J Braz Comput Soc (2010) 16: 21–34
DOI 10.1007/s13173-010-0005-3

INVITED PAPER

OWL schema matching

**Luiz André P. Paes Leme · Marco A. Casanova ·
Karin K. Breitman · Antonio L. Furtado**

Received: 1 March 2010 / Published online: 20 April 2010

Abstract Schema matching is a fundamental issue to many database applications, such as query mediation and data warehousing. It becomes a challenge when different vocabularies are used to refer to the same real-world concepts. In this context, a convenient approach, sometimes called extensional, instance-based, or semantic, is to detect how the same real world objects are represented in different databases and to use the information thus obtained to match the schemas. Additionally, we argue that automatic approaches of schema matching should store provenance data about matchings. This paper describes an instance-based schema matching technique for an OWL dialect and proposes a data model for storing provenance data. The matching technique is based on similarity functions and is backed up by experimental results with real data downloaded from data sources found on the Web.

Keywords Schema matching · OWL · Similarity · Provenance

L.A.P.P. Leme (✉) · M.A. Casanova · K.K. Breitman ·
A.L. Furtado
Department of Informatics, Pontifical Catholic University of Rio de Janeiro, Rua Marquês de S. Vicente, 225, Rio de Janeiro, RJ, CEP 22451-900, Brazil
e-mail: lleme@inf.puc-rio.br

M.A. Casanova
e-mail: casanova@inf.puc-rio.br

K.K. Breitman
e-mail: karin@inf.puc-rio.br

A.L. Furtado
e-mail: furtado@inf.puc-rio.br

1 Introduction

A *database conceptual schema*, or simply a *schema*, is a high level description of how database concepts are organized. A *schema matching* from a source schema S into a target schema T defines concepts in T in terms of the concepts in S.

The problem of finding schema matchings becomes a challenge when different vocabularies are used to refer to the same real-world concepts [7]. In this case, a convenient approach, sometimes called extensional, instance-based or semantic, is to detect how the same real-world objects are represented in different databases and to use the information thus obtained to match the schemas. This approach is grounded on the interpretation, traditionally accepted, that "terms have the same extension when true of the same things" [24].

Moreover, in many applications, the schema matchings alone are not sufficient; it is also required to unveil the evidences, and to reveal the methods used to get to the final alignments. We refer to such data as the provenance data about matchings.

In this paper, we address the problem of matching two schemas that belong to an expressive OWL dialect. We adopt an instance-based approach, assuming that a set of instances from each schema is available.

The major contributions of this paper are four-fold. First, we decompose the problem of OWL schema matching into the problem of *vocabulary matching* and the problem of *concept mapping*. We also introduce sufficient conditions guaranteeing that a vocabulary matching induces correct concept mappings. Second, we describe an OWL schema matching technique based on the notion of similarity. Third, we evaluate the precision of the proposed technique using data available on the Web. Finally, we propose a data model to store provenance data.

Several papers address schema matching. Rahm and Bernstein [25] is an early survey of schema matching techniques. Euzenat and Shvaiko [13] survey ontology matching techniques. Castano et al. [8] describe the H-Match algorithm to dynamically match ontologies.

Bilke and Naumann [2] describe an instance-based technique that explores similarity algorithms. Brauner et al. [4] adopt the same idea to match two thesauri. Wang et al. [27] describe a technique to match Web databases, which uses a set of typical instances. Brauner et al. [6] apply this idea to match geographical database. Brauner et al. [5] describe a matching algorithm based on measuring the similarity between attribute domains.

Unlike any of the above techniques, the schema matching process we propose uses similarity functions to induce vocabulary matchings in a non-trivial way, using an expressive OWL dialect. Through a set of examples, we also illustrate that the structure of OWL schemas may lead to incorrect concept mappings, and indicate how to avoid such pitfalls.

This paper is organized as follows. Section 2 introduces the OWL dialect adopted and the notions of vocabulary matching and concept mapping. Section 3 describes our technique to obtain OWL vocabulary matchings and contains experimental results. Section 4 describes how to induce concept mappings from vocabulary matchings. Section 5 presents a provenance data model for schema matchings. Finally, Sect. 6 lists the conclusions and directions for future work.

2 OWL schema matching

2.1 OWL extralite

We assume that the reader is familiar with basic XML concepts. In particular, recall that a *resource* is anything identified by an URIref and that an XML *namespace* or a *vocabulary* is a set of URIrefs. A *literal* is a character string that represents an XML Schema datatype value. We refer the reader to [1] for more details.

An *RDF statement* (or simply a *statement*) is a triple `(s,p,o)`, where `s` is a URIref, called the *subject* of the statement, `p` is a URIref, called the *property* of the statement, and `o` is either a URIref or a literal, called the *object* of the statement; if `o` is a literal, then `o` is also called the *value* of property the `p`.

The *Web Ontology Language* (OWL) describes classes and properties in a way that facilitates machine interpretation of Web content. The description of OWL is organized as three dialects: OWL Lite, OWL DL, and OWL Full.

We will define and work an OWL dialect, that we call *OWL Extralite*. It supports:

- named classes
- datatype and object properties
- subclasses
- individuals
- *domain* and range of datatype and object properties
  - the domain is always a class
  - the *range* of a datatype property is an XML schema type, whereas the *range* of an object property is a class
- *minCardinality* and *maxCardinality*, with the usual meaning
- *inverseFunctionalProperty*, which captures simple keys (we note that only OWL Full supports the *inverseFunctionalProperty* for datatype properties)

Note that, OWL Extralite thus defined has the same expressiveness as UML [22]. In the context of this paper, we use OWL Extralite to express database schemas since it is a convenient technology to exchange data in the web, as well as to manipulate database schemas.

An *OWL schema* (more often called an OWL *ontology*) is a collection of RDF triples that uses the OWL vocabulary. A *concept* in an OWL schema is a class, datatype property or object property defined in the schema. The *vocabulary* of the schema is the set of concepts defined in the schema (a set of URIrefs). It is important to note that, unlike UML, the scope of a property name is global to the OWL Extralite schema.

A triple of the form `(s,rdf:type,c)` indicates that `s` is an instance of a class `c`; a triple of the form `(s,p,v)` indicates that `s` has a datatype property `p` with value `v`; and a triple of the form `(s,p,o)` indicates that `s` and `o` are related by an object property `p`.

In the rest of the paper, we refer to OWL Extralite schemas simply as *schema*. Figures 1 and 2 show schemas for fragments of the Amazon and the eBay databases, using a shorthand notation to save space and improve readability. Consistently with XML usage, from this point on, we will use the namespace prefixes `am:` and `eb:` to refer to the vocabularies of the Amazon and the eBay schemas respectively, and qualified names of the form `V:T` to indicate that `T` is a term of the vocabulary `V`.

In Fig. 1, for example, `am:title` is defined as a datatype property with domain `am:Product` and range `string` (an XML Schema data type), `am:Book` is declared as a subclass of `am:Product`, and `am:publisher` is defined as an object property with domain `am:Book` and range `am:Publ`. Note that the scope of `am:title` and `am:publisher` is the schema, and not the classes defined as their domains.

Furthermore, although not indicated in Fig. 1, we assume that all properties, except `am:author`, have *maxCardinality* equal to 1, and that `am:isbn` is inverse functional. This means that all properties are single-valued, except `am:author`, which is multi-valued, and that `am:isbn` is a key of `am:Book`. Likewise, although not shown in Fig. 2,

```
Product
  title         range string
  listPrice     range decimal
  currency      range string
Book is-a Product
  author        range string
  edition       range integer
  isbn          range string
  ean           range string
  detailPageURL range anyURI
  publisher     range Publ
Publ
  name          range string
  address       range string
Music is-a Product
Video is-a Product
PCHardware is-a Product
```

Fig. 1 An OWL schema for a fragment of the Amazon Database

```
Seller
  name                range string
  redistrationDate      range dateTime
  offers                range Offer
Offer
  quantity              range integer
  startPrice            range double
  currency              range string
  seller                range Seller
  product               range Product
Product
  title                 range string
  condition             range string
  returnPolicyDetails   range string
  offers                range Offer
Book is-a Product
  author                range string
  edition               range integer
  publicationYear       range integer
  isbn-10               range integer
  isbn-13               range integer
  publisher             range string
  binding               range string
  condition             range string
Music is-a Product
DVDMovies is-a Product
  ComputerNetworking is-a Product
```

Fig. 2 An OWL schema for a fragment of the eBay Database

all properties, except `eb:author`, have *maxCardinality* equal to 1, and `eb:isbn-10` and `eb:isbn-13` are inverse functional.

2.2 Vocabulary matching and concept mapping

We decompose the problem of schema matching into the problem of *vocabulary matching* and the problem of *concept mapping*. In this section, we introduce both notions with the help of examples.

In what follows, let $\boldsymbol{S}$ and $\boldsymbol{T}$ be two schemas, and $\boldsymbol{V}_S$ and $\boldsymbol{V}_T$ be their vocabularies, respectively. Let $\boldsymbol{C}_S$ and $\boldsymbol{C}_T$ be the sets of classes and $\boldsymbol{P}_S$ and $\boldsymbol{P}_T$ be the sets of datatype or object properties in $\boldsymbol{V}_S$ and $\boldsymbol{V}_T$, respectively.

A *contextualized vocabulary matching* between S and T is a finite set μ of quadruples (v_1, e_1, v_2, e_2) such that

- if $(v_1, v_2) \in \boldsymbol{C}_S \times \boldsymbol{C}_T$, then e_1 and e_2 are the top class $\top$
- if $(v_1, v_2) \in \boldsymbol{P}_S \times \boldsymbol{P}_T$, then e_1 and e_2 are classes in $\boldsymbol{C}_S$ and $\boldsymbol{C}_T$ that must be subclasses of the domains, or the domains themselves, of properties v_1 and v_2, respectively

If $(v_1, e_1, v_2, e_2) \in \mu$, we say that μ *matches* v_1 *with* v_2 *in the context of* e_1 *and* e_2, that e_i *is the context of* v_i and that (e_i, v_i) is a *contextualized concept*, for $i = 1, 2$. A *contextualized property (or class) matching* is a matching defined only for properties (or classes).

Intuitively, a vocabulary matching expresses equivalences between properties and classes in a given context. The context of a property `P` in a vocabulary matching is an RDF class that specifies the `rdf:type` of subjects of existing triples of the form `(?subject P ?object)` for which the matchings holds. The context of a class is always the top class $\top$ (i.e., this notion is not used for class matchings). Note that, if the database instances follows the schema the class of the `?subject` must be either the domain of the property `p` or a subclass of its domain, because the property `p` can only be applied to database instances of that classes.

For example, Table 1 shows a fragment of the vocabulary matching between the schemas in Figs. 1 and 2. The first row indicates that the classes `am:Book` and `eb:Book` are equivalent. The last row indicates that the property `am:Publ` applied to instances of type `am:Publ` is equivalent to the property `eb:publisher` applied to instances of type `eb:Book`.

A *concept mapping from a source schema S to a target schema T* is a set of transformation rules that express concepts of the target schema T in terms of concepts of the source S such that it is possible to translate queries over T into queries over S.

In this paper, we consider queries defined in the SPARQL Query Language for RDF [23]. The query of Fig. 3a returns *titles*, *authors* and *publishers* of book instances from the Amazon database. The variable `?b` in lines 4–6 means that only instances which have the properties *author*, *title*, and *publisher* attached to them match the `WHERE` criteria. The variable `?p` in lines 6 and 7 means that, in addition to the previous criteria, only instances which are related to another instance through the property `am:publisher` match the `WHERE` criteria. This is the JOIN relational operator for RDF graphs. Figure 3b shows an equivalent query over the eBay database.

Let A and E be the schemas of Amazon and eBay databases, respectively, and A_S and E_S be states of these two databases, i.e. A_S and E_S contain individuals and their property values of the two databases. The query depicted in Fig. 3a is valid over the RDF graph $A \cup A_S$.

Now consider the graph $G = E \cup A \cup A_S$, i.e. the Amazon data with the vocabularies of Amazon and eBay. Let us see

Table 1 Example of a vocabulary matching between Amazon and eBay schemas

Amazon		eBay	
v_1	e_1	v_2	e_2
am:Book	T	eb:Book	T
am:title	am:Book	eb:title	eb:Book
am:author	am:Book	eb:author	eb:Book
am:listPrice	am:Product	eb:startPrice	eb:Offer
am:name	am:Publ	eb:publisher	eb:Book

Fig. 3 Simple SPARQL queries over the Amazon and eBay databases

```
PREFIX am:<...>
SELECT ?title ?author ?pub
WHERE
  {?b am:author ?author.
   ?b am:title ?title.
   ?b am:publisher ?p.
   ?p am:name ?pub}
```

a) SPARQL query over the Amazon database

```
PREFIX eb:<...>
SELECT ?title ?author ?pub
WHERE
  {?b eb:author ?author.
   ?b eb:title ?title.
   ?b eb:publisher ?pub}
```

b) SPARQL query over the eBay database

how to add triples to G in order to get the same answer while submitting the previous two queries to G. To do that, we adopt the Semantic Web Rule Language (SWRL) [16], in a simpler syntax, to infer additional triples. An example of the rules in our simplified syntax would be:

1. eb:title(b,t) ← am:title(b,t), am:Book(b)
2. eb:Book(b) ← am:title(b,t), am:Book(b)

The first rule says that, if b is an individual attached to the property am:title and it is of class am:Book then b is attached to the property eb:title. The second rule means that if the same conditions hold then b is of class eb:Book. Note that these two rules can be derived from the second matching depicted on Table 1 because the matching says that the property am:title while attached to instances of class am:Book is equivalent to eb:title when subjects are of class eb:Book. For example, imagine the triples of the form (b, am:title, t) and (b, rdf:type, am:Book). We can directly infer the triples (b, eb:title, t) and (b, rdf:type, eb:Book).

We can extend the previous set of rules using other rows of Table 1 as follows.

3. eb:author(b,a) ← am:author(b,a), am:Book(b)
4. eb:Book(b) ← am:author(b,a), am:Book(b)
5. eb:startPrice(b,pr) ← am:listPrice(b,pr), am:Book(b)
6. eb:Book(b) ← am:listPrice(b,pr), am:Book(b)
7. eb:publisher(b,n) ← am:publisher(b,p), am:name(p,n), am:Book(b)
8. eb:Book(b,n) ← am:publisher(b,p), am:name(p,n), am:Publ(p)
9. eb:Book(b) ← am:Book(b)

```
1. PREFIX am:<...>
2. SELECT ?title ?author ?pub
3. WHERE
4.   {?b am:author ?author.
5.    ?b rdf:type am:Book.
6.    ?b am:title ?title.
7.    ?b am:publisher ?p.
8.    ?p am:name ?pub.
9.    ?p rdf:type am:Publ}
```

Fig. 4 Translate SPARQL query from Fig. 2

Note that rule 8 is not directly derived from Table 1. We will later specify, how to derive such a rule.

Now let R be the set of 9 rules derived from vocabulary matching of Table 1 and $R(G)$ be the set of inferred triples from G by R. Then the queries of Fig. 3 over $G \cup R(G)$ return the same set of answers.

The rules can be used to do query translation. Consider that a query over eBay should be translated into a query over Amazon. According to rule 3, triples of the form (b, eb:author, a) can be derived from triples (b, am:author, a), (b, rdf:type, am:Book). By the same rule, the triple pattern {?b eb:author ?author} can be replaced by {?b am:author ?author. ?b am:author ?author.}. Using rules 1, 3, and 7, the query over eBay can be translated into a query over Amazon as in Fig. 4. The translated query can be simplified if we consider that the domain of am:publisher is am:Book and the domain of am:name is am:Publ. In this case, lines 5 and 9 can be omitted.

3 Instance-based vocabulary matching

3.1 Instance-based technique

In this section, we describe an instance-based process to create contextualized vocabulary matchings that are structurally consistent.

Let S and T be two (OWL Extralite) schemas, and V_S and V_T be their vocabularies, respectively. 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.

A *contextualized vocabulary matching* between S and T is a finite set μ_V of quadruples (v_1, e_1, v_2, e_2) such that

(i) if $(v_1, v_2) \in C_S \times C_T$, then e_1 and e_2 are the top class $\top$
(ii) 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 of v_1 and v_2, respectively
(iii) and these are the only possible quadruples in μ_V

If $(v_1, e_1, v_2, e_2) \in \mu_V$, we say that μ_V *matches* v_1 *with* v_2 *in the context of* e_1 *and* e_2, that e_i *is the context of* v_i and that (v_i, e_i) is a *contextualized concept*, for $i = 1, 2$. A *contextualized property (or class) matching* is a matching defined only for properties (or classes).

We first recall the matching technique for catalogue schemas based on similarity heuristics introduced in [17]. Briefly, a *catalogue* is a relational database whose schema S has a single table. Given a catalogue state U_S, an attribute A of S is represented by the set of values of A that occur in U_S, or by the set of pairs (i, v) such that v is the value of A for the object with id i that occurs in U_S. If the domain of A is a set of strings, the set of values is replaced by a set of tokens, and the attribute representations are reinterpreted accordingly. Similarity, models were then applied to such attribute representations to generate attribute matchings between two catalogue schemas.

Bilke and Naumann [2] propose an instance matching technique where each database tuple is represented by a character string, created by concatenating all attribute values of each tuple. The technique uses k-mean clustering algorithms to find duplicate tuples. The identification of duplicates is necessary for creating (i, v) representations of attributes. However, we note that the representations of the same object in distinct databases may differ in the list of attributes and/or in the attribute values. As a consequence, we may end up with dissimilar tuples that are used to represent the same object.

For example, suppose that we apply the Bilke and Naumann technique to match the two instances that represent the book “The Tragedy of Romeo and Juliet”, whose property-value pairs are shown in Table 2. If we measure the similarity between the sets of tokens by the percentage of common tokens extracted from all property values of each instance, we obtain a score of 43% of common tokens. By contrast, if we consider only the values of properties that match, the similarity increases to 70%. Please note that, to improve the instance matching strategy, we used the fact that `am:Book` matches `eb:Book`, and the fact that several other properties match.

Combining these observations, we propose the four-step vocabulary matching process outlined as follows:

(1) Generate a preliminary property matching using similarity functions.
(2) Use the property matching obtained in Step (1) to generate a class matching.
(3) Use the property matching obtained in Step (1) to generate an instance matching.

Table 2 Example the same book instance representation in eBay and Amazon

eBay	Amazon
`isbn-10 = “039577537X”`	`isbn = “039577537X”`
`isbn-13 = 9780395775370`	`ean = 9780395775370`
`title = “The Tragedy of Romeo and Juliet”`	`title = “Tragedy of Romeo and Juliet: And Related Readings (Literature Connections)”`
`author = “William Shakespeare”`	`author = “William Shakespeare”`
`publisher = “Houghton Mifflin”`	`name = “Houghton Mifflin Company”`
`returnPolicyDetails = “NO RETURNS ARE ACCEPTED”`	-
`condition = “Like New”`	-
`binding = “Hardcover”`	-
-	`listPrice = 18.92`
-	`currency = “USD”`

(4) Use the class matching and the instance matching obtained in Step (2) to generate a refined contextualized property matching.

The final vocabulary matching is the result of the union of the class matching obtained in Step (2) and the refined property matching obtained in Step (4). Step (1) generates a preliminary property matching based on the intuition that "*two properties match iff they have many values in common and few values not in common*". Step (2) creates class matchings that reflect the intuition that "*two classes match iff they have many matching properties*". To work correctly, Steps (2) and (3) require that Step (1) generates preliminary property matchings that do not use (i, v) pairs to represent properties.

In what follows, let S and T be two schemas, V_S and V_T be their vocabularies, P_S and P_T be their sets of properties, and C_S and C_T be their sets of classes, respectively. Let U_S and U_T be fixed sets of triples of S and T, respectively, to be used to compute the vocabulary matchings.

Step (1): Preliminary property matching

Let $\boldsymbol{\mathcal{U}}$ be the universe of all tokens extracted from literals and all URIrefs. Consider a similarity function $\sigma : \boldsymbol{\mathcal{U}} \times \boldsymbol{\mathcal{U}} \rightarrow [0, 1]$, a *similarity threshold* $\tau \in [0, 1]$ and a *related similarity threshold* $\tau' \in [0, 1]$ such that $\tau' < \tau$.

For each property $P \in P_S$, for each class $C \in C_S$ such that C is the domain of P or a subclass of the domain of P, consider the contextualized property $P^C = (P, C)$ and construct the set $\boldsymbol{o[U_S, P^C]}$ *of all* $\boldsymbol{v}$ such that either there are triples of the form `(I,P,v)` and `(I,rdf:type,C')` in U_S or there are triples of the form `(I,P,s)` and `(I,rdf:type,C')` in U_S where v is a token in the literal string s, where $C' = C$ or C' is a subclass of C, and likewise for a property in P_T. We call $\boldsymbol{o[U_S, P^C]}$ the *observed-value representation* of P^C in U_S. This construction explores the fact that P is inherited by all subclasses of its domain.

The *contextualized property matching* $\boldsymbol{\mu_P}$ *between S and T induced by* σ *and* τ, and based on the observed-value representation of properties, is the relation μ_P such that

$$(P, C, Q, D) \in \mu_P \quad \text{iff } \sigma\big(o\big[U_S, P^C\big], o\big[U_T, Q^D\big]\big) \geq \tau. \tag{1}$$

Step (2): Class matching

For each class C in C_S, let $props[S, C]$ be the set of properties in P_S whose domain is C or that C inherits from its superclasses, and likewise for classes in C_T. We call $props[S, C]$ the *property representation* of C in U_S.

The *contextualized class matching* $\boldsymbol{\mu_C}$ *between S and T induced by* σ, τ and μ_P is the relation $\mu_C \subseteq C_S \times C_T$ such that (recall that T is the top class)

$$(C, \mathsf{T}, D, \mathsf{T}) \in \mu_C \quad \text{iff } \sigma\big(props[S, C], props[S, T, D]\big) \geq \tau \tag{2}$$

where $props[S, C, T, D] = relprops[S, C, T, D] \cup not\_relprops[S, C, T, D]$, $relprops[S, C, T, D]$ denotes the set of properties P of class C of S such that there is a property Q of the class D of T such that $(P, C, Q, D) \in \mu_P$ and where $not\_relprops[S, C, T, D]$ denotes the set of properties P of the class D of T such that there are not related properties in S by μ_P. Note that it does not make sense to directly compute $\sigma(props[S, C], props[T, D])$, since $props[S, C]$ and $props[T, D]$ are sets of URIrefs from different vocabularies. To avoid this problem, we replaced $props[T, D]$ by $props[S, C, T, D]$.

Step (3): Instance matching

From the matchings directly induced by σ and τ, computed in the previous step, the process then derives an instance matching and a refined contextualized property matching, as follows.

Figure 5 shows the algorithm that computes the instance matching. Its inputs are S, T, class matching μ_C and property matching μ_P. It also implicitly receives U_S and U_T as input. It outputs the instance matching $\boldsymbol{\mu_I}$ that relates matching class instances in U_S and U_T.

In Fig. 5, if C is a class in C_S, and I is an instance of C in U_S, then $t_I[S, C, I, T, D]$ denotes the set of tokens extracted from all values v such that, for some property $P \in P_S$, for some property Q in P_T, for some class $D \in C_T$, there is a triple (I, P, v) in U_S and there is a quadruple (P, C, Q, D) in μ_P. In addition, if C is a class in C_S, and J is an instance of D in U_T, then $t_J[S, C, T, D, J]$ denotes the set of tokens extracted from all values v such that, for some property $P \in P_S$, for some property Q in P_T, for some class $D \in C_T$, there is a triple (I, Q, v) in U_T and there is a quadruple (P, C, Q, D) in μ_P.

Step (4): Refined property matching

Figure 6 shows the algorithm that computes the refined contextualized property matching. It depends on the following additional definitions. For each $(P, C, Q, D) \in \mu_P$ such that $(C, \mathsf{T}, D, \mathsf{T}) \in \mu_C$, construct the set q of triples (I, u, v) such that there are triples of the form (I, P, u) and `(I,rdf:type,C)` in U_S, there are triples of the form (J, Q, v) and `(J,rdf:type,D)` in U_T, and $(I, C, J, D) \in \mu_I$ (where μ_I is the instance matching of Fig. 3). Define $iv[P, C, Q, D] = (s, t)$ such that $s = \{(I, u)/(\exists v)(I, u, v) \in q\}$ and $t = \{(I, v)/(\exists u)(I, u, v) \in q\}$. We call s the *instance-value representation* of P^C in

Fig. 5 The class instance matching algorithm

```
INSTANCE-MATCHING(S,T,$\mu_C$,$\mu_p$)
for each pair of classes (C,D) in S and T
such that $\mu_C$ matches C with D
  for each pair of instances (I,J) of C and D in $U_s$ and $U_T$
       if $\sigma$($t_I$[S,C,I,T,D],$t_J$[C,C,T,D,J]) ≥ $\tau$ then
          $\mu_I$ = $\mu_I$ ∪ (I,C,J,D)
```

```
CONTEXTUALIZED-PROPERTY-MATCHING(S,T,$\mu_C$,$\mu_I$)
for each pair of classes (C,D) in S and T
such that $\mu_C$ matches C with D
or C′ dominates C and $\mu_C$ matches C′ with D
or $\mu_C$ matches C with D′ and D′ dominates D
    for each pair (P,Q) of properties of C and D
        X = $\sigma$(o[$U_s$,$P^C$],o[$U_T$,$Q^D$])
        if (C matches D) then
            (s,t)=iv[P,C,Q,D]
            Y = $\sigma$(s,t)
        else
            Y = 0
        if max(X,Y) ≥ $\tau'$ then
            $\mu_A$ = $\mu_A$ ∪ (P,C,Q,D)
```

Fig. 6 The contextualized property matching algorithm

Table 3 Automatically obtained vocabulary matching from eBay into Amazon

#	eBay		Amazon		Match
	v_1	e_1	v_2	e_2	type
1	Books	T	Books	T	tp
2	author	B	author	B	tp
3	edition	B	edition	B	tp
4	format	B	biding	B	tp
5	isbn-10	B	isbn	B	tp
6	isbn-13	B	ean	B	tp
7	editionDesc	B	format	B	fp
8	Offer	T	Books		fp

U_S (and likewise for t). This second representation is useful since it helps distinguish between properties with similar sets of values that refer to distinct instances, matched by μ_I.

Returning to the algorithm in Fig. 6, it has similar inputs to the algorithm depicted in Fig. 5. Its output, however, is the contextualized property matching μ_A between properties whose domains are classes directly or indirectly matched by μ_C. The algorithm uses the maximum similarity values computed using the observed-value and the instance-value representations for a pair of properties P and Q, and the more relaxed similarity threshold. Although not shown in Fig. 6, object properties receive a special treatment, since their representations are sets of URIrefs that are compared with help of the instance matching μ_I (computed by the algorithm in Fig. 5).

The final vocabulary matching μ is the union of the class matching μ_C induced by σ, τ and μ_P and the contextualized property matching μ_A computed by the algorithm in Fig. 6.

3.2 Experimental vocabulary matching results

We conducted an experiment to assess the performance of the vocabulary matching process described in Sect. 3.1, using product data obtained from Amazon and eBay websites.

We preferred to use data directly downloaded from the web, rather than using the benchmark proposed in [12], because the last does not include instances and, therefore, is unsuitable to test the proposed process.

We first defined a set of terms, which was used to query both databases. From the query results, we extracted the less frequent terms common in both databases. We then used this set of terms to query the databases once more. This preprocessing step enhanced the probability of retrieving duplicate objects from the databases, which is essential to evaluate any instance-based schema matching technique. We extracted a total of 116,201 records: 16,410 from Amazon and 99,791 from eBay.

As similarity functions, we adopted the contrast model [20] for property matchings, and the cosine distance with TF/IDF for instance matchings. The experiments provided us with enough empirical data to conclude that the contrast model performs better in situations where the goal is to emphasize the differences between two sets of values. This follows because the contrast model allows for parameter calibration.

Table 3 shows sample entries of the vocabulary matching obtained. The headings indicate that e_1 is the context of v_1, and e_2 that of v_2. Also, "B" abbreviates classes `eb:Book` and `am:Book`.

The rightmost column of Table 3 classifies the matchings in types: *tp* for true positive, *fp* for false positive and *fn* for false negative. Since the total number (not all shown in Table 3) of true positives is 25, that of false positives is 4 and that of false negatives is 10, the performance measures therefore are:

$$precision = \frac{tp}{tp + fp} = 86\%, \qquad recall = \frac{tp}{tp + fn} = 71\%,$$

$$F = 2\frac{precision \cdot recall}{precision + recall} = 78\%.$$

Lines 3, 5, and 6 of Table 3 refer to matchings that would have been considered false negatives, if the algorithm in

Fig. 4 ignored the instance-value representation of properties. In this case, the performance measures would drop to:

$$\mathit{precision} = 82\%, \qquad \mathit{recall} = 51\%,$$
$$\mathit{fMeasure} = 63\%.$$

4 Concept mapping induced by vocabulary matching

4.1 Definition

Let S be an OWL Extralite schema in what follows.

We say that S is *well-formed* iff

- for any property p of S, the domain of p is a class of S
- for any object property p of S, the range of p is a class of S
- for any class c of S, if s is defined as a superclass of c in S, then s is also a class of S

We understand S as a theory $T[S] = (A[S], C[S])$ in $\mathcal{ALCQI}$ [9], a dialect of Description Logics, such that

- the concepts and roles of the alphabet $A[S]$ are the classes and properties of S
- the axioms of $C[S]$ are the constraints of S, denoted in $\mathcal{ALCQI}$ as follows:
  - a property p has domain d and range r: $\top \sqsubseteq \forall p.r \sqcap \forall p^-.d$
  - a property p, with range r, is inverse functional: $r \sqsubseteq (\leq 1p^-)$
  - a property p, with domain d, has minCardinality k: $d \sqsubseteq (\geq kp)$
  - a property p, with domain d, has maxCardinality k: $d \sqsubseteq (\leq kp)$
  - a class s is defined as a superclass of c: $c \sqsubseteq s$

In what follows, we will also use from $\mathcal{ALCQI}$ the *intersection* of two concepts, denoted $c \sqcap d$, and the *subsumption* of two concepts, denoted $c \sqsubseteq d$.

Let $\boldsymbol{V}$ be the set of *variables*, which is assumed to be disjoint from the set of concepts of S. A *class literal* is an expression of the form $c(x)$, where c is a class and x is a variable; a *property literal* is an expression of the form $p(x, y)$, where p is a property and x and y are variables; a *literal* is a class literal or a property literal. A *conjunction* is a list of literals separated by commas. A *disjunction* is a list of conjunctions separated by semi-colons. (This notation should be familiar to Prolog programmers).

A *rule* is an expression of one of the forms:

- $c(x) \leftarrow B[x]$, where $c(x)$ is a class literal and $B[x]$ is a disjunction where the variable x occurs in each conjunction
- $p(x, y) \leftarrow B[x, y]$, where $p(x, y)$ is a property literal and $B[x, y]$ is a disjunction where the variables x and y occur in each conjunction

The literal is the *head* and the disjunction is the *body* of the rule. We use the notation $B[x]$ and $B[x, y]$ to stress which variables must occur in the body.

Let I be a set of triples of S. The *universe* of I is the set $U[I]$ of all URIrefs and literals that occur in triples of I.

Consistently with the notion of interpretation of Description Logics, given a class c of S, the *interpretation of c in I* is the set

$$c^I = \{i \in U[I] \mid (i, \texttt{:type}, c) \in I\}$$

and, given a property p of S, the *interpretation of p in I* is the binary relation

$$p^I = \{(i, o) \in U[I] \times U[I] \mid (i, p, o) \in I\}.$$

The *interpretation* of the intersection of two concepts $c \sqcap d$ is the set

$$(c \sqcap d)^I = c^I \cap d^I.$$

We say that the subsumption of two concepts $c \sqsubseteq d$ is *true in I*, denoted $I \models c \sqsubseteq d$, iff $c^I \subseteq d^I$.

Rather than resorting to the formalization in $\mathcal{ALCQI}$, we directly define when a constraint σ of S is *true in I*, denoted $I \models \sigma$:

- if σ declares that a property p has domain d and range r, then $I \models \sigma$ iff $p^I \subseteq d^I \times r^I$
- if σ declares that a property p, with range r, is inverse functional, then $I \models \sigma$ iff, for any $b \subseteq r^I$, $\mathit{card}(\{a \in U[I] \mid (a, b) \in p^I\}) \leq 1$
- if σ declares that a property p, with domain d, has minCardinality k, then $I \models \sigma$ iff, for any $a \subseteq d^I$, $\mathit{card}(\{b \in U[I] \mid (a, b) \in p^I\}) \geq k$
- if σ declares that a property p, with domain d, has maxCardinality k, then $I \models \sigma$ iff, for any $a \subseteq d^I$, $\mathit{card}(\{b \in U[I] \mid (a, b) \in p^I\}) \leq k$
- if σ declares that a class s is a superclass of c, then $I \models \sigma$ iff $c^I \subseteq s^I$

We now turn to the semantics of rules. A *valuation* for the set of variables $\boldsymbol{V}$ in I is a function v that maps the variables in $\boldsymbol{V}$ into elements of $U[I]$.

We extend the notion of interpretation of the right-hand side of the rule as follows. We first define when the right-hand side of rule B is *true in I for v*, denoted $I, v \models B$, inductively as follows:

- if B is of the form $c(x)$ then $I, v \models B$ iff $v(x) \in c^I$
- if B is of the form $p(x, y)$ then $I, v \models B$ iff $(v(x), v(y)) \in p^I$
- if B is of the form C, D then $I, v \models B$ iff $I, v \models C$ and $I, v \models D$
- if B is of the form $C; D$ then $I, v \models B$ iff $I, v \models C$ or $I, v \models D$

The *interpretation* of the right-hand side of a rule of the form $B[x]$ in I is the set

$$B[x]^I = \{a \in U[I] \mid \text{there is a valuation } v \text{ for } \boldsymbol{V} \text{ in } I \text{ such that } I, v \models B[x] \text{ and } v(x) = a\}$$

and the *interpretation* of the right-hand side of a rule of the form $B[x, y]$ in I is the binary relation

$$B[x, y]^I = \{(a, b) \in U[I] \times U[I] \mid \text{there is a valuation } v \text{ for } \boldsymbol{V} \text{ in } I \text{ such that } I, v \models B[x, y] \text{ and } v(x) = a \text{ and } v(y) = b\}.$$

Finally, we say that a set I of triples of S is *consistent* iff I satisfies all constraints of S.

4.2 Derivation from vocabulary matching

Given an OWL schema, we say that a class f *dominates* a class c or the intersection $c = d \sqcap e$ of two classes d and e iff there is a sequence $(c_1, c_2, \ldots, c_n)$ such that

- $f = c_1$ and $c = c_n$
- c_{n-1} subsumes c_n
- 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, or
  - c_{i+1} is a class, c_i is an object property and c_{i+1} is declared as the range of c_i, or
  - c_{i+1} is an object property, c_i is a class and c_i is declared as the domain of c_{i+1}

We also say that $\pi = (c_1, c_2, \ldots, c_n)$ is a *dominance path from c to d* and $\theta = (p_{k_1}, p_{k_2}, \ldots, p_{k_m})$, 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).

Let S and T be two (OWL Extralite) schemas in what follows. Recall that a contextualized vocabulary matching between S and T is a finite set μ_V of quadruples (v_1, e_1, v_2, e_2).

A contextualized vocabulary matching μ_V from S into T is *structurally correct* iff, for all $(v_1, e_1, v_2, e_2) \in \mu_V$ such that v_1 and v_2 are properties:

(i) there is a class f of S such that μ_V matches f with the domain of v_2 and f dominates $d_1 \sqcap e_1$, where d_1 is the domain of v_1

(ii) if v_1 is a datatype property, then the range of v_1 is a subtype of the range of v_2

(iii) if v_1 is an object property, then μ_V matches the range of v_1 with the range of v_2

Let μ_V be a structurally correct contextualized vocabulary matching. A *concept mapping M from S into T induced by* μ_V is a set of rules derived from the quadruples of μ_V as follows.

For each quadruple $(v_1, e_1, v_2, e_2) \in \mu_V$, the concept mapping M contains the following rules:

Case 1: v_1 and v_2 be classes. Then M contains rules of the form

$$\begin{array}{l} v_2(x) \leftarrow v_1(x), \\ s(x) \leftarrow v_1(x) \end{array} \quad \text{for each superclass } s \text{ of } v_2.$$

Case 2: v_1 and v_2 are properties. Let d_1 and d_2 be the domains, and r_1 and r_2 be the ranges of v_1 and v_2 (recall that r_1 and r_2 are XML Schema data types, if v_1 and v_2 are datatype properties, and that μ_V matches the range of v_1 with the range of v_2, if v_1 and v_2 are object properties).

Case 2.1: μ_V matches d_1 with d_2. Then, M contains a rule of the form

$$v_2(x, y) \leftarrow v_1(x, y), e_1(x).$$

Case 2.2: μ_V does not match d_1 with d_2. Let f be a class of S such that μ_V matches f with d_2 and f dominates $d_1 \sqcap e_1$. Let $p_{k_1}, p_{k_2}, \ldots, p_{k_m}$ be the property path corresponding to a dominance path from f to $d_1 \sqcap e_1$. Then, M contains a rule of the form

$$v_2(x, y) \leftarrow p_{k_1}(x, x_1), p_{k_2}(x_1, x_2), \ldots, p_{k_m}(x_{m-1}, z), v_1(z, y), e_1(z)$$

if the property path is non-empty; otherwise the rule reduces to that of case 2.1. (Note that, since μ_V is structurally correct, a dominance path from f to d_1 indeed exists. Also note that, since the dominance path may not be unique, the concept mapping induced by μ_V is not unique).

Note that the contextualized vocabulary matching μ_V may have more than one quadruple for the same concept v_2 of the target schema, which implies that the above process may generate more than one rule for v_2. In addition, v_2 may be a superclass of more than one class which, again, implies that the process described in Case 1, may generate more than one rule for v_2. Therefore, as a last step in the construction of the concept mapping M, we collect all rules for v_2 in a single rule with a disjunctive body. More precisely, if v_2 is a class, and the above process generates rules of the following form:

$$v_2(x) \leftarrow B_i[x], \quad \text{for } i \in [1, n]$$

then we replace all such rules by a single rule ρ of the form

$$v_2(x) \leftarrow B_1[x]; \ldots; B_n[x]$$

and likewise, if v_2 is a property.

We say that a rule ρ in M *defines* a concept v_2 of T iff the head of ρ is of the form $v_2(x)$, if v_2 is a class, or of the form

$v_2(x, y)$, if v_2 is a property (by the transformation described above, M has at most one rule for each concept of T).

However, there might be a concept v_2 of T such that M has no rule that defines v_2. We therefore define T/M as the subset of T restricted to the concepts that M defines. Then the constraints of T/M are the constraints of T defined over such vocabulary. In particular, we can prove that superclasses, domains, and ranges are properly defined in T/M.

Proposition 1 *Let μ_V be a structurally correct contextualized vocabulary matching and M be a concept mapping from S into T induced by μ_V. Then*:

(i) *for any class c of T/M, if s is a superclass of c in T, then s is also a class of T/M.*
(ii) *for any property p of T/M, the domain of p is also a concept of T/M.*
(iii) *for any object property p of T/M, the range of p is also a concept of T/M.*

Proof

(i) Let c be a class of T/M and s be a superclass of c in T. Since c is a class of T/M, by Case 1 of the construction of M, there is a rule in M of the form $c(x) \leftarrow p_1(x)$. Since s is a superclass of c, again by Case 1, there is a rule in M of the form $s(x) \leftarrow p_1(x)$. Hence, s is defined in M, that is, s is a class of T/M.
(ii) Let p be a property of T/M. Let d be domain of p. Since p is a property of T/M, by Case 2, there is a rule in M of the form $p(x, y) \leftarrow B[x, y]$ and a class f of S such that μ_V matches f with the domain d of p. Then by Case 1, there is a rule in M of the form $d(x) \leftarrow f(x)$. Hence, d is defined in M, that is, d is a class of T/M.
(iii) Let p be an object property of T/M. Let r be the range of p. Since p is an object property of T/M, by Case 2, there is a rule in M of the form $p(x, y) \leftarrow B[x, y]$ and a class g of S such that μ_V matches g with the range r of p. Then by Case 1, there is a rule in M of the form $r(x) \leftarrow g(x)$. Hence, r is defined in M, that is, r is a class of T/M. □

Corollary 1 *T/M is a well-defined OWL Extralite schema.*

Finally, we define the function $\bar{M}$ *induced by* M as the mapping from sets of triples of S into sets of triples of T/M such that, for each set of triples I of S, $J = \bar{M}(I)$ iff, for each rule ρ in M

- if ρ is of the form $c(x) \leftarrow B[x]$, then J contains a triple $(i, \texttt{:type}, c)$ iff $i \in B[x]^I$
- if ρ is of the form $p(x, y) \leftarrow B[x, y]$, then J contains a triple (i, p, j) iff $(i, j) \in B[x, y]^I$

We stress that M is used to map queries submitted to the target schema T into queries of the source schema S, whereas $\bar{M}$ is a theoretical device to prove the consistency of the concept mapping, as discussed in the next section.

4.3 Consistency

In this section, we briefly discuss the consistency of OWL Extralite vocabulary matchings, referring the reader to [18] for the detailed definitions and proofs.

In what follows, we use the notion of subsumption as in Description Logic. We say that a class c *dominates* a class d iff there is a sequence $(c_1, c_2, \dots, c_n)$ of classes such that $c = c_1, d = c_n$ and, for each $i \in [1, n-2)$, either c_{i+1} is declared as a subclass of c_i or there is an object property whose domain is c_i and whose range is c_{i+1}, and c_{n-1} subsumes c_n. We consider that a class dominates itself.

A contextualized vocabulary matching μ from S into T is *structurally correct* iff, for all $(v_1, e_1, v_2, e_2) \in \mu$ such that v_1 and v_2 are properties:

(i) there is a class f of S such that μ matches f with the domain of v_2 and f dominates e_1 (recall from the definition of vocabulary matching that e_1 is a subclass of the domain of v_1)
(ii) if v_1 is a datatype property, then the range of v_1 is a subtype of the range of v_2
(iii) if v_1 is an object property, then μ matches the range of v_1 with the range of v_2

A concept mapping γ from S into T *induced by* a structurally correct contextualized vocabulary matching μ is a set of rules derived from μ as suggested by the examples in Sect. 2.2. The rules in γ in turn induce a function $\bar{\gamma}$ that maps sets of triples of S into sets of triples of T.

We say that the declarations of the domain and range of properties, property characteristics, cardinality restrictions, and subclass declarations are the *constraints* of a schema.

We denote the minCardinality and the maxCardinality of a property p by $mC[p]$ and $MC[p]$, respectively. By convention, we take $mC[p] = 0$ (and $MC[p] = \infty$), if minCardinality (or maxCardinality) is not declared for p.

A property q is *no less constrained* than a property p iff $mC[p] \leq mC[q]$ and $MC[p] \geq MC[q]$ and, if p is declared as inverse functional, then so is q. Note that this definition applies even if p and q are from different schemas.

Let S and T be two schemas, μ be a structurally correct contextualized vocabulary matching from S into T, and γ be a concept mapping from S into T induced by μ.

Let ρ be a rule in γ of the form $p(x, y) \leftarrow B[x, y]$. By construction, p is a property of T and all classes and properties that occur in $B[x, y]$ belong to S. We introduce a property of S, denoted $prop[B]$, defined by $B[x, y]$. We say that ρ is *correct* iff $prop[B]$ is no less constrained than p. We then say that γ is *correct* iff all rules in γ are correct.

Finally, we say that a constraint α of T *is relevant for* γ iff α uses only concepts that occur in the heads of the rules in γ. We then say that γ is *consistent* iff, if I is a consistent set of triples of S, then the set of triples of T defined by $J = \bar{\gamma}(I)$ satisfies all constraints of T that are relevant for γ.

Lemma 1 *Let μ be a structurally correct contextualized vocabulary matching and γ be a concept mapping from S into T induced by μ. Assume that γ is correct. Then γ is consistent.*

(The proof generalizes Examples 2, 3, and 4. See [18] for the details).

5 Storing provenance data for matchings

In this section, we discuss the problem of storing provenance data for schema matchings and propose a data model for provenance applied to the algorithm presented in Sect. 2.

Clearly, schema matching process is a laborious task. Automatic or semi-automatic tools that are able to identify such correspondences definitely boost up the process. Nevertheless, schema matching algorithms, in general, must be calibrated so as to achieve better performance in relation to *false positives and negatives*. Leme et al. [17] propose a cross validation process which aims at choosing the best similarity model and calibrations for a given set of test data. In this context, provenance data could be used to store parameters and calibrations for each matching result in order to allow for the identification of the best models.

Benchmarks are also very important to refine matching algorithms and to identify best suited scenarios for each algorithm. Of course, the algorithms must be compared over the same dataset, otherwise it does not make sense to compare performance measures. A classification of the scenarios might also be useful. If schemas are classified according their application domains, we could identify the ones that work better in the geographic domain, for instance.

Finally, matching entries can be validated by using a semi-automated matching process. In this case, we would make the internal representation of schema elements, the intermediate matching calculations and the final similarity degree between elements available. This information would help users to decide and/or validate the matchings.

Our provenance model consists of an OWL *schema* (top left of Fig. 7), modeled as an aggregation of *elements*, specialized into *classes*, *properties*, and *instances*. A *matchable* is any object that can be matched to another object (classes or properties).

Each schema is associated with one of more *datasets* (bottom left of Fig. 7). Each dataset contains a set of triples, which describes the elements of the schema, including instances of classes and properties. A schema also has a set of *features* that can be used to identify the best model for a given scenario. For example, if *feature*[0] contains the classification categories of the schemas, it would be possible to select the best algorithms for the particular domain of feature [0].

From a dataset, *representations* (Set in Fig. 7) for each matchable of the schema are extracted [20]. Each representation is a set of *values*, and has a *type*. For example, a prop-

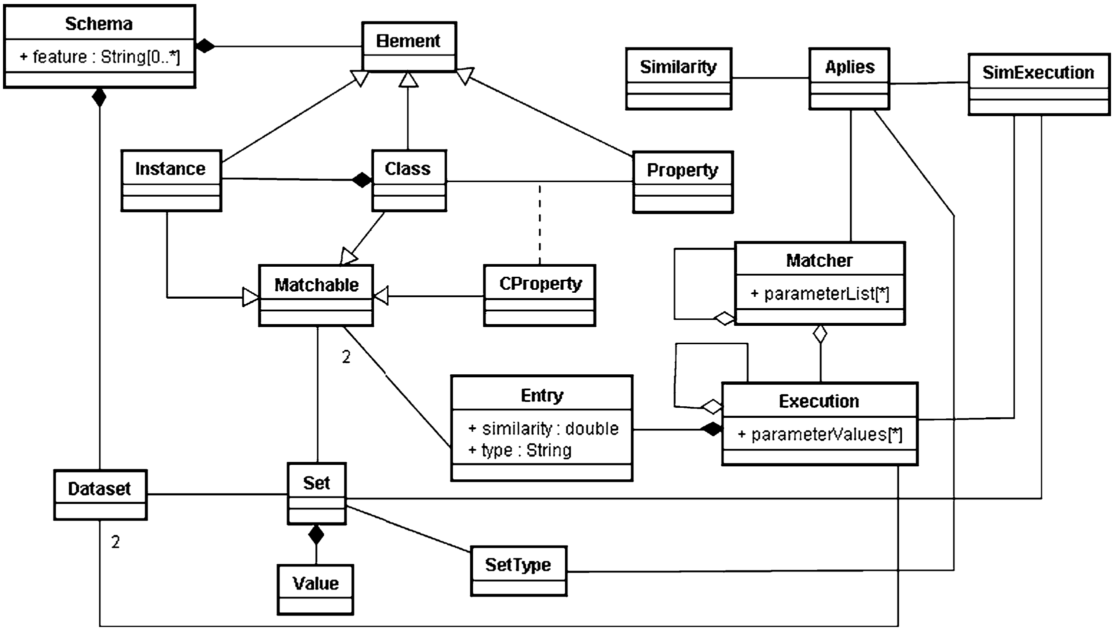

Fig. 7 Diagram of the internal database

erty can be represented by a set of tokens extracted from its observed values, in this case the set is of type *Token* and values are the extracted tokens of the property values. Our matching technique proposes the following representation sets for each type schema elements:

(i) Classes:
   a. Set of properties (denoted by *props*)
(ii) Properties (datatype and object properties)
   a. Set of tokens (denoted by o)
   b. Set of instance value (i, v) pairs (denoted by iv)
(iii) Instances
   a. Set of tokens (denoted by t)

The *Matcher* (top right of Fig. 7), stores descriptions of matching algorithms, or *matchers*, and of *similarity functions* (*Similarity* in Fig. 7). To model the fact that a matching algorithm has a series of matching steps, as in Sect. 3.1, a matcher is modeled as an aggregation of matchers. Each matcher *applies* one or more similarity functions, using a *parameter list*, if available. The matching algorithm described at the end of Sect. 3.1 provides the archetypal example of the family of instance-based matching algorithms that the tool supports. In this example we used the similarity functions *cosine distance* and *contrast model* which are stored in *Similarity*. In step 1, we used the *contrast model* function applied to the token representation of properties (o). In step 2, we applied the *contrast model* function to the property representation of classes (*props*). In step 3, we used *cosine distance* function applied to token representation of instances (t). In step 4, we used *contrast model* function applied to *token* (o) and *instance value* (*iv*) representations of properties. The configuration of the similarity functions is stored by the *Aplies* class (top right of Fig. 7).

Each *execution* of a matcher (bottom right of Fig. 7) stores the *parameter values* that were used, and the similarity functions were applied (*SimExecution* in Fig. 7). It also stores the order in which the similarity functions were applied, as well as values used in the computations. Each execution results in an aggregation of matching entries (*Entry*), which, in turn, model a vocabulary matching. In step 1, we used $\alpha = 1.0$, $\beta = \gamma = 3.5$ and *threshold* $(\tau) =$ *max similarity* $- 31\%$ as the parameters values. In step 2, we used *threshold* $(\tau) = 0.8$ as the parameter values. In this step, because we used the cosine distance, there were no configuration parameters required to run the similarity function. In step 3, we used $\alpha = 1.0$, $\beta = \gamma = 3.5$ and *threshold* $(\tau) =$ *max similarity* $- 26\%$. In step 4, we used $\alpha = 1.0$, $\beta = \gamma = 3.5$ and *threshold* $(\tau) =$ *max similarity* $- 12\%$ for the similarities between token and instance representations of properties. All parameters were stored in the *parameterValues* of *Execution*. The *type* of an *Entry* tells us if it is *false positive*, *false negative* and so on. It is only used in cross validation matchings, as inputs for performance measuring. The similarity is the final similarity measure for each particular entry.

6 Conclusions

In this paper, we proposed hybrid matching techniques based on instance values and on schema information, such as datatypes, cardinality, and relationships. The techniques uniformly apply similarity functions to generate matchings and are grounded on the interpretation, traditionally accepted that "terms have the same extension when true of the same things" [24]. In our context, two concepts match if they denote similar sets of objects. The techniques essentially differ on the nature of the sets to be compared and on the similarity functions adopted. For example and in a very intuitive way, two classes match if their sets of observed instances are similar, two terms from different thesauri match if the sets of instances they classify are similar; properties match if their sets of observed values are similar.

The assumptions that the database schemas we want to match are described in OWL notation, and that data from the databases can be obtained as sets of RDF triples facilitated the construction of matching techniques. However, the techniques introduced in the paper can be directly applied to conceptual schemas described in other conceptual modeling representation languages, such as the relational model [10]. In conjunction, these assumptions permitted us to concentrate on a strategy to unveil the semantics of the database schemas to be matched, without being distracted by syntactical peculiarities. In fact, we consider good practice to provide OWL descriptions of the export schemas of data source providers. In conjunction with WSDL descriptions of the web services encapsulating the backend databases, this measure facilitates the interoperability of databases.

We focused on the more complex problem of matching two schemas that belong to an expressive OWL dialect. We decomposed the problem of OWL schema matching into the problems of vocabulary matching, and the problem of concept mapping. We also introduced sufficient conditions to guarantee that a vocabulary matching induces correct concept mappings. We adopted the contrast model [26] as the preferred similarity function, which proved to efficiently capture the notion of similarity in this context, and described heuristics that led to practical OWL matchings.

Differently from the work of [11, 21], we did not use machine learning techniques to acquire knowledge about matchings. Instead, we captured semantic similarity by adopting similarity functions and heuristics that depended on schema concepts. We consider this strategy to be more general because it identifies matching candidates that do not belong to the training corpus.

Unlike any of the instance-based techniques previously defined, see Sect. 1, the OWL schema matching process we described uses similarity functions to induce vocabulary matchings in a non-trivial way. The results demonstrated that the proposed technique performs well, with precision and recall rates around 80%.

Contrasting to the work of [6, 27], which measure similarity between concepts based on the commonalities between sets of values alone, we made use of similarity functions that took into account not only the commonalities, but also the differences between concepts.

Differently from the work of [2], we overcame the limitations of representing instances using strings that concatenated all of its property values, by representing instances using strings that were constructed using only matching properties, as the first approximation.

As future work, we are considering three broad areas. First, further work is required on techniques to gradually construct the matchings as new data becomes available, which is typical of a query mediation environment. We refer the reader to [3, 5] for discussions about this issue. Second, belief revision techniques should be investigated to help adjust the mediated schemas in time, as new data sources are integrated into the mediated environment. Third, implementation issues are pending, although [14, 15] is a step in this direction.

In summary, unlike previous approaches, we proposed hybrid matching techniques that are uniformly grounded on similarity functions to generate matchings between simple catalogue schemas, as well as between more complex OWL schemas. We introduced the idea of decomposing the problem of schema matching into the problems of vocabulary matching and concept mapping, which are often confused in the literature. We also showed when a vocabulary matching induces correct concept mappings, with respect to the integrity constraints of the schema, an issue also frequently overlooked in the literature.

Acknowledgements This work was partly supported by CNPq under grants 557128/2009-9, 142103/2007-1, 301497/2006-0, and 473110/2008-3, and by FAPERJ under grant E-26/170028/2008.

References

1. Bechhofer S, van Harmelen F, Hendler J, Horrocks I, McGuinness DL, Patel-Schneider PF, Stein LA (2004) OWL web ontology language reference. W3C recommendation. Last access on Dec 2008 at: http://www.w3.org/TR/owl-ref/
2. Bilke A, Naumann F (2005) Schema matching using duplicates. In: Proceedings of the 21st international conference on data engineering, pp 69–80
3. Brauner DF, Casanova MA, Milidiú RL (2006) Mediation as recommendation: an approach to the design of mediators for object catalogs. In: On the move to meaningful internet systems 2006: OTM 2006 workshops. Lecture notes in computer science, vol 4277. Springer, Berlin, pp 46–47
4. Brauner DF, Casanova MA, Milidiú RL (2007) Towards gazetteer integration through an instance-based thesauri mapping approach. In: Advances in geoinformatics; VIII Brazilian symposium on geoinformatics (GEOINFO), pp 235–245
5. Brauner DF, Gazola A, Casanova MA (2008) Adaptive matching of database web services export schemas. In: Proceedings of the 10th international conference on enterprise information systems (ICEIS), pp 49–56
6. Brauner DF, Intrator C, Freitas JC, Casanova MA (2007) An instance-based approach for matching export schemas of geographical database Web services. In: Proceedings of the IX Brazilian symposium on geoinformatics (GEOINFO), pp 109–120
7. Casanova M, Breitman K, Brauner D, Marins A (2007) Database conceptual schema matching. Computer 40(10):102–104
8. Castano S, Ferrara A, Montanelli S, Racca G (2004) Semantic information interoperability in open networked systems. In: Proceedings of semantics for grid databases, first international IFIP conference (ICSNW). Lecture notes in computer science, vol 3226. Springer, Berlin, pp 215–230
9. Chomicki J, Saake G (1998) Description logics for conceptual data modeling. In: Logics for databases and information systems. Springer, Berlin, chapter 8
10. Codd EF (1970) A relational model of data for large shared data banks. Commun ACM 13(6):377–387
11. Doan A, Domingos P, Halevy AY (2001) Reconciling schemas of disparate data sources: a machine-learning approach. In: Proceedings of the 2001 ACM SIGMOD international conference on management of data, vol 30, pp 509–520
12. Duchateau F, Bellahsène Z, Hunt E (2007) XBenchMatch: a benchmark for XML schema matching tools. In: Proceedings of the 33rd international conference on very large data bases, demo sessions: group 1, pp 1318–1321
13. Euzenat J, Shvaiko P (2007) Ontology matching. Springer, Berlin
14. Gazola A (2008) A software infrastructure for catalog matching. Master's thesis, Departamento de Informática, PUC-Rio
15. Gazola A, Brauner D, Casanova MA (2007) A mediator for heterogeneous gazetteers. In: Poster session of the 22nd Brazilian symposium on database
16. Horrocks I, Patel-Schneider PF, Boley H, Tabet S, Grosofand B, Dean M (2004) SWRL: A semantic web rule language combining OWL and RuleML. W3C member submission. Last access on Dec 2008 at: http://www.w3.org/Submission/SWRL/
17. Leme LAP, Brauner DF, Breitman KK, Casanova MA, Gazola A (2008) Matching object catalogues. J Innov Syst Softw Eng 4(4):315–328
18. Leme LAPP (2009) Conceptual schema matching based on similarity heuristics. DSc thesis (Advisor: Casanova MA), Department of Informatics, Pontifical Catholic University of Rio de Janeiro
19. Leme LAPP, Casanova MA, Breitman KK, Furtado AL (2008) Evaluation of similarity measures and heuristics for simple RDF schema matching. Monografias em Ciência da Computação MCC44/08, Department of Informatics, Pontifical Catholic University of Rio de Janeiro
20. Leme LAPP, Casanova MA, Breitman KK, Furtado AL (2009) Instance-based OWL schema matching. In: Proceedings of the 11th international conference on enterprise information systems. Lecture notes in business information processing, vol 24. Springer, Berlin, pp 14–26
21. Madhavan J, Bernstein P, Doan A, Halevy A (2005) Corpus-based schema matching. In: Proceedings of the 21st international conference on data engineering, pp 57–68
22. OMG (2009) OMG unified modeling language, superstructure

23. Prud'hommeaux E, Seaborne A (2008) SPARQL query language for RDF. W3C recommendation. Last access on Dec 2008 at: http://www.w3.org/TR/rdf-sparql-query

24. Quine WV (1968) Ontological relativity. J Philos 65(7):185–212

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

26. Tversky A, Gati I (1978) Studies of similarity. Cogn Categ 1:79–98

27. Wang J, Wen J, Lochovsky F, Ma W (2004) Instance-based schema matching for web databases by domain-specific query probing. In: Proceedings of the 13th international conference on very large data bases, pp 408–419