

Database Mediation using Multi-Agent Systems

Luiz A. P. Paes Leme, Marco A. Casanova, Karin K. Breitman, Antonio L. Furtado
Departamento de Informática
Pontificia Univ. Católica do Rio de Janeiro
Rua Marquês de São Vicente, 225 – Rio de Janeiro – RJ – Brazil – CEP 22.453-900
{lleme, casanova, karin, furtado}@inf.puc-rio.br

Abstract – This paper first proposes a multi-agent architecture to mediate access to data sources. The mediator follows the classical approach to process user queries. However, in the background, it post-processes query results to gradually construct matchings between the export schemas and the mediated schema. The central theme of the paper is an extensional schema matching strategy based on similarity functions. The paper concludes with experimental results that assess the quality of the matching strategy.

Keywords: *schema matching, database mediation, multi-agent systems, similarity functions*

I. INTRODUCTION

A data source consists of a wrapper that provides access to a backend database. The export schema of a data source describes the subset of the backend database schema that is visible to the clients. A conventional mediator is a software component that facilitates access to the data sources through a mediated schema. It receives queries over the mediated schema, translates them to the export schemas, forwards the translated queries to the data sources and returns the results to the clients. To translate the queries, the mediator uses a set of mappings between the export schemas and the mediated schema.

In this paper, we investigate the design of a mediator that dynamically constructs the mediated schema and the schema mappings to reflect changes in the data sources. We concentrate on the problem of dynamically generating schema mappings.

The design of the mediator assumes that: (1) the export schemas are written in a dialect of OWL; (2) OWL properties with similar ranges are semantically equivalent; (3) instances with similar property values are semantically equivalent. The first assumption should be viewed as a recommendation to those designing databases with the intent of making them freely available over the Web. The second and third assumptions reflect an approach to schema matching based on detecting how the same real-world objects are represented in different databases and using such information to match the schemas. Such approach is often called extensional or semantic.

The major contributions of the paper are four-fold. First, we describe a multi-agent mediation architecture that supports the extensional approach. Second, we precisely

define the notion of OWL schema matching. Third, we introduce a matching strategy induced by similarity functions. Fourth, we describe experiments that evaluate the precision of the matching strategy.

As for related work, Bilke and Naumann [1] describe an extensional technique based on similarity algorithms. Brauner et al. [2] adopt the same idea to match pairs of thesauri. Wang et al. [11] describe a technique based on query probing to match Web databases, which relies on human intervention to select a set of typical instances. Brauner et al. [3,4] describe a matching algorithm based on measuring the similarity between the property domains of distinct databases. Madhavan et al. [9] propose the use of a set of schemas and mappings to help the schema matching algorithms. They use predictor algorithms that measure the similarity between schema elements, adopted in the PayGo architecture [10]. Contrasting with [11], we do not require the use of a pre-defined mediated schema and a set of global instances, which are hard to define. The work reported here also contrasts with [1], [3], [4] and [9] in that the matching functions adopted are based on customized Tversky contrast models, which may result in more general many-to-many matchings. Finally, we note that the XBenchmark [5] provides a set of sample schemas and reference matchings. However, it does not provide sample data, which precludes its use to assess extensional methods.

This paper is organized as follows. Section 2 outlines the mediator architecture. Section 3 introduces the OWL dialect adopted and the notions of vocabulary matching and concept mapping. Section 4 describes a technique to obtain vocabulary matchings. Section 5 contains experimental results. Finally, Section 6 lists the conclusions.

II. AGENT ARCHITECTURE

A *data source* consists of a *wrapper* that provides access to a backend database. The *export schema* of a data source describes a subset of the backend database schema that is visible to the clients. A classical *mediator* receives user queries over a *mediated schema*, translates them to the export schemas of data sources, forwards the translated queries to the corresponding source, and returns the complete set of results to the clients. To translate queries, the mediator uses a set of mappings between the export schemas and the mediated schema.

The problem of query mediation becomes a challenge in

the context of the Web, where the number of sources may be enormous and the mediator does not have much control over them. The sources may join or leave the mediated environment or change their schemas at will. This scenario requires a strategy to dynamically update the mediated schema and the schema mappings. It also demands strategies to discover new data source and to classify them according to their application area. We address in this paper the problem of dynamically generating schema mappings.

The mediation architecture we propose has four basic characteristics. First, it assumes that the export schemas and the mediated schema are written in the OWL dialect defined in Section 3, which facilitates the matching process, and yet retains sufficient expressiveness.

Second, it departs from the classical mediation process and constructs the mediated schema and the schema mappings a posteriori, as new data sources join the environment.

Third, it adopts an instance-based technique, which consists of comparing sample data obtained from the different data sources to match the export schemas with the mediate schema and to revise the mediated schema. This technique therefore requires collecting and maintaining sample data from the data sources.

Four, it uses as data samples the results from client queries, instead of sending *probing queries* to obtain sample data from a new data source. This is justified because creating probing queries proved to be a difficult task [4].

Figure 1 summarizes the mediation architecture. Each data source joins the mediation environment by sending its export schema to a *mediator agent*.

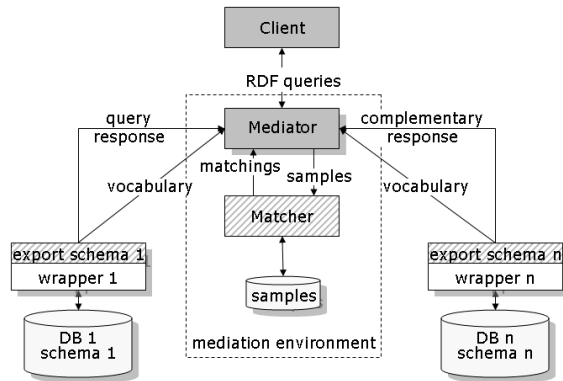


Figure 1: Mediation architecture

The mediator agent maintains the mediated schema MS and the export schemas. The mediator receives a query Q from a client, translates Q to local queries, and sends them to the appropriated data sources. Q is written using the concepts in the mediated schema.

After processing Q , the mediator agent sends the local queries and their results to a *matcher agent*, which stores them in a *sample repository*. The matcher agent periodically processes all samples, generates mappings between the export schemas and the mediated schema, and revises the mediated schema. This process continues indefinitely to

maintain the mediated schema and the mappings. The matcher agent timestamps the samples to facilitate discarding old ones and thereby reduce their influence in the process.

The final architecture features a set of mediator agents and a recruiter. The recruiter receives queries, selects a mediator according to its workload, and forwards the query to it.

This paper concentrates on the problem of creating the schema mappings and does not address the problem of revising the mediated schema. Very briefly, we consider the mediated schema MS and the external schemas as OWL vocabularies (see Section 3.1). MS is visible to the clients, whereas the external schemas are not. We initialize MS with the export schema of the data source that first joins the environment. When a new data source, with export schema ES , joins the environment, we add to MS the vocabulary defined in ES , until enough samples have been collected to compute the matching between ES and MS and to revise MS appropriately.

For example, MS may initially include the concept `Video` and the new concept `DVDMovies` from ES . After the mediator processes a few queries, the matcher agent may have collected enough samples to detect that `DVDMovies` and `Video` in fact denote the same concept. Hence, `DVDMovies` is matched with `Video` and dropped from MS . Clients will no longer observe `DVDMovies` and will use `Video` instead.

The challenge therefore lies in deciding when two concepts match, based on data samples, the problem we address in the rest of the paper.

III. OWL SCHEMA MATCHING

A. OWL Extralite

Recall that a *resource* is anything identified by an URIref and that an *XML namespace* or *vocabulary* is a set of URIrefs. A *literal* is a character string that represents an XML Schema datatype value. We refer the reader to Breitman et al. (2007) for the 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 p .

The *Web Ontology Language* (OWL) describes classes and properties in a way that facilitates machine interpretation of Web content

We will work with an OWL dialect, that we call *OWL Extralite*. It supports *classes*, *datatype* and *object properties*, subclasses, and individuals. The *domain* of a datatype or object property is a class, the *range* of a datatype property is an XML schema type, whereas the *range* of an object property is a class. As property restrictions, the dialect admits *minCardinality* and *maxCardinality*, with the usual meaning. As property characteristic, it allows just the *inverseFunctional* property, which captures simple keys.

We note that only OWL Full supports the *inverseFunctional* property for datatype properties.

An *OWL schema* (more often called an *OWL ontology*) is a collection of RDF triples that use the OWL vocabulary. A *concept* of 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. The scope of a property name is global to the OWL schema, and not local to the class indicated as its domain.

To indicate that an instance *s* belongs to a class *c*, we use a triple of the form (*s*, `rdf:type`, *c*); to indicate that an instance *s* has a datatype property *p* with value *v*, we use a triple of the form (*s*, *p*, *v*); and to indicate that an object property *p* relates an instance *s* with an instance *o*, we use a triple of the form (*s*, *p*, *o*).

In the rest of the paper, when we refer to a schema, we mean an OWL Extralite schema.

Figures 2 and 3 show OWL schemas for fragments of the Amazon and the eBay databases, using a simplified and unofficial 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 OWL schemas, and qualified names of the form `V:T` to indicate that `T` is a term of the vocabulary `V`.

In Figure 2, 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 Figure 2, 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 Figure 3, all properties, except `eb:author`, have `maxCardinality` equal to 1, and `eb:isbn-10` and `eb:isbn-13` are inverse functional.

We adopt the SPARQL Query Language to express queries. Briefly, a triple pattern is like an RDF triple, except that the subject, predicate or object may be a variable. A basic graph pattern is a set of triple patterns. A basic graph pattern matches a subgraph of the RDF data when the RDF terms from that subgraph may be substituted for the variables and the result is an RDF graph equivalent to the subgraph. A graph pattern is a set of basic graph pattern combined via the union construct. An SPARQL query consists of a SELECT clause, that identifies the variables to appear in the query results, and the WHERE clause, that defines the graph pattern to match against the data graph.

For example, the query in Figure 4.a returns titles of book instances from the Amazon database. The variable `?b`

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			

Figure 2. An OWL schema for the Amazon Database

Seller			
name	range	string	
redistributionDate	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			

Figure 3. An OWL schema for the eBay Database.

in lines 5 and 6 indicates that the instance that has property `am:title` must also be an instance of the class `am:Book`. If line 5 were omitted, the query would return titles of instances of the class `am:Product`, since this class is the domain of `am:title` (see Figure 2). Figure 4.b returns titles and authors of books, but in this case there is no need to restrict the type of the instances, since the domain of `am:author` is `am:Book` (again see Figure 2). Figure 4.c extends the query in Figure 4.b by adding to its result set the name of the publisher of the book. Line 6 of Figure 4.c is analogous to an equijoin, in the relational model, between classes `am:Book` and `am:Publisher` of the Amazon schema.

```

1. PREFIX am:<http://amazon.com>
2. prefix rdf:<http://www.w3.org/1999
   /02/22-rdf-syntax-ns#>
3. SELECT ?t4. WHERE
5.   {?b rdf:type am:Book.
6.    ?b am:title ?t}

```

a) SPARQL query over the Amazon schema returning titles of books

```

1. PREFIX am:<http://amazon.com>
2. SELECT ?t ?a
3. WHERE
4.   {?b am:author ?a.
5.    ?b am:title ?t}

```

b) SPARQL query over the Amazon schema returning title and author of books

```

1. PREFIX am:<http://amazon.com>
2. SELECT ?t ?a ?n3. WHERE
4.   {?b am:author ?a.
5.    ?b am:title ?t.
6.    ?b am:publisher ?p.
7.    ?p am:name ?n}

```

c) SPARQL query over the Amazon schema returning title, author and publisher of books

Figure 4. SPARQL queries over the Amazon schema

The SELECT keyword may be replaced by the CONSTRUCT keyword to return an RDF graph, instead of a relation. Figure 5 shows a query which returns triples of publisher names, from the Amazon database, rewritten as triples of the eBay database.

```

1. PREFIX am:http://amazon.com
2. PREFIX eb:<http://ebay.com>
3. CONSTRUCT {?b eb:publisher ?n}
4. WHERE
5.   {?b am:publisher ?p.
6.    ?p am:name ?n}

```

Figure 5. SPARQL query returning an RDF graph

Finally, again to improve readability and save space, we express schema mappings using rules that follow a syntax derived from SPARQL and inspired on Datalog and the Semantic Web Rule Language [6].

A rule is an expression of the form “ $H \leftarrow B$ ”, where H , the head of the rule, is triple pattern, and B , the body of the rule, is a triple pattern or a basic graph pattern such that all variables that occur in the head also occur in the body.

An example of a rule would be:

```

?b eb:publisher ?n ←
    {?b am:publisher ?p.
     ?p am:name ?n}

```

which expresses that, if $?b$ and $?p$ are related by $am:publisher$, and $?p$ and $?n$ by $am:name$, then $?b$ and $?n$ are related by $eb:publisher$.

B. Vocabulary Matchings and Concept Mappings

We decompose the problem of schema matching into the problems of *concept mapping* and *vocabulary matching*. We first introduce both concepts informally and present more formal definitions at the end of the section.

A *concept mapping* from a *source schema* S to a *target schema* T is a set of rules which express concepts of T in terms of concepts of S such that it is possible to translate queries over T to queries over S . Returning to Section 2, the target schema would be the mediated schema and the source schema, an external schema.

From the schemas of our running example, shown in Figures 2 and 3, and an analysis of sample data (see Section 5), one may infer that properties $am:title$ and $eb:title$ are likely to be equivalent because their names are syntactically equal and the experiments describe in Section 5 show that the set of values of both properties are very similar. This mapping can be expressed as the following rule

$$?p \text{ eb:title } ?t \leftarrow ?p \text{ am:title } ?t$$

which indicates that $eb:title$ from the eBay schema can be translated to $am:title$ from the Amazon schema, i.e., triple patterns of the form

$$?p \text{ eb:title } ?t$$

can be translated to triple patterns of the form

$$?p \text{ am:title } ?t$$

Figure 6 shows a query over the eBay schema on the left and its translation to the Amazon schema on the right.

<pre> PREFIX eb:<...> SELECT ?t WHERE {?s eb:title ?t} </pre>	<pre> PREFIX am:<...> SELECT ?t WHERE {?s am:title ?t} </pre>
---	---

Figure 6. Equivalent queries over the eBay and the Amazon databases that return titles

However, a closer analysis of the sample data might not confirm the equivalence between $am:title$ and $eb:title$ for all classes. For example, lines 1 and 2 of Table I indicate that $am:title$, restricted to $am:Book$, is indeed equivalent to $eb:title$, restricted to $eb:Book$, and likewise $am:title$, restricted to $am:Music$, is equivalent to $eb:title$, restricted to $eb:Music$.

Table I illustrates a fragment of what we call a *vocabulary matching between S and T* , which expresses equivalences between properties and between classes, under to a given by context.

In view of Table I, the previous rule should be replaced by the following two rules.

```

?p eb:title ?t ←
  {?p am:title ?t. ?p rdf:type am:Book}
?p eb:title ?t ←
  {?p am:title ?t. ?p rdf:type am:Music}

```

TABLE I: FRAGMENT OF VOCABULARY MATCHING BETWEEN AMAZON AND EBAY SCHEMAS.

Amazon Database		eBay Database	
Concept	Context	Concept	Context
am:title	am:Book	eb:title	eb:Book
am:title	am:Music	eb:title	eb:Music
am:name	am:Publ	eb:publisher	eb:Book
am:Book	T	eb:Book	T
am:Music	T	eb:Music	T

These new rules state that, although the domain of `eb:title` is `eb:Product`, properties `eb:title` and `am:title` are equivalent only for instances of classes `am:Book` or `am:Music`, i.e., the concept of title has a narrower meaning in the Amazon database.

Figure 7 shows the same query over the eBay schema as Figure 6 on the left and its translation to the Amazon schema, using these new rules, on the right.

<pre>PREFIX eb:<...> SELECT ?t WHERE {?s eb:title ?t}</pre>	<pre>PREFIX am:<...> PREFIX rdf:<...> SELECT ?t WHERE {{?s am:title ?t. ?s rdf:type am:Book} union {?s am:title ?t. ?s rdf:type am:Music}}</pre>
---	--

Figure 7. Equivalent queries over the eBay and the Amazon databases that return titles, when only book and music titles are equivalent

In general, in a vocabulary matching, the *context* of a property p is a subclass of its domain that restricts the subject $?s$ in triple patterns of the form “ $?s p ?o$ ”. The *context* of a class is always the top class `T`, i.e., a vocabulary matching does not distinguish the cases where two classes are equivalent.

The translation of a vocabulary matchings into rules that express concept matchings is not always straightforward. For example, consider line 3 of Table I, which indicates that `am:name`, in the context of `am:Publ`, matches `eb:publisher`, in the context of `eb:Book`.

If we mimic the previous examples, we would generate the following rule:

`?b eb:publisher ?n ← ?b am:name ?n`

Note that the body of the rule does not include the triple pattern “`?b rdf:type am:Publ`”, which would be redundant since the domain of `am:name` is indeed `am:Publ`. This rule induces the query translation shown in Figure 8.a.

However, this rule is wrong. Indeed, observe that: (1) `am:name` has domain `am:Publ` and, hence, in the rule body, variable $?b$ denotes an instance of `am:Publ`; (2) `eb:publisher` has domain `eb:Book` and, hence, in the rule head, variable $?b$ denotes an instance of `eb:Book`; (3)

<pre>PREFIX eb:<...> SELECT ?n WHERE {?s eb:publisher ?n}</pre>	<pre>PREFIX am:<...> PREFIX rdf:<...> SELECT ?n WHERE {?s am:name ?n}</pre>
---	---

a) Wrong translation of an eBay query

<pre>PREFIX eb:<...> SELECT ?t, ?n WHERE {?s eb:publisher ?n}</pre>	<pre>PREFIX am:<...> PREFIX rdf:<...> SELECT ?n WHERE {?s am:publisher ?p. ?p am:name ?n}</pre>
---	--

b) Correct translation of an eBay query

Figure 8. Equivalent queries over the eBay and the Amazon schemas requiring the translation of `eb:publisher`

Table I does not match `am:Publ` and `eb:Book`, which indicates that instances of `am:Publ` cannot be reinterpreted as instances of `eb:Book`.

The correct rule would be:

`?b eb:publisher ?n ←`
`{?b am:publisher ?p. ?p am:name ?n}`

This rule indicates that, to translate a triple pattern

`?b eb:publisher ?n`

from the eBay schema to the Amazon schema, we have to perform the equivalent of an equijoin, expressed by the basic graph pattern

`{?b am:publisher ?p. ?p am:name ?n}`

In words, we have to follow the object property `am:publisher` from an instance $?b$ of `am:Book` to an instance $?p$ of `am:Publ`, and then follow the datatype property `am:name` from $?p$ to the literal $?n$. Again note that the body of the rule does not include the triple pattern “`?p rdf:type am:Publ`”, which would be redundant since the domain of `am:name` is indeed `am:Publ`. Figure 8.b shows query translation induced by this new rule.

Note that this rule can only be partially inferred from the vocabulary matching, since line 3 of Table I shows that `eb:publisher`, a concept of the target schema (eBay), matches `am:name`, a concept of the source schema (Amazon). This indicates that both properties have similar values. However, no line of Table I indicates that their domains match, which respectively are `eb:Book` and `am:Publ`.

The body of the above rule can only be generated by analyzing the schema definitions and the vocabulary matching. Indeed, note that `eb:Book`, the context of `eb:publisher`, matches `am:Book`, and there is an equijoin path in the Amazon schema from `am:Book` to `am:Publ`, the context of `am:name`. The equijoin path is expressed by the basic graph pattern

`{?b am:publisher ?p. ?p am:name ?n}`

If there were no matching class for `eb:Book` or if there were no path from `am:Book` to `am:Publ`, we would say that the vocabulary matching is *inconsistent*, which implies that rows would have to be removed from the vocabulary matching.

Because of the above considerations, we say that the concept mapping is *derived from* the vocabulary matching or, conversely, the vocabulary matching *induces* the concept mapping.

Note that the generation of a concept mapping is not a symmetric process. In our running example, if we consider the eBay schema as the source and the Amazon schema as the target, the property `am:name` will have no translation, since there will not be a concept equivalent to `am:Publ` in the eBay schema.

The last type of rule covers how to translate classes. Line 4 of Table I indicates that `am:Book` matches `eb:Book`. The corresponding rule would be:

```
?b rdf:type eb:Book ←
    ?b rdf:type am:Book
```

which indicates that a triple pattern of the form

```
?s rdf:type eb:Book
```

must be translated to the pattern

```
?s rdf:type am:Book
```

However, consider the following query over the eBay schema:

```
PREFIX eb:<http://ebay.com>
SELECT ?s
WHERE
  {?s rdf:type eb:Product}
```

If the above rule were used, the result set would contain instances of `eb:Book`, `eb:Music`, `eb:DVDMovies` and `eb:ComputerNetworking`. Since Table I only indicates that `am:Book` matches `eb:Book` and `am:Music` matches `eb:Music`, it is expected that an equivalent query over the Amazon schema returns instances of `am:Book` and `am:Music`:

```
PREFIX am:<http://amazon.com>
SELECT ?s
WHERE
  {{?s rdf:type am:Book} union
  {?s rdf:type am:Music}}
```

This translation can be achieved by creating two additional rules:

```
?p rdf:type eb:Product ←
    ?s rdf:type am:Book
?p rdf:type eb:Product ←
    ?s rdf:type eb:Music
```

This concludes our informal discussion. In the rest of this section, we formalize the definitions of vocabulary

matching and concept mapping.

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, D_S and D_T be the sets of datatype properties, and O_S and O_T be the sets of object properties in V_S and 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 $(v_1, v_2) \in C_S \times C_T$, then e_1 and e_2 are the top class \top
- if $(v_1, v_2) \in D_S \times D_T \cup O_S \times O_T$, then e_1 and e_2 are classes in V_S and V_T that must be subclasses of the domains 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).

Let \mathcal{Q} be a rule language that supports the definition of classes and properties. A *concept mapping* from S into T in \mathcal{Q} is a set γ of expressions of \mathcal{Q} that define concepts in T in terms of the concepts of S . Schemas S and T are called the *source* and the *target* of the concept mapping.

Finally, to detect when two instances denote the same real-world object, we need a third notion. Let U_S and U_T be sets of instances of S and T , respectively. An *instance matching* from U_S into U_T is a partial, many-to-many relation $\mu_I \subseteq U_S \times C_S \times U_T \times C_T$. We say that an instance I of a class C in U_S *matches* an instance J of a class D in U_T iff $(I, C, J, D) \in \mu_I$. A *class instance matching* is an instance matching defined only for class instances.

IV. INSTANCE-BASED VOCABULARY MATCHINGS

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

We first recall the matching technique for catalogue schemas based on similarity heuristics described in [8]. 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.

We also recall that the instance matching technique described in [1]. Briefly, the technique represents each database tuple as a character string and uses k-mean clustering algorithms to find duplicate tuples. However, we note that the representations of the same object in distinct databases may differ in the list of attributes and in the attribute values. We may thus end up with dissimilar tuples that represent the same object.

For example, suppose that we apply the instance matching technique to match instances that represent the book “The Tragedy of Romeo and Juliet”. Table II shows

TABLE II. EXAMPLE OF REPRESENTATIONS IN THE EBAY AND THE AMAZON DATABASES OF THE SAME BOOK INSTANCE.

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"

their lists of property-value pairs. If we measure the similarity between the sets of 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 the properties that match, the similarity increases to 70%. However, note that, to improve the instance matching strategy, we used the fact that `am:Book` matches `eb:Book`, and the fact that several properties match.

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

- (1) Generate a preliminary property matching using similarity functions.
- (2) Use the property matching obtained in Step (1) to generate: (a) a class matching; and (b) a class instance matching.
- (3) Use the class matching and class instance matching obtained in Step (2) to generate a refined contextualized property matching.
- (4) The final vocabulary matching is the result of the union of the property and class matchings obtained in Steps (2) and (3), adjusted until it becomes consistent.

Step (1) generates preliminary property matchings based on the intuition that "two properties match iff they that have many common values and few non-common values". Step (2) creates class matchings that reflect the intuition that "two classes match iff they have many matching properties". However, to work correctly, Step (2) requires that Step (1) generates preliminary property matchings only for highly similar properties.

For example, in the experiments described in Section 4, with data from the eBay and the Amazon databases, if we use a threshold $\tau=0.12$, then `eb:level`, with context `eb:Seller`, matches `am:color`, with context `am:PCHardware`, and `eb:title`, with context `eb:Music`, matches `am:title`, with content

`am:Video`. These property matchings may cause classes `eb:Seller` and `am:PCHardware` to match, as well as `eb:Music` and `am:Video`, depending on the threshold and the total amount of common properties among the classes (as discussed below, class matching depends on the similarity between sets of properties). If we increase the threshold to 0.13, the previous property matchings do not hold and we may avoid the above unwanted class matchings.

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 instances of S and T , respectively, used to compute the vocabulary matchings.

Let \mathcal{U} be the universe of all tokens extracted from literals and all URIs. Consider a similarity function $\sigma : \mathcal{U} \times \mathcal{U} \rightarrow [0, 1]$, 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 $o[S, P^C]$ of all values v such that there are triples of the form (I, P, v) and $(I, \text{rdf:type}, C')$ in U_S , where $C' = C$ or C' is a subclass of C , and likewise for a property in P_T . We call $o[S, P^C]$ the *observed-value representation* for P^C in U_S . This construction explores the fact that P is inherited by all subclasses of its domain.

The basis of the matching process are *matchings directly induced by σ and τ* , defined as follows.

The *contextualized property matching between S and T induced by σ and τ* , and based on the observed-value representation for properties, is the relation μ_P such that

$$(P, C, Q, D) \in \mu_P \text{ iff } \sigma(o[S, P^C], o[T, Q^D]) \geq \tau$$

For each class C in C_S , let $\text{props}[S, C]$ 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 $\text{props}[S, C]$ the *representation* of C in U_S .

The *contextualized class matching 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 \top is the top class)

$$(C, T, D, \top) \in \mu_C \text{ iff } \sigma(\text{props}[S, C], \text{relprops}[S, C, T, D]) \geq \tau$$

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

From the matchings directly induced by σ and τ , the process then derives a class instance matching and a refined contextualized property matching, as follows.

Figure 9 shows the class instance matching algorithm. It receives as input S and T , and the class matching μ_C induced by σ , τ and μ_P . It also implicitly receives as input U_S and U_T .

```

INSTANCE-MATCHING(S, T,  $\mu_C$ )
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  $U_S \times U_T$ 
      if  $\sigma(t[S, C](I), t[T, D](J)) \geq \tau$ 
        then  $\mu_I = \mu_I \cup (I, C, J, D)$ 

```

Figure 9. The class instance matching algorithm.

It outputs a class instance matching μ_I between class instances in U_S and U_T . In Figure 9, if C is a class in C_S , and I is an instance of C in U_S , then $t[S, C](I)$ 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 tuple (P, C, Q, D) in μ_P , and likewise for $t[T, D](J)$.

For each $(P, C, Q, D) \in \mu_P$ such that $(C, T, D, T) \in \mu_C$, 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 class instance matching of Figure 9). Define $iv[P, C, Q, D] = (s, t)$ such that $s = \{(I, u) / (\exists v)(I, u, v) \in q\}$ and $t = \{(J, v) / (\exists u)(I, u, v) \in q\}$. We call s the *instance-value representation* of P^C in U_S (and likewise for t). This second representation for properties is useful since it helps distinguish properties with similar sets of values, but that refer to distinct instances, matched by μ_I .

Figure 10 shows the refined contextualized property matching algorithm. It has the same input as the algorithm in Figure 9, including the class matching μ_C induced by σ , τ and μ_P , and outputs a contextualized property matching μ_A only between properties whose domains are classes directly or indirectly matched by μ_C . The algorithm uses the maximum of the 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 Figure 10, object properties receive a special treatment, since their representations are sets of URIs that are compared with help of the class instance matching μ_I (computed by the algorithm in Figure 9).

The final vocabulary matching μ is the union of the class

```

CONTEXTUALIZED-PROPERTY-MATCHING(S, T,  $\mu_C$ )
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 =  $\sigma(o[S, P^C], o[T, Q^D])$ 
    if (C matches D)
      then (s, t) = iv[P, C, Q, D]
      Y =  $\sigma(s, t)$ 
    else Y = 0
    if max(X, Y)  $\geq \tau'$ 
      then  $\mu_A = \mu_A \cup (C, P, Q, D)$ 

```

Figure 10. The contextualized property matching algorithm.

matching μ_C induced by σ , τ and μ_P and the contextualized property matching μ_A computed by the algorithm in Figure 10. However, μ may have to be adjusted, by dropping matchings, until it becomes structurally consistent (details omitted for brevity).

V. EXPERIMENTAL RESULTS

We conducted an experiment to assess the performance of the vocabulary matching process of Section 4, using data about products obtained from the Amazon and the eBay databases.

We tested the process with data downloaded from the Web, rather than with the benchmark proposed in Duchateau et al. [5], which does not include instances and is, therefore, unsuitable to test our process.

We first defined a set of terms, which were used to query the databases. From the query results, we extracted the less frequent terms common to both databases. We then used these terms to once more query the databases. This pre-processing 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 16,410 instances from the Amazon database and 99,791 instances from the eBay database.

We adopted as similarity functions the contrast model [8] for property matchings, and the cosine distance with TF/IDF for instance matchings. The experiment lead us to conclude that the contrast model has a better performance when we want to emphasize the difference between two sets of values. This follows because the contrast model has room for calibrating several parameters.

Table III shows sample entries of the vocabulary matching obtained. The headings indicate that **e1** is the context of **v1**, and **e2** that of **v2**. Also, “B” abbreviates classes **eb:Book** and **am:Book**, and similarly for the other classes. The rightmost column of Table III classifies the matchings as: *tp*, for true positive; *fp*, for false positive; and *fn*, for false negative.

TABLE III. AUTOMATICALLY OBTAINED VOCABULARY MATCHING FROM EBAY INTO AMAZON.

eBay	Amazon			
v1	e1	v2	e2	
Books		Books		tp
author	B	author	B	tp
binding	B	biding	B	tp
edition	B	edition	B	tp
format	B	biding	B	tp
isbn-10	B	isbn	B	tp
isbn-13	B	ean	B	tp
editionDesc	B	format	B	fp
Offer		Books		fp
currency	O	currencyCode	B	fp
startPrice	O	listPrice	B	fp

The total number of true positives obtained was 25, that of false positives was 4 and that of false negatives was 10.

Hence, the performance measures were:

$$\textit{precision} = \frac{tp}{tp + fp} = 86\% \quad \textit{recall} = \frac{tp}{tp + fn} = 71\%$$

$$f\textit{Measure} = 2 \frac{\textit{precision} \cdot \textit{recall}}{\textit{precision} + \textit{recall}} = 78\%$$

The highlighted lines in Table III refer to matchings that would have been considered false negatives, if the algorithm in Figure 10 ignored the instance-value representation for properties. In this case, the performance measures would drop to:

$$\textit{precision} = 82\% \quad \textit{recall} = 51\% \quad f\textit{Measure} = 63\%$$

Lastly, if we consider only the class matchings in Table III, the performance measures are:

$$\textit{precision} = 80\% \quad \textit{recall} = 80\% \quad f\textit{Measure} = 80\%$$

VI. CONCLUSIONS

In this paper, we first described a multi-agent mediation architecture that supports the extensional approach. Then, we discussed how to decompose the problem of schema matching into the problems of concept mapping and vocabulary matching.

We concentrated on a vocabulary matching strategy that is instance-based and that uses similarity functions in a non-trivial way. We illustrated the strategy with experiments using data available on the Web.

The results described in the paper admit several extensions, in particular, to more complex OWL schemas. Specially challenging is the problem of generating a new version of the mediated schema as a result of trying to match a new export schema with the current version of the mediated schema.

ACKNOWLEDGMENTS

This work is partly supported by CNPq under grants 142103/2007-1, 301497/2006-0 and 473110/2008-3.

REFERENCES

- [1] A. Bilke and F. Naumann, "Schema matching using duplicates," in *Proc. of the 21st Int'l. Conf. on Data Engineering*, Apr 2005, pp. 69–80.
- [2] D. F. Brauner, M. A. Casanova, and R. L. Milidiú, "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*, vol. 4277, 2006, pp. 46–47.
- [3] D. F. Brauner, A. Gazola, and M. A. Casanova, "Adaptative matching of database web services export schemas," in *Proc. of the 10th Int'l. Conf. on Enterprise Information Systems*, 2008, pp. 49–56.
- [4] D. F. Brauner, C. Intrator, J. C. Freitas, and M. A. Casanova, "An instance-based approach for matching export schemas of geographical database Web services," in *Proc. of the IX Brazilian Symp. on GeoInformatics (GeoInfo)*, 2007, pp. 109–120.
- [5] F. Duchateau, Z. Bellahsène, and E. Hunt, "XBenchMatch: a benchmark for XML schema matching tools," in *Proc. of the 33rd Int'l. Conf. on Very Large Data Bases, Demo Sessions: group I*, Sep 2007, pp. 1318–1321.
- [6] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Groszofand, and M. Dean. (2004, May) SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission. W3C. Available at: <http://www.w3.org/Submission/SWRL>, last access on Dez 2008.
- [7] L. A. P. P. Leme, "Conceptual schema matching based on similarity heuristics," DSc. Thesis (in preparation), Pontificia Universidade Católica do Rio de Janeiro, 2009.
- [8] L. A. P. P. Leme, M. A. Casanova, K. K. Breitman, and A. L. Furtado, "Evaluation of similarity measures and heuristics for simple RDF schema matching," Pontificia Universidade Católica do Rio de Janeiro, Tech. Rep. 44/08, Oct 2008.
- [9] J. Madhavan, P. Bernstein, A. Doan, and A. Halevy, "Corpus-based schema matching," in *Proc. of the 21st Int'l. Conf. on Data Engineering*, Apr 2005, pp. 57–68.
- [10] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu, "Web-scale data integration: You can afford to Pay as You Go," in *Proc. of CIDR*, vol. 7, 2007, pp. 342–350.
- [11] J. Wang, J. Wen, F. Lochovsky, and W. Ma, "Instance-based schema matching for web databases by domain-specific query probing," in *Proc. of the 13th Int'l. Conf. on Very Large Data Bases*, Aug 2004, pp. 408–419.