An integrative proximity measure for ontology alignment*

Jérôme Euzenat
INRIA Rhône-Alpes
Jerome.Euzenat@inrialpes.fr

Petko Valtchev
Université de Montréal
Petko.Valtchev@umontreal.ca

Abstract

Integrating heterogeneous resources of the web will require finding agreement between the underlying ontologies. A variety of methods from the literature may be used for this task, basically they perform pair-wise comparison of entities from each of the ontologies and select the most similar pairs. We introduce a similarity measure that takes advantage of most of the features of OWL-Lite ontologies and integrates many ontology comparison techniques in a common framework. Moreover, we put forth a computation technique to deal with one-to-many relations and circularities in the similarity definitions.

1 The ontology alignment problem

Like the Web, the semantic Web will necessarily be distributed and heterogeneous. Therefore, the integration of resources found on the semantic Web is a key issue. A standard approach to the resulting problem lies in the use of ontologies for data description. However, the available ontologies could themselves introduce heterogeneity: given two ontologies, the same entity can be given different names in each of them or simply be defined in different ways, whereas both ontologies may express the same knowledge but in different languages.

Semantic interoperability can be grounded in ontology reconciliation. The underlying problem, which we call the "ontology alignment" problem, can be described as follows: given two ontologies each describing a set of discrete entities (which can be classes, properties, rules, predicates, etc.), find the relationships (e.g., equivalence or subsumption) that hold between these entities. Alignment results can be used for various purposes such as displaying the correspondences, transforming one source into another or creating a set of bridge axioms between the ontologies. An overview of alignment methods is presented in §2.

The present paper focuses on automatic and autonomous ontology alignment, although more interactive scenarios may be built on top of the proposed technique (e.g., complete a partial alignment or use the result as a suggestion to the user). It will be also assumed that the ontologies are described within the same knowledge representation language: OWL-Lite (§3).

The language is based on various features: classes and subsumption, properties and type constraints, etc., and the goal of this paper is to define a similarity measure that encompasses all those features (§4.1) while overcoming major alignment problems such as circularities (§4.2) and the presence of external data types. Our approach is based on previous work on object-based knowledge representation similarity which is here adapted to the current web languages. Interested readers are refered to [14] for a detailed discussion of the proposed measure.

2 Alignment methods

There has been important background work that can be used for ontology alignment: in discrete mathematics for matching graphs and trees [7], in databases for reconciling and merging schemas [11], in machine learning for clustering compound objects described in a restricted FOL [1].

Basically, aligning amounts at defining a pair-wise distance between entities (which can be as reduced as an equality predicate) and computing the best match between them,

*This work has been partially supported by grants from the French consulate in Montréal and the Centre Jacques Cartier. We thank an anonymous reviewer for interesting critical remarks that have helped improving this presentation.

i.e., the one that minimizes the total distance (or maximizes a similarity measure). But there are many different ways to compute such a distance. Roughly speaking, they can be classified as (this complements the taxonomy provided in [11] and only consider features found in actual systems):

terminological (T) comparing the labels of the entities; **string-based (TS)** does the terminological matching through string structure dissimilarity (e.g., editing distance); **terminological with lexicons (TL)** does the terminological matching modulo the relationships found in a lexicon (i.e., considering synonym as equivalent and hyponyms as subsumed);

internal structure comparison (I) comparing the internal structure of entities (e.g., the value range or cardinality of their attributes);

external structure comparison (S) comparing the relations of the entities with other entities; **taxonomical structure (ST)** comparing the position of the entities within a taxonomy; **external structure comparison with cycles (SC)** an external structure comparison robust to cycles;

extensional comparison (E) comparing the known extension of entities, i.e. the set of other entities that are attached to them (in general instances of classes);

semantic comparison (M) comparing the interpretations (or more exactly the models of the entities).

Some contributions can be found in Table 1, we only provide some salient points for each of them: [3] matches conceptual graphs using terminological linguistic techniques and comparing superclasses and subclasses. [12] computes the dissimilarity between two taxonomies by comparing for each class the labels of their superclasses and subclasses. FCA-Merge [13] uses formal concept analysis techniques to merge two ontologies sharing the same set of instances while properties of classes are ignored. Anchor-Prompt [10] uses a bounded path comparison algorithm with the originality that anchor points can be provided by the users as a partial alignment. Cupid [8] is a first approach combining many of the other techniques. It aligns acyclic structures taking into account terminology and data types (internal structure) and giving more importance to leaves. [9] creates a graph whose nodes are candidate aligned pairs and arcs are shared properties. Arcs are weighted by their relevance to the nodes and similarity values are propagated through this graph until a fixed point is reached. T-tree [5] infers dependencies between classes (bridges) of different ontologies sharing the same set of instances based only on the "extension" of classes. Semantic similarity is comparable to the work on subsumption in description logics. In addition, a number of other systems use machine learning techniques for finding class similarity from instances [4].

Many of these algorithm use various techniques for finding a alignment, though they still neglect some aspects of the ontology definitions. Moreover, they are not often robust to cycles in definitions, e.g., the fixed-point computation in [9] is not proven to converge. Our goal is to design a measure that integrates all aspects of OWL-Lite and can deal with cyclic definitions.

3 Ontology representation

For that purpose, we will first exhibit a representation for OWL-Lite ontologies (§3.1) that emphasises entities and their relationships (§3.2).

3.1 The web ontology language OWL

OWL [2] is a language for expressing ontologies on the web. Due to space restrictions, we only present here the ontology constructors proposed by the language (the reader can find elsewhere more information on their semantics). OWL can be thought of as a description logic embedded in a frame-like syntax. It comes in three flavors: OWL-Lite, OWL-DL, and OWL-Full. We concentrate on OWL-Lite which is sufficient for many purposes while creating various difficulties for alignment algorithms.

OWL-Lite is an extension of RDF which allows the definition of individuals as instances of a class and the expression of relations between individuals. Additionally[1], OWL-Lite:

- uses RDF Schema keywords (`rdfs:subClassOf`, `rdfs:Property`, `rdfs:subPropertyOf`, `rdfs:range`, `rdfs:domain`) for defining taxonomies of classes and properties and restricting the range of properties;

[1] We do not present all the constructors, some of them can be easily defined from the others. E.g., `owl:sameClassAs` can be defined through reciprocal `rdfs:subClassOf` assertions. Any semantically grounded measure should be able to account for these equivalences.

Reference	T	TS	TL	I	S	ST	SC	E	M
Dieng & Hug [3]	x		x			x			
Staab & Mädche [12]		x			x	x			
FCA-Merge [13]						x			x
Anchor Prompt [10]	x			x	x	x			
Cupid [8]	x	x		x	x				
Similarity flooding [9]		x			x		x		
T-tree [5]					x			x	

Table 1: Various contributions to alignment at a glance.

- allows the definition of a class (`owl:Class`) as more specific or equivalent to the intersection of other classes;
- allows the assertion of equality (`owl:sameAs`) or difference (`owl:differentFrom`) between two individuals;
- characterizes properties as transitive (`owl:TransitiveProperty`), symmetric (`owl:SymmetricProperty`) or inverse of another property (`owl:inverseOf`);
- can restrict the range of a property in a class to be another class (`owl:allValuesFrom`) or assert that some objects of a particular class must be in the property (`owl:someValuesFrom`).
- can restrict the number of object in a particular relation with another one through the use of cardinality constraints (`owl:minCardinality` and `owl:maxCardinality`). In OWL-Lite, these constraints can only take values 0, 1, or infinite.

OWL makes use of external data types. In particular it relies on the XML Schema data types without having to know them.

3.2 Representation

Instead of computing similarity on an OWL-Lite syntax, it will be computed on a corresponding graph based syntax. Such a graph will contain several types of nodes: class (C), object (O), relation (R), property (P), property instance (A), datatype (D), datavalue (V), property restriction labels (L). These nodes are linked by various kinds of relationships:

- `rdfs:subClassOf` between two classes or two properties ($\mathcal{S}$);
- `rdf:type` ($\mathcal{I}$) between objects and classes, property instances and properties, values and datatypes;
- $\mathcal{A}$ between classes and properties, objects and property instances;
- `owl:Restriction` ($\mathcal{R}$) expressing the restriction on a property in a class;
- valuation ($\mathcal{U}$) of a property in an individual

The relation symbols will be used as set-valued fonctions ($\mathcal{F}(x) = \{x; \exists y; \langle x, y\rangle \in \mathcal{F}\}$). Additionaly, each node is identified ($\lambda : C \cup O \cup R \cup P \cup D \cup A \longrightarrow URIRef$) by a URI reference and can be attached annotations.

Finally, to provide the most complete basis for comparison, one may wish to bring knowledge encoded in relation types to the object level. This could be done by adding some edges between objects that are reverse, symmetric or transitive for an existing edge or a pair of edges. Relation types can be handled by saturation of the graph or in a lazy way: for `owl:TransitiveProperty` by adding transitivity arcs; for `owl:SymmetricProperty` by adding symmetric arcs; for `owl:inverseOf` by adding the reverse arcs (both in generic and individual descriptions); for `owl:FunctionnalProperty` by adding a cardinality constraint; `owl:InverseFunctionnalProperty` is not accounted for at that stage.

4 Principles of similarity

Alignment amounts at finding the best correspondance between entities of two ontologies. This requires the definition of a similarity on entity pairs (§4.1). Since relation-

ships between entities constitute a major part of the ontological knowledge, a sensible similarity measure must process them suitably, in particular, by comparing two entities with respect to the sets of "surrounding" entities in the corresponding ontologies. Consequently, relationships entail dependencies between similarity values which further require an effective computation mechanism to avoid the pitfalls of circularity (§4.2).

4.1 Similarity measure

The graphic representation chosen for OWL-Lite highlights the various categories of entities, of links between entities and of descriptive features for entities. The target correspondence between two ontologies maps entities from one ontology to the most similar entities of the other one, a principle that is based on a dedicated similarity measure. We choose to use a similarity measure for ease of explaination. A dual disimilarity can be obtained by an easy transformation. The measure ranks a pair of entities to a real number in $[0\ 1]$ whereby 0 (1) stands for completely different (similar) entities. It is based on two key assumptions:

- all the components of an entity category are *a priori* relevant for similarity assessment, although their relative importance can be tuned through weights. This is backed by most of the techniques used for ontology alignment (see §2);
- the entities within each category are dealt with in the same way, but comparison means for different categories may diverge.

In summary, the approach followed here consists in assigning each entity category, e.g., a class, a specific measure which is defined as a function of the results computed on the related entity categories, e.g., a property, a sub-class, etc., by the respective measures. We choose to aggregate the various components through a weighted sum. Some other aggregation operators could be used but at the expense of the difficulty to find a solution. Weights allow to tune the importance of a component in the similarity whereby a zero weight amounts to completely ignoring the component. E.g., for two classes classes c, c' :

$$\begin{aligned} Sim_C(c,c') = \pi_L^C sim_L(\lambda(c),\lambda(c')) \\ + \pi_O^C MSim_O(\mathcal{I}(c),\mathcal{I}'(c')) \\ + \pi_S^C MSim_C(\mathcal{S}(c),\mathcal{S}'(c')) \\ + \pi_P^C MSim_P(\mathcal{A}(c),\mathcal{A}'(c')) \end{aligned}$$

The similarity is normalised: the sum of all weights is 1, i.e., $\pi_L^C + \pi_S^C + \pi_O^C + \pi_P^C = 1$, whereas set similarities ($MSim$) are basically averages of components similarities, as illustrated by the measure for super-class sets:

$$MSim_C(S,S') = \frac{\sum_{\langle c,c'\rangle \in Pairing(S,S')} Sim_C(c,c')}{max(|S|,|S'|)}$$

Here $Pairing(S,S')$ is a mapping of element of S to elements of S' which maximises the $MSim_C$ similarity. Thus, the similarity between the sets is the average of the values on matched pairs (see definition in [14]). Table 2 lists all the defined measures.

The target similarity values ultimately depend on the similarities between data types, values and URIRef and the way these are propagated through the relationships in the graphs. Measures for data types and values should be provided together with an abstract data type definition, URIRef can be compared by an equality predicate or by a string similarity applied to suffixes.

4.2 Computing similarities

One may notice from the above example that $Sim_C(c,c')$ depends on the result of Sim_C on other classes, both through specialization and properties. In the second case, the dependancy may easily lead to a "deadlock" where $Sim_C(c_1,c_2)$ depends on $Sim_C(c_3,c_4)$ and *vice versa*. Consequently, similarities can only be expressed as equations. More precisely, a system is composed in which a variable corresponds to an entity pair whereas an equation is drawn from the definition of that pair similarity, namely by substituting all similarity occurences by the corresponding variables:

$$\begin{cases} x_{1,1} = Sim_C(c_1,c'_1) & y_{1,1} = Sim_P(p_1,p'_1) \\ x_{1,2} = Sim_C(c_1,c'_2) & y_{1,2} = Sim_P(p_1,p'_2) \\ \ldots \end{cases}$$

Function	Node	Factor	Measure
Sim_O	$o \in O$	$\lambda(o)$	sim_L
		$a \in A$, $(o,a) \in \mathcal{A}$	$MSim_A$
Sim_A	$a \in A$	$r \in R$, $(a,r) \in \mathcal{R}$	Sim_R
		$b \in O \cup V$	$MSim_V / MSim_O$
Sim_V	$v \in V$	value literal	type dependent
Sim_C	$c \in C$	$\lambda(c)$	sim_L
		$p \in P$, $(c,p) \in \mathcal{R}$	$MSim_P$
		$c' \in C$, $(c,c') \in \mathcal{S}$	$MSim_C$
sim_D	$d \in D$	$\lambda(r)$	XML-Schema specific
Sim_R	$r \in R$	$\lambda(r)$	sim_L
		$c \in C$, $(r, \texttt{domain}, c) \in \mathcal{R}$	$MSim_C$
		$c \in C$, $(r, \texttt{range}, c) \in \mathcal{R}$	$MSim_C$
		$d \in D$, $(r, \texttt{range}, d) \in \mathcal{R}$	Sim_D
		$r' \in R$, $(r,r') \in \mathcal{S}$	$MSim_R$
Sim_P	$p \in P$	$r \in R$, $(p,r') \in \mathcal{S}$	Sim_R
		$c \in C$, $(p, \texttt{allValuesFrom}, c) \in \mathcal{R}$	$MSim_C$
		$n \in \{0, 1, +\infty\}$, $(p, \texttt{cardinality}, n) \in \mathcal{R}$	equality

Table 2: Similarity function decompositon.

In case some similarity values (or some similarity or dissimilarity assertions) are provided as an input to the program, the corresponding equation can be replaced by the assertion of the similarity between the objects.

If each of the $MSim$ were deterministic (only one entity is compared to another), this system would be solvable directly because all variables are of degree one. However, in the case of OWL-Lite, the system is not linear since there could be many candidate pairs for the best match. Nevertheless, the resolution of the resulting system can still be carried out as an iterative process that simulates the computation of the fixed point of a vector function, as shown by Bisson [1]. The trick consists in defining an approximation of the $MSim$-measures, solving the system, replacing the approximations by the newly computed solutions and iterating. The first values for these $MSim$-measures are the maximum similarity found for a pair, without considering the dependent part of the equations. The subsequent values are those of the complete similarity formula filled by the solutions of the system. The system converges: the similarities cannot decrease between steps – in an equation, the "ground" part remains steady while dependencies may only propagate their own increase – and the similarity is bounded by 1 – no variable value can exceed 1 since none of its components can (inductively). The process halts when none of the values increases by more than ϵ with respect ot the previous iteration. The algorithm may well converge to a local optimum, i.e., a different matching in one equation may, at least theoretically, lead to a different global solution. A solution could lay in a random change of some matchings.

The result of the process is an approximation of the similarity between entities from opposite ontologies. The ultimate alignment goal is a satisfactory mapping between ontologies which uses the similarity values as a basis for the ranking of entity pairs.

5 Conclusion

In order to be able to align ontologies written in OWL-Lite, we adapted a method developed for measuring object-based similarity to OWL-Lite. This method has the benefit of considering many of the features of ontology descriptions in computing the alignment: it deals successfully with external data types, internal structure of classes as given by their properties and constraints, external structure of classes as

given by their relationships to other classes and the availability of individuals.

This is an improvement towards other methods that take advantage of only a subpart of the language features. The proposed methods does not only compose linearly individual methods for assessing the similarity between entities, it uses an integrated similarity definition that make them interact during computation. Moreover, it copes with the unavoidable circularities that occur within ontologies.

Yet, this measure does not cover all syntactic constructions of OWL-Lite (e.g., `owl:AllDifferent`, `owl:InverseFunctionnalProperty`). A more complete description of this similarity measure is in preparation [6]. We also plan to neatly integrate some features of OWL-DL (e.g., `owl:oneOf`). Moreover, thorough tests of our measure must be performed to find weights and external similarity measures that provide satisfactory results.

The proposed similarity measure is not semantically justified, but exhibits good features such as not imposing injective mapping. However, we would like it to be at least syntax-independent for OWL-Lite. To that extent, we must ensure that whatever the description of two entities, if they are semantically equivalent, they behave identically with respect to the similarity measure. This will amounts to either normalising the graph (adding `owl:minCardinality` constraints for each `owl:someValueFrom` for instance) or comparing heterogeneous components.

References

[1] Gilles Bisson. Learning in FOL with similarity measure. In *Proc. 10th American Association for Artificial Intelligence conference, San-Jose (CA US)*, pages 82–87, 1992.

[2] Mike Dean and Guus Schreiber (eds.). OWL web ontology language: reference. Working draft, W3C, 2003. http://www.w3.org/TR/owl-ref/.

[3] Rose Dieng and Stefan Hug. Comparison of "personal ontologies" represented through conceptual graphs. In *Proc. 13th ECAI, Brighton (UK)*, pages 341–345, 1998.

[4] An-Hai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Ontology matching: A machine learning approach. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies in Information Systems*, pages 397–416. Springer-Verlag, Heildelberg (DE), 2003.

[5] Jérôme Euzenat. Brief overview of T-tree: the Tropes taxonomy building tool. In *Proc. 4th ASIS SIG/CR workshop on classification research, Columbus (OH US)*, pages 69–87, 1994. ftp://ftp.inrialpes.fr/pub/sherpa/publications/euzenat93c.ps.gz.

[6] Jérôme Euzenat and Petko Valtchev. Alignment in OWL-Lite, 2003. in preparation.

[7] John Hopcroft and Robert Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[8] Jayant Madhavan, Philip Bernstein, and Erhard Rahm. Generic schema matching using Cupid. In *Proc. 27th VLDB, Roma (IT)*, pages 48–58, 2001. http://research.microsoft.com/ philbe/CupidVLDB01.pdf.

[9] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: a versatile graph matching algorithm. In *Proc. 18th International Conference on Data Engineering (ICDE), San Jose (CA US)*, 2002.

[10] Natalya Noy and Mark Musen. Anchor-PROMPT: Using non-local context for semantic matching. In *Proc. IJCAI 2001 workshop on ontology and information sharing, Seattle (WA US)*, pages 63–70, 2001. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-47/.

[11] Erhard Rahm and Philip Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[12] Steffen Staab and Alexander Mädche. Measuring similarity between ontologies. *Lecture notes in artificial intelligence*, 2473:251–263, 2002.

[13] Gerd Stumme and Alexander Mädche. FCA-merge: bottom-up merging of ontologies. In *Proc. 17th IJCAI, Seattle (WA US)*, pages 225–230, 2001.

[14] Petko Valtchev. *Construction automatique de taxonomies pour l'aide à la représentation de connaissances par objets*. Thèse d'informatique, Université Grenoble 1, 1999.