JHU/APL Onto-Mapology Results for OAEI 2006

Wayne L. Bethea, Clayton R. Fink, John S. Beecher-Deighan

Research and Technology Development Center
Johns Hopkins University Applied Physics Laboratory
Laurel, MD 20723, USA
Wayne.Bethea@jhuapl.edu

Abstract. Numerous techniques for ontology alignment and mapping have appeared in the literature, but there has been little discussion on the use of formal semantics for the task. Typical solutions apply multiple techniques to produce their results. We demonstrate that a hybrid solution that brings together a number of matching techniques yields the best results. An essential component of any ontology mapping solution is the ability for users to interact with the system and manipulate intermediate and final results. We introduce Onto-Mapology; an approach to ontology mapping that integrates techniques based on string/text matching, structure/graph matching, and semantic (rule-based/logic-based) matching. After the initial design, development, testing and evaluation we applied Onto-Mapology to the OAEI 2006 test cases with encouraging results.

1 Onto-Mapology: The Mapping Process

Ontology mapping techniques have been discussed in the literature that describe string and text matching techniques [1], schema matching techniques [2], categorical information mapping techniques [3], and machine learning techniques [4], but very little has been discussed that describes formal semantic matching techniques. Onto-Mapology is the Johns Hopkins University Applied Physics Lab (JHU/APL) ontology mapping software solution that was designed and developed with strong consideration for human participation in the mapping process. It integrates techniques based on string/text matching, structure/graph matching, and semantic (rule-based/logic-based) matching. It allows users to apply different combinations of these techniques, or a hybrid algorithm that produces solid results in our testing. This paper discusses Onto-Mapology, our approach to the ontology mapping process, and our results for OAEI 2006.

1.1 Purpose, General Statement

We determined at an early stage in the design of our mapping solution that given the state of the art in ontology mapping, human participation would be a crucial part of any successful real world application. This meant that we needed to design user interaction as an important part of the software design rather than an afterthought.

Onto-Mapology was developed as an Eclipse Plug-in, where Eclipse.org is an open source community whose projects are focused on providing an extensible development platform and application frameworks for building software [5]. The Eclipse SDK is a development environment that many software developers and end users are familiar with, and provided us with the user interface and the environment to offer important user interactions.

Upon reviewing the literature it was clear that there was not much discussion on using formal semantics (e.g. using reasoning engines and inference) in the ontology mapping process. It is appropriate to hypothesize that this is due to the fact that much of the semantic meaning expressed in past and present ontologies is expressed through the linguistic content. In contrast, as ontologies mature and users get better at using the tools at their disposal for creating and maintaining ontologies, we will begin to see more semantically expressive ontologies. We wanted to determine what would be needed to successfully utilize formal semantics to help accomplish ontology mapping, and we wanted to implement some semantic matching techniques in Onto-Mapology. We have identified the features of formal semantics expressed in ontologies that will improve the results of ontology mapping dramatically using future ontologies.

In current ontologies, the majority of the information available for communicating semantic meaning, and thus for matching and mapping, is in the text of the ontologies. Many ontology mapping solutions rely predominantly on matching techniques performed on the textual content of ontologies, and that is where we started identifying and implementing our matching techniques.

As ontologies become more structurally sophisticated, or as textual content becomes more degraded, structure matching techniques can play increasingly significant roles in ontology matching. Also, in the literature there is a long tradition of supplementing text matching techniques with structure or graph matching techniques, and several approaches are described in the references provided above and the following [6, 7, 8]. In addition to envisioning ontologies becoming more structurally sophisticated one can envision ontologies becoming more semantically expressive. At present the majority of the ontologies that have been developed or are available in the public domain are not very rich semantically. They rely largely on capturing and conveying meaning through linguistic content. But the vision of the Semantic Web implies that the ontologies will be sufficiently expressive as to allow software agents on the Web to act “intelligently” [9].

Here we have provided the motivation and goals for the design and development of Onto-Mapology; we only had to make some minor adjustments to apply the OAEI 2006 benchmark test cases.

1.2 Specific Techniques Used

The mapping solution integrates techniques based on string matching, structure matching, and semantic matching. As we discuss our matching techniques we are assuming the ontologies are expressed using OWL.

1.2.1 Linguistic Matching Techniques

Onto-Mapology can use an implementation of any string comparison matching algorithm, as long as the implementation can use a provided abstract interface. We implemented the algorithms from the SecondString [10] project by creating wrappers around the SecondString string comparison classes. These classes include the Jaro, Jaro-Winkler, TFIDF, and Monge-Elkan string similarity algorithms. Our testing yielded the Jaro-Winkler algorithm as the best performer of the SecondString classes in our implementation. This algorithm calculates the edit distance between two strings and captures the string similarity using:

$$Jaro(s,t) = \frac{1}{3} \cdot \left(\frac{|s'|}{|s|} + \frac{|t'|}{|t|} + \frac{|s'| - T_{s',t'}}{2|s'|} \right)$$

$$Jaro\text{-}Winkler(s,t) = Jaro(s,t) + \frac{P'}{10} \cdot (1 - Jaro(s,t))$$

For two strings *s* and *t*, let *s'* be the characters in *s* that are "common with" *t*, and let *t'* be the characters in *t* that are "common with" *s* … Let $T_{s',t'}$ measure the number of transpositions of characters in *s'* relative to *t'*. *P* is the length of the longest common prefix of *s* and *t*, and *P' = max* (*P*, 4) [11].

Onto-Mapology also implements an algorithm that uses an *n=2* n-gram comparison, affectionately known as "How to Strike a Match" [12]. It bases the similarity score on a comparison of consecutive letter pairs in the two strings. This approach bases its metric upon the similarity of adjacent letters within a string. It meets the following two criteria; 1) strings with slight discrepancies will be scored as similar; 2) strings that contain the same words but differ in arrangement will be scored as similar. This algorithm captures the string similarity using:

$$similarity(s1,s2) = \frac{2 \times |pairs(s1) \cap pairs(s2)|}{|pairs(s1)| + |pairs(s2)|}$$

For two strings *s1* and *s2* the similarity is twice the number of character pairs that are common to both strings divided by the sum of the number of character pairs in the two strings. When using a single matching technique in Onto-Mapology this algorithm tended to yield the best results on the OAEI 2006 test suite.

Onto-Mapology also uses the Lucene [13] text search engine and indexing tool to create a matcher that compares the terms in two ontologies based on the content of their comments, labels and local names. Lucene is high-performance, scalable, full-featured, open-source, and written in Java. We index one ontology using Lucene, treating each term as a "document" and the term's local name, comment text and label text as the document's content. Lucene removes all stop words from the text and creates an index organized by term. Subsequent ontologies are processed term by term, and each term's local name, comment text and label text are processed using Lucene's string processing capabilities to remove all stop words. The resulting list of words is then used as a search argument against the index created from the first ontology. Lucene is configured to use a letter distance algorithm to score the hits against the index. We treat a high scoring hit as a match between a term in one ontology with a term in another ontology.

1.2.2 Structure Matching Techniques

Onto-Mapology implements an algorithm called "Neighborhood Match" where each ontology is viewed as a graph with nodes and edges, the nodes are classes (or data types) and the edges are properties. For each node in the respective graphs the similarity between nodes (ontology terms) is determined by the number of nodes and edges from each nodes "neighborhood" that match. The neighborhood is determined by specifying how many edges the algorithm should traverse from the starting node. The match is determined by the type of the node or edge or by the text of the node or edge if the user wants to use an algorithm that combines text matching techniques and structure matching techniques.

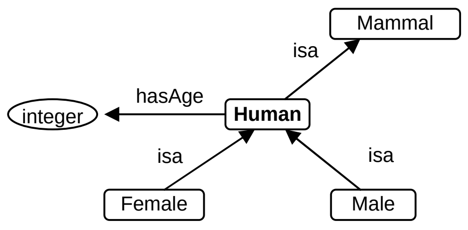

Figure 1: Human Node Neighborhood

So, from a starting node in each of the graphs, the algorithm follows all edges leading from those nodes and compares the edges and related nodes. For example, using the **Human** node in figure 1 the neighborhood 1 edge away would be the *subClass* property relating to **Mammal**, the *subClass* property relating to **Male**, the *subClass* property relating to **Female**, the *hasAge* property relating to **integer**, the classes **Mammal**, **Male**, **Female**, and the data type **integer**. Using the **HomoSapien** node in figure 2 the neighborhood 1 edge away would be the *subClass* property relating to

Mammalian, the *subClass* property relating to **Female**, the *subClass* property relating to **Male**, the *hasAge* property relating to **integer**, the classes **Mammalian**, **Female**, **Male**, and the data type **integer**.

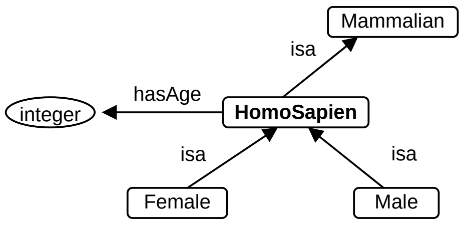

Figure 2: HomoSapien Node Neighborhood

In a purely structural context, our algorithm would compare the 3 subClass properties, 1 data type property, 3 classes, and 1 integer data type neighborhood of the **Human** node to the 3 subClass properties, 1 data type property, 3 classes, and 1 integer data type neighborhood of the **HomoSapien** node and find a match. In a combined linguistic and structural context, our algorithm would also compare the strings of the neighborhoods. For example, it would compare "Mammal," "Male," "Female," and "hasAge" from the Human node with "Mammalian," "Female," "Male," and "hasAge" from the HomoSapien node. In this example, text matching techniques would not produce a match between Human and HomoSapien where structure matching would.

1.2.3 Semantic Matching Techniques

Jena includes a general purpose rule-based reasoner which is used to implement both the RDFS and OWL reasoners but is also available for general use. This reasoner supports rule-based inference over RDF graphs and provides forward chaining, backward chaining and a hybrid execution model. A rule for the rule-based reasoner is defined by a Java Rule object with a list of body terms (premises), a list of head terms (conclusions) and an optional name and optional direction. Each term or **ClauseEntry** is either a triple pattern, an extended triple pattern or a call to a built-in primitive. A rule set is simply a List of Rules.

Onto-Mapology implements rules based on class hierarchy and property hierarchy. For example, we have a rule that states if a class in one ontology is determined to be equivalent to a class in another ontology then the super classes of the equivalent classes are equivalent. The rule looks like this:

```
(?a owl:equivalentClass ?b), notEqual(?a, ?b),
(?a rdfs:subClassOf ?c), (?b rdfs:subClassOf ?d),
notEqual(?c, ?d), notBNode(?c), notBNode(?d) ->
```

```
(?c owl:equivalentClass ?d)
```

We also have a rule that states that if the domain and range of a property in one ontology are determined to be equivalent to the domain and range of a property in another ontology, respectively then the properties are equivalent. The rule looks like this:

```
(?a rdfs:domain ?b), (?c rdfs:domain ?d),
(?b owl:equivalentClass ?d),
(?a rdfs:range ?e), (?c rdfs:range ?f),
(?e owl:equivalentClass ?f) ->
(?a owl:equivalentProperty ?c)
```

Semantic matching through rules doesn't fully access the formal semantics expressed in the ontologies. For sufficiently expressive ontologies an OWL DL reasoning engine should be able to indicate terms that are equivalent and terms that are not equivalent because of the expressed formal semantics. In order for Onto-Mapology to exploit formal semantics expressed in ontologies to assist in ontology alignment we have incorporated Pellet [14], an open-source Java based OWL DL reasoner, into our solution.

1.2.4 Hybrid Algorithm

The Onto-Mapology hybrid algorithm first generates a list of alignments based on name equivalence (100% similarity) using the Jaro-Winkler matching technique. Terms matched in this way are placed into a "high confidence" list. Terms in this list can not be matched again. Next, alignments are created using the lemma matching technique and matched terms are added to the high confidence list. Alignments made in this step do not consist of matches created during the Jaro-Winkler phase. Finally the remaining terms in each ontology are compared based on type. If two terms are the same type then they are compared both structurally and semantically.

Structural comparison is performed as follows: if two terms share 80% equivalent neighborhoods they are judged to be equivalent. Two neighbors are judged to be equivalent if they have been aligned previously or if they share the same type. Semantic equivalence is based upon OWL language relations. We define properties to be equivalent if they have had their domains and ranges aligned. For classes, we state that if two classes share equivalent child class then they are defined to be equivalent. We have completed the task of bringing these techniques together in one algorithm, but we need to add the formal semantic reasoning and characterize which parts of the algorithm will work best under which circumstances. After we have the full implementation and the characterization we can fine tune the algorithm to give the best results given multiple and different types of ontologies.

2 OAEI 2006 Results

Here we present the results of alignment experiments performed on the OAEI 2006 campaign. All the output is produce using the same input parameters. In the presentation of our results and analysis of our algorithms we have also included our experiment results from the OAEI 2005 benchmark tests. The OAEI 2005 based experiment results used linguistic matching techniques to establish alignments based on name similarity. These results were not submitted to the OAEI 2005 campaign because we had not known about the OAEI until after the submission deadline. We will not discuss the OAEI 2005 results or algorithms any further in this paper.

2.1 Benchmark

The benchmark test cases are broken up into five main categories. The first series of tests (#101-104) examine an algorithm's ability to make basic matches. It also determines the program's ability to handle discrepancies of OWL Language usage, like generalization and restriction.

In this first grouping of tests we found our algorithm to be relatively successful in obtaining satisfactory results. However, we found that test #102 created problems for our algorithm. In this test case we compare the reference ontology to one that is irrelevant. The string similarities of the terms in each document are quite different; this leads our structure matching component to become more prevalent thus causing a precision of 0 to occur when any mappings were made. The average performance of this group is depicted below:

	Precision	Recall	F-Measure
Average 2005	0.81	0.99	0.89
Average 2006	0.75	1.00	0.75

The next series of tests (#201-266) manipulate six parameters: name, comments, specialization hierarchy, instances, properties, and classes. These tests allow for algorithms to be examined in specific situations. This set of tests was the most useful to us; they allowed us to see the specific areas where we need improvement.

Tests (#201-210) manipulate names and comments. In this set of test cases our algorithm performed relatively well except in those cases where name similarity was not high (#201, 202, 209, & 210). Even in those cases our recall was still quite high.

	Precision	Recall	F-Measure
Average 2005	0.64	0.28	0.28
Average 2006	0.53	0.96	0.64

Tests (#221-247) manipulate structure. In this set of test cases our algorithm performed very well. This was due to the fact that the terms in these test cases had high string similarity, and in the cases where specific terms did not have similar names or comments, our algorithm was able to use structural or semantic features of each ontology to derive the remaining alignments.

	Precision	Recall	F-Measure
Average 2005	0.75	0.86	0.76
Average 2006	0.99	1.00	0.99

Tests (#248-266) randomize the names and comments while manipulating structure. In this set of test cases our algorithm performed very poorly. Since we rely heavily on string similarity we were unable to extract meaningful results from this section.

	Precision	Recall	F-Measure
Average 2005	0.07	0.00	0.00
Average 2006	0.06	0.58	0.11

The last set of tests (#301-304) use ontologies that are adapted from real life ontologies. Since they were not initially created for the purposes of the OAEI library, they give some insight as to how well each algorithm will perform outside of testing. In this set of test cases the set of terms in either ontology never subsumed the other. This means that there were a number of terms within each ontology that were not meant to be aligned. In addition there were several terms that were synonyms of each other. These two factors led to a heavy reliance on our structure and semantic algorithm components, which lead to poor recall and precision.

	Precision	Recall	F-Measure
Average 2005	0.72	0.51	0.55
Average 2006	0.19	0.61	0.28

3 Comments on Results

As Onto-Mapology demonstrates, our algorithm performed very well when names were highly similar, as did many other solutions in the OAEI 2005. Onto-Mapology was able to derive the terms that did not match lexically, as long as there were enough aligned terms to make those associations, given the semantic and structural aspects of our algorithm. Since we used a combination of methods our weaknesses came into effect when: a) names were random or dissimilar; b) comments were random or dissimilar; c) structures of two disjoint objects were identical; d) semantics of two disjoint objects were similar (e.g. same subclass). The test cases were extremely well conceived. They cover a wide variety of cases and also attempt to isolate specific weaknesses within algorithms. They also include real world ontologies which may give indication of how the algorithm will perform in practice.

4 Conclusion

Onto-Mapology is an ontology mapping solution that is both flexible and interactive. Users can choose from a number of matching techniques and apply a single matching technique or a preconfigured combination of matching techniques. Users may also choose our hybrid matching algorithm that brings together several matching techniques across linguistics, structure and semantics. The results of using the hybrid algorithm are discussed in this paper and we have some work to do to improve the performance. The hybrid solution within Onto-Mapology will perform very well as ontologies become more structurally sophisticated and semantically expressive.

References

1. Milo, T., Zohar, S., Using Schema Matching to Simplify Heterogeneous Data Translation, In Proceedings of the International Conference on Very Large Databases (VLDB), 1998
2. Rahm, E., Bernstein, P. A., A Survey of Approaches to Automatic Schema Matching, the VLDB Journal, 2001
3. Zhou, N., A Study on Automatic Ontology Mapping of Categorical Information, 2002
4. Doan, A., Madhavan, J., Domingos, P., Halevy, A., Learning to Map between Ontologies on the Semantic Web, 2002
5. Eclipse.org Home, http://www.eclipse.org/, 2006
6. Noy, N., Musen, M., PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment, In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI)*, 2000
7. McGuinness, D. L., Fikes, R., Rice, J., Wilder, S., An Environment for Merging and Testing Large Ontologies, In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, 2000
8. Ehrig, M., Sure, Y., Ontology Mapping: An Integrated Approach, Accepted for publication at 1st European Semantic Web Symposium, 2004
9. Fensel, D., Hendler, J., Lieberman, H., Whalster, W., *The Semantic Web: Why, What, and How*, MIT Press, 2001
10. SecondString Project Page, http://secondstring.sourceforge.net/, 2006
11. Cohen, W. W., Ravikumar, P., Fienberg, S. E., A Comparison of String Distance Metrics for Name-Matching Tasks, Carnegie Mellon University, 2003
12. White, S., "How to Strike a Match," *Development Cycles*, 2004
13. Lucene Search Engine, http://lucene.apache.org/java/docs/, 2006
14. Pellet, http://www.mindswap.org/2003/pellet/index.shtml, 2006
15. Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hübner, S., Ontology-Based Integration of Information-A Survey of Existing Approaches, In Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01), 2001
16. De Bruijn, J., et. al., “State-of-the-art Survey on Ontology Merging and Aligning,” Digital Enterprise Research Institute, University Innsbruck, 2004
17. Kalfoglou, Y., Schorlemmer M., Ontology Mapping: The State of the Art, Dagstuhl Seminar Proceedings, Semantic Interoperability and Integration, 2005