A., Breitman, K. K., Furtado, A. L. 2009. "Instance-based OWL Schema Matching". In: Proc. 11th Int'l. Conf. Enterprise Info. Sys., Milan, Italy.
- Lutz, M. 2006. "Ontology-based Discovery and Composition of Geographic Information Services". Phd Thesis, Institut für Geoinformatik.
- Melnik, S.; Garcia-Molina, H.; Rahm, E. 2002. "Similarity flooding: a versatile graph matching algorithm and its application to schema matching". In: Proc. 18th Int'l. Conf. on Data Engineering, pp. 117-128.
- Rahm, E., Bernstein, P. A. 2001. "A Survey of Approaches to Automatic Schema Matching", In: *VLDB Journal* 10(4), pp. 334-350.
- Sacramento, E. R.; Vidal, V. M. P.; Macêdo, J. A.; Lóscio, B. F.; Lopes, F. L. R.; Casanova, M. A.; Lemos, F. 2010. "Towards Automatic Generation of Application Ontologies". To be published in: Proc. 12th International Conference on Enterprise Information Systems - ICEIS, 2010, Funchal, Ilha da Madeira. 12th Int. Conf. on Enterprise Information Systems.
- Shvaiko, P., Euzenat, J. 2004. "A Survey of Schema-based Matching Approaches". Technical Report DIT-04-087, Informatica e Telecomunicazioni, Univ. Trento.
- Vidal, V.M.P., Sacramento E. R., José A. F., Casanova M. A. 2009. "An Ontology-Based Framework for Geographic Data Integration". In: Proc. 3rd International Workshop on Semantic and Conceptual Issues in GIS (SeCoGIS 2009), in conjunction with the 28th International Conference on Conceptual Modeling (ER 2009). Berlin / Heidelberg: Springer, 2009.
- Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H. and Hübner, S. 2001. "Ontology-based Integration of Information - A Survey of Existing Approaches". In: Proc. IJCAI-01 Workshop: Ontologies and Information Sharing, pp. 108-117.