MatchMaking – A Tool to Match OWL Schemas

Raphael do Vale A. Gomes 1

Luiz André P. Paes Leme 1

Marco A. Casanova 1

Abstract: This paper describes a software tool that implements an instance-based schema matching technique for OWL dialects. The technique is based on a matching algorithm that depends on the definition of similarity functions that evaluate the semantic proximity of elements from two different schemas. The tool is engineered to accommodate different similarity functions and variations of the matching algorithm, thereby facilitating experimentation with alternative setups.

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 a schema matching becomes a challenge when different vocabularies are used to refer to the same real-world concepts [1]. 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" [6].

We describe in this paper a software tool that implements an instance-based schema matching technique first introduced in [3] for OWL dialects. Briefly, the technique is based on a matching algorithm that uses similarity functions to evaluate the semantic proximity of elements from two different schemas. The tool is engineered to accommodate several

[1] Department of Informatics – Pontifical Catholic University of Rio de Janeiro
Rua Marquês de S. Vicente, 225 – Rio de Janeiro, RJ – Brazil CEP 22451-900
{rgomes, lleme, casanova}@inf.puc-rio.br

similarity functions and variations of the matching algorithm, thereby facilitating experimentation with different matching strategies and with data from distinct application domains.

The paper is organized as follows. Section 2 summarizes the major features of the matching technique. Section 3 describes the arquiteture and the major implementation decisions pertaining to the software tool. Finally, Section 4 contains the conclusions.

We refer the reader to [2,3,4,5] for a full discussion, as well as examples, of the concepts used in this paper.

2 The Schema Matching Technique

We assume that the reader is familiar with basic XML and OWL concepts. We will work with OWL schemas that support named classes, datatype and object properties, subclasses, and individuals. Furthermore, the domain of a datatype or object property must be a named class, the range of a datatype property must be an XML schema type, whereas the range of an object property must be a named class.

In what follows, let *S* and *T* be two OWL 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.

We decompose the problem of schema matching into the problems of defining a *contextualized vocabulary matching* and defining a *concept mapping* [4,5].

A *contextualized vocabulary matching* between *S* and *T* is a finite set of quadruples (v_1,e_1,v_2,e_2) such that either $(v_1,v_2)\in \boldsymbol{C_S}\times\boldsymbol{C_T}$ and $e_1=e_2=$Nill, or $(v_1,v_2)\in \boldsymbol{P_S}\times\boldsymbol{P_T}$ and e_1 and e_2 are subclasses of the domains, or the domains themselves, of v_1 and v_2, respectively. Table 1 illustrates a fragment of a contextualized vocabulary matching. The first line indicates that classes `am:Book` and `eb:Book` match, whereas the second line indicates that properties `am:name` and `eb:publisher` match in the context of `am:Publ` and `eb:Book`. Intuitively, matching classes have the same meaning, and so does matching properties, but in their context.

A *concept mapping* from *S* into *T* is a set γ of expressions of a query or rule language that define concepts in *T* in terms of the concepts of *S*.

To detect when two instances denote the same real-world object, we need a third notion. Let $\boldsymbol{U_S}$ and $\boldsymbol{U_T}$ be sets of triples of *S* and *T*, respectively. An *instance matching* from *S* into *T* is a set μ_I of quadruples such that, if $(I,C,J,D)\in\mu_I$, then there are triples $(I,$`rdf:type`$,C)\in \boldsymbol{U_S}$ and $(J,$`rdf:type`$,D)\in \boldsymbol{U_T}$.

We adopt a 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) a class matching; and (b) an instance matching.

(3) Use the class matching and the 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 class matching obtained in Step (2) and the property matching obtained in Step (3), adjusted until it becomes structurally correct.

Step (1) generates preliminary property matchings 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”*. However, to work correctly, Step (2) requires that Step (1) generates preliminary property matchings only for highly similar properties.

Table 1. Example of a vocabulary matching.

Amazon		**eBay**	
`am:Book`	T	`eb:Book`	T
`am:name`	`am:Publ`	`eb:publisher`	`eb:Book`

3 The MachMaking Tool

The MatchMaking tool addresses the problem of vocabulary matching. It takes two schemas and generates a contextualized vocabulary matching. The tool is engineered to accommodate multiple matching setups, and to capture matching provenance data. It features an internal database (see Figure 1) that stores schemas, matching entries, sets of values, representing the matchable elements, and the matching setups. The present version of the tool therefore does not generate concept mappings.

An OWL *schema* (top left of Figure 1) is modeled as an aggregation of *elements*, specialized into *classes*, *properties*, and *instances*. A *matchable* is either an instance, a class or a *contextualized property* (CProperty in Figure 1), that is, a pair consisting of a class and a property.

Each schema is associated with one of more *datasets* (bottom left of Figure 1). Each dataset contains a set of triples, which describe the elements of the schema, including instances of classes and properties. From a dataset, *representations* [3] (Set in Figure 1) for each Matchable of the schema are extracted. Each representation is a set of *values* and has a *type*.

To capture provenance data (top right of Figure 1), the internal database also stores descriptions of matching algorithms, or *matchers*, and of *similarity functions* (Similarity in

Figure 1). To model the fact that a matching algorithm has a series of matching steps, as in Section 2, a matcher is modeled as an aggregation of matchers. Each matcher *applies* one or more similarity functions, and may have a *parameter list*. The matching algorithm described at the end of Section 2 provides the archetypal example of the family of instance-based matching algorithms that the tool supports.

Each *execution* of a *matcher* (bottom right of Figure 1) stores which *parameter values* were defined and what sets of values were used by each similarity function. Each execution results is an aggregation of *Entries*, which therefore model a vocabulary matching.

To run the tool, the user must first register a set of external schemas of data sources and a description of the matching algorithms he wants to experiment with. Then, the user selects a matching algorithm, specifies the applicable parameters, selects a pair of external schemas, and downloads sets of triples (*dataset* objects) from the data sources.

The tool then identifies schema elements, extracts representations for each of these elements and stores them in the internal database. Finally, the tool invokes the desired matching algorithm to find a vocabulary matching between the pair of external schemas.

The user may define a batch script to run the same matching algorithm with different parameters values.

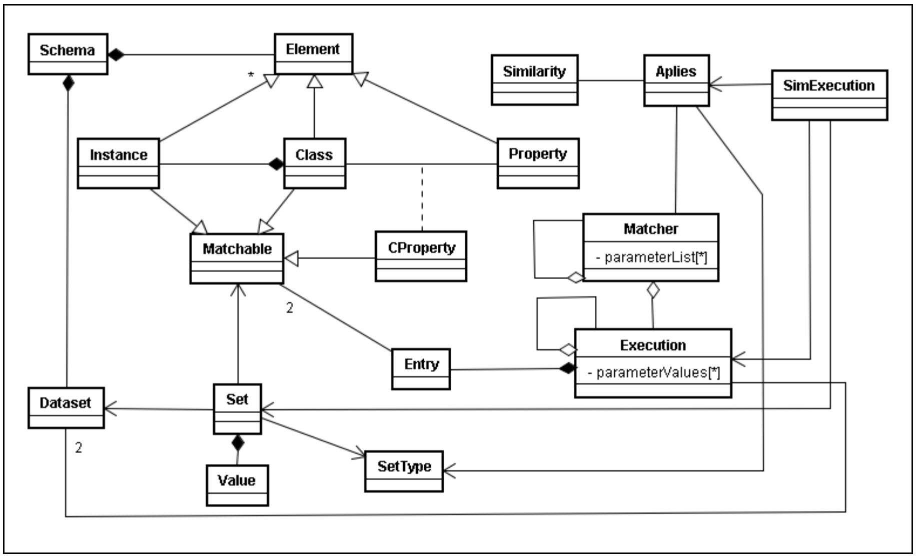

Fig. 1. Diagram of the internal database.

4 Conclusions

In this paper, we described a software tool that implements an instance-based schema matching technique for OWL schemas. The tool is engineered to accommodate several similarity functions and variations of the matching algorithm, described in Section 2, thereby facilitating experimentation with different setups. In particular, all similarity functions described in [2,5] may be registered with the tool and used in matching experiments.

We plan to expand the tool to also cover concept mappings, and to incorporate the tool into a full-fledged mediator equipped with a query processor.

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

References

1. Casanova, M., Breitman, K., Brauner, D., and Marins, A. “Database conceptual schema matching”. Computer, 40(10):102–104.
2. Leme, L. A. P. P. et. al. “Evaluation of similarity measures and heuristics for simple RDF schema matching”. Technical Report 44/08, Dept. Informatics, PUC-Rio.
3. Leme, L.A.P.; Casanova, M.A.; Breitman, K.K; Furtado, A.L. (2009) “Instance-based OWL Schema Matching”. Proc. 11th Int’l. Conf. on Enterprise Inf. Systems, Milan, Italy.
4. Leme, L. A. P. P. at al. , Brauner, D. F., Breitman, K. K., Casanova, M. A., and Gazola, A. “Matching object catalogues”. J. Innovations in Systems and Software Engineering 4(4), Springer, p. 315–328.
5. Leme, L. A. P. P. *Conceptual schema matching based on similarity heuristics*. D.Sc. Thesis, Dept. Informatics, PUC-Rio.
6. Quine, W.V. “Ontological Relativity”. J. of Philosophy, Vol. 65, No. 7, p. 185-212.