Resolving Terminological Heterogeneity In Ontologies

Prasenjit Mitra and Gio Wiederhold [1,2]
{mitra, gio}@db.stanford.edu

Abstract. A system that enables interoperation among information sources using ontologies needs to resolve the terminological differences between ontologies. In this work, we present several methods that we have designed to match terms used in different ontologies. We have implemented two methods based on linguistic similarities of terms used in the ontologies. The first looks up a dictionary or semantic network like WordNet and the second determines similarities of words based on word similarity compuoed from a domain-specific corpus of documents. We discuss our experiments that indicate that a method that uses both heuristics produces good results.

1 Introduction

Often, we cannot answer a query from a single source, and need to compose information from multiple information sources. These information sources are autonomously created and maintained. Integrating the information in them to create a single source is not an option where the owners of the information source prefer to maintain their autonomy. The merging approach of creating an unified source is not scalable and is costly. Besides, an integrated information source would need to be updated as soon as any information in any individual source changes [11]. Furthermore, in certain cases a complete unification of a large number of widely disparate information sources into one monolithic information source is not feasible due to unresolvable inconsistencies between them that are irrelevant to the application. For a particular application, resolution of inconsistencies between a pair of knowledge sources is typically feasible, but it becomes nearly impossible when the objective is undefined and the number of sources is large.

Due to the complexity of achieving and maintaining global semantic integration, the merging approach is not scalable. We have adopted a distributed approach which allows the sources to be updated and maintained independent of each other and enables composition of information via interoperation.

1.1 The Need for Autonomous Ontologies

Ontologies are increasingly being used to assist the integration of information. They specify the terminology (and its semantics) used in information sources. These sources are autonomously created and maintained.

The alternative to individual ontologies for individual sources is to use standard ontologies across multiple information sources. Efforts to create and use standardized ontologies have met with limited success due to the different requirements of the different businesses that construct the information sources. Even if such efforts succeed in creating a standard ontology, the large size of such an ontology results in poor performance while using the ontology.

Everyday new discoveries expand our knowledge and change our views of the universe that we live in. Any ontology representing such knowledge has to be updated periodically. The maintainers of the information sources that use the standard ontology will have to agree on the updates being proposed and on the restructuring of the ontology. They may have entirely different applications in mind or may not subscribe to a newly discovered theory. Furthermore, some participants might see the changes required to support the proposed updates as an unnecessary imposition since restructuring the information source will require substantial effort on their part. Thus generating new consensus on updates to the standard ontology is a time-consuming and tenuous process. For quickly changing fields, arriving at a consensus within a short period of time is not even feasible.

Besides, even if a standard ontology is devised and widely used in future, the ontologies that exist today cannot be wished away. To handle such legacy ontologies, and to allow interoperation among information systems with autonomous ontologies, we need to interoperate among the ontologies themselves.

1.2 Resolving Semantic Heterogeneity

Problems of heterogeneity in hardware, operating systems, and data structures have been widely addressed, but issues of diverse semantics have been handled mainly in an ad-hoc fashion. While composing information from information sources, we need to ensure that the information that we are composing have some semantically meaningful relationship. Semantic heterogeneity among information sources needs to be resolved to enable meaningful information exchange or interoperation among them.

The two major sources of heterogeneity among the sources are as follows: First, different sources use different data formats and modeling languages to represent their data and meta-data. Second, sources using the same data format differ in their structure and semantics of the terminology they use. Such heterogeneity are a result of the autonomous nature of the ontologies and the fact that information sources are constructed by different people with different objectives in mind.

Often different sources use different terminologies to describe the objects in the sources. The same term, used in different sources, often have overlapping or somewhat different semantics, e.g., the term ′nail" has entirely different semantics in a ′cosmetics" ontology and the ′carpentry" ontology. Similarly, different sources, often, use different terms to refer to semantically similar objects, e.g., the terms "truck" and "lorry" in two transportation ontologies might refer to the same class of objects.

[1] Infolab, Stanford University, Stanford, CA 94305, USA
[2] This work has been supported by the AFOSR New World Vistas program and the DARPA DAML program.

In order to enable interoperation, we intend to capture the semantic bridges between two ontologies using *articulation rules*. These rules express the relationship between two (or more) concepts belonging to the ontologies that we seek to interoperate. Since these ontologies can be fairly large, establishing such rules manually is a very expensive and laborious task. Fully automating the process is also not feasible. First, despite the rapid advances made in the field of natural language processing, the technology still remains inadequate to automatically resolve semantic heterogeneity among these information sources using different terminology. Second, even though ontologies expose some of the semantics of the terms and their relationships, they often remain incomplete or inadequate if we consider the needs of the various applications that use them.

The problem of ontology alignment has been studied for some time. Tools like OntoMorph [4], PROMPT [10], and Chimaera [8] help significantly automate the process. However, these tools do not contain a component that identifies concept names that are linguistically similar automatically and use that knowledge as the basis for furhter alignment of the ontologies. They require manual construction of articulation rules or base their matches on the structure of the ontologies. A similar problem is that of schema matching in databases. However, most of the techniques used in matching tools [14],[7], [5], [9], [12], [3] etc. are not adequate when the primary differences among sources are due to differences in terminology in sources with little structural similarity or when instance data is not available.

In this paper, we propose a semi-automated algorithm for resolving the terminological heterogeneity among the ontologies and establishing the articulation rules necessary for meaningful interoperation. This algorithm forms the basis of the *articulation generator* for our ONtology compositION sytesm (ONION). Our experiments show that basing such matching on structural information is inadequate. We describe several heuristics to resolve the terminological heterogeneity among ontologies. Experimental results show that combining the information obtained by using multiple heuristics provides a better match between semantically related terms in the ontologies.

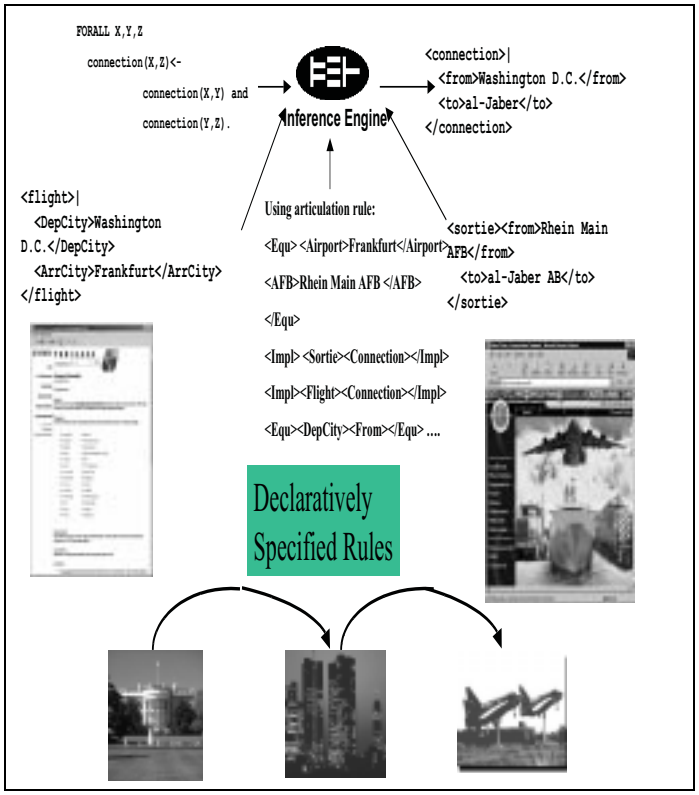

Figure 1. An application using an articulation between the United Airlines Ontology and the TRANSCOM Ontology

2 Ontologies and Their Articulations

In this work, we assume that the ontologies we use are represented as a graph along with a set of logical rules. Formally, an ontology $O = (G, R)$ is represented as a directed labeled graph G and a set of rules R. The graph $G = (V, E)$ comprises a finite set of nodes V and a finite set of edges E. The label of a node is given by a non-null string that is often a noun-phrase that represents a concept name. The label of an edge is the name of a semantic relationship among the concepts and can be null if the relationship is not known. The label of an edge can be any user-defined relationship. The set of relationships with pre-defined semantics is $\{SubClassOf, PartOf, AttributeOf, InstanceOf, ValueOf\}$. All other relationships are not interpreted by the articulation generator in ONION.

Articulation rules are of two types - ones that are simple statements of the form $(Match"DepartmentofDefence""DefenseMinistry")$ expressing matches between equivalent concepts and the more complex rules expressed in datalog that are mostly supplied by the expert.

In Figure 1, we show an example articulation. On the left hand side, is a portion of the United Airlines Ontology and on the right a portion of the TRANSOM Ontology. These ontologies were constructed manually for experimentation. The objective of the application is to transport military men and materiel from Washington D.C. to Al Jabar Airbase in Kuwait. A combination of commercial flights and special purpose sorties is to be used to meet the transport objective.

The United Airlines source has $flight$, whose $DepCity$ is $Washington\ D.\ C.$ and $Arrity$ is $Frankfurt$. This corresponds to a flight from Washington D. C. to Frankfurt. There exists an articulation rule, supplied by the domain expert, that says that the connection relation is transitive.

The TRANSCOM source has a $sortie$ that runs from $RheinMainAFB$ in Frankfurt, Germany to $AlJabar$ airbase in Kuwait.

We establish the articulation rules semi-automatically. They indicate that the $Frankfurt$ airport is the co-located with the $RheinMainAFB$. It tells us that if there is a $sortie$ or a $flight$ between two cities, then there is a $connection$ between them. It also indicates that $DepCity$ in the United ontology is the same as $From$ in the TRANSCOM ontology. Due to lack of space in the figure, the rule that states that $United.ArrCity$ is equivalent to $TRANSCOM.To$ is not shown.

Using these rules, an inference engine can easily establish that there is a connection between $Washington\ D.\ C.$ and $AlJabarAirbase, Kuwait$.

The tool generates and suggests the simpler articulation rules to indicate the terms in the two ontologies that are related. The expert then validates these suggestions and the final set of articulation rules are stored to be used during query rewriting and execution.

3 Generation of Ontology Articulations

ONION has an automated articulation generator (ArtGen) that suggests articulations based on a library of heuristic matchers. Each matcher matches terms in the two ontologies. A human expert, knowledgeable about the semantics of concepts in both ontologies, validates the suggested matches generated by ArtGen using a GUI tool. The expert can either accept the match, keep the match but modify the suggested relationship between the matched terms, delete a suggested match or say that the match is irrelevant for the application at hand. The expert can also indicate new matches that the articulation generator might have missed.

The process of constructing an articulation is an iterative process and after the expert is satisfied with the rules generated, they are stored and used when information needs to be composed from the two ontologies. The response of the expert is also logged and the articulation generator uses the expert's feedback to generate better articulations in future while articulating similar ontologies for similar applications. This learning process improves the quality of future generation of articulations from similar information sources.

The heuristic matchers used by the automated articulation generator can be classified into two broad types - iterative and non-iterative. Since the articulation generator is modular in nature, any application-specific matching algorithm can be plugged in. However, we believe that a set of basic matching algorithms will be useful in a wide variety of applications and we experimented to determine such a set.

3.1 Non-iterative Algorithms

Non-iterative algorithms are ones that identify the matching concepts in the two ontologies in one pass. Our linguistic matcher employs only non-iterative algorithms.

3.1.1 Linguistic Matching

The linguistic matcher looks at all possible pairs of terms from the two ontologies it is matching and assigns a similarity score to each pair. If the similarity score is above a threshold, then the match is accepted and an articulation rule is generated. The threshold can be modified by the expert performing the articulation to increase or decrease the number of matches generated.

We expect that a concept name is represented as a string of words. The matcher constructs all possible pairs of words where the two words in a pair come from different strings. The matcher uses a word-similarity table generated by a *word relator* which we describe below. It looks up a word-similarity table to determine the similarity between all such pairs of words. Finally, it computes the similarity of the strings based on the similarity of the pairs of words.

- match(String s1, String s2, WordSimilarityTable wst)
  - List similarityList;
  - for each word w1 in s1:
    - for each word w2 in s2:
      similarityScore ← wst.lookup(w1, w2);
      Add (w1, w2, similarityScore) to similarityList;
  - Sort similarityList on the similarity score of the tuples;
  - Set matchedWords ← null;
  - floatingPointNumber matchingScore ← 0.0;
  - for each tuple (w1, w2, ss) in similarityList:
    - if either w1 or w2 is in matchedWords continue;
    - else
      matchingScore ← matchingScore + ss;
      add w1, and w2 to matchedWords;
  - similarityScore ← similarityScore / min(size(s1), size(s2));
  - return similarityScore;

For example, given the strings "Department of Defence" and "Defense Ministry", we see that $match(Defence, Defense) = 1.0$. Similarly, we have $match(Department, Ministry) = 0.4$. Therefore, we calculate the similarity between the two strings as:

```
match("Department of Defence",
      "Defense Ministry") = (1 + 0.4)/2 = 0.7.
```

The denominator is the number of words in the string with less number of words.

This similarity score of two strings is then normalized with respect to the highest generated score in the application. The normalization step removes the bias of word-relators that give very low similarity scores for all pairs of words or those that give very high scores to all pairs of words. If the generated similarity score is above the threshold, then the two concepts are said to match, and we generate an articulation rule, $(Match"Departmentof Defence""DefenseMinistry"), 0.7$, the last number gives the confidence measure with which the articulation generator generated this match. The confidence measure varies betwen 0 and 1.

Constructing the Word-Similarity Table:
We have experimented with several ways to generate the table containing the similarity between all pairs of words. After checking if the words are spelt similarly, we derive word similarity using methods that can be differented into two main groups: a) thesaurus based, b) corpus-based.

Thesaurus-Based Word-Relator: We have devised matching algorithms based on dictionaries or semantic networks, like Nexus [6] and WordNet [1]. WordNet gives us a list of synonyms for each word. If the two words are found to be synonyms, then we return a similarity score of 1.0. If the two words are not synonyms, we look at the the number of words that are "similar" in the defnitions of each word. This process of looking into the definitions of words to find their similarity can be repeated recursively until a fixed-point is reached or uptil a specified depth is reached at which point we require "similar" to be "same".

- GenerateSimilarity(word w1,word w2,dictionary dict,depth dep)
  - if (w1 == w2) return 1;
  - if (dep == 0) return 0;
  - else
    - def1 ← dict.lookup(w1);
    - def2 ← dict.lookup(w2);
    - List similarityList ← new List;
    - for each word wd1 in def1:
      - for each word wd2 in def2:
        Add (w1, w2, GenerateSimilarity(wd1, wd2, dep-1))
        to similarityList;
    - Sort similarityList on the similarity score of the tuples;
    - Set matchedWords ← null;

* floatingPointNumber matchingScore ← 0.0;
* for each tuple (w1, w2, ss) in similarityList:
  - if either w1 or w2 is in matchedWords continue;
  - else
    matchingScore ← matchingScore + ss;
    add w1, and w2 to matchedWords;
* similarityScore ← similarityScore / min(size(def1), size(def2));
* return similarityScore;

For example, the definitions of "truck" and "boat" are "an automotive vehicle suitable for hauling",and "a vessel for water transportation". If the specified depth is 1, we do not look into the definitions of "vehicle" and "vessel" to determine their similarity. Since they are not exactly the same, we say their similarity is 0. If however, the depth were set to 2 (or more), we would look up the definitions of "vehicle" and "vessel", discover their definitions both have "transportation" in common, and generate a similarity measure and propagate that similarity up to generate a non-zero similarity for "truck" and "boat".

Corpus-Based Word Relator: Word similarities used by the linguistic matcher can also be generated using a corpus-based matching algorithm. The word relator uses a corpus of documents belonging to the domain of the ontologies that are being matched. The terms that appear in the ontology should also appear in the documents. The word relator calculates word-similarity scores based on the similarity of the contexts in which the words appear in the documents [13].

We identify the context in which a word, w, appears by looking at words that appear in a 1000-character neighbourhood of all occurrences of w in documents in the corpus. For example, the words "in", "the", "For", and "example" constitute the 30-character neighbourhood of the word "corpus" at the end of the last sentence. In the example, we looked at a 15-character window ahead of the word and 15 characters behind the word and chose all words that are complete in these windows. Therefore, even though part of the word "documents" appears in the 15-character window before the word "corpus" in that sentence it is ignored.

We look at all words that appear in the corpus. For each occurrence of a word, we identify the words in its context. The number of rows in the context vector, V_w, of a word w is equal the number of words in the corpus. Let $V_w[i] = c$. This implies that the i^{th} word in the corpus occurs with a frequency c in the 1000-character neighbourhood of the word w. The cosine of such normalized context vectors of two words gives a measure of the similarity of contexts in which the two words appear. We use this similarity measure to generate a table of word similarities that is then used by the linguistic matcher.

Ideally, we would have one corpus associated with one ontology, where the documents in the corpus use the terms in the exact sense as it is used in the ontology. However, for our experiments we did not have such a domain-specific corpus. We generated a corpus by searching the web(google) using 5 keywords each from the two ontologies that we were seeking to articulate. Typically, a corpus of 200 pages proved adequate to produce good matches.

3.1.2 Instance-based Heuristics

Instance-based matching heuristics have been used to successfuly match schemas in databases [14]. Such matchers look at data types, and extract other features like lengths of attributes, numerical or lexical statistics of attributes, and match classes based on such feature vectors. Though, we can handle ontologies, whose concepts also have instances associated with them, oftentimes, businesses are reluctant to make instances available. Thus, we have designed our algorithms assuming that no instance data is available. However, if such information is available, the matcher can be extended to use instance information.

3.2 Iterative Algorithms

Iterative algorithms are algorithms that depend upon existing articulation rules to generate further articulation rules. They require multiple iterations over the two source ontologies in order to generate semantic matches between them.

3.2.1 Structure-based Heuristics

These algorithms look for structural isomorphism between subgraphs of the ontologies to find matching concepts. For the ontologies we have experimented with, we see that a purely structural matcher - one that simply looks for isomorphism between subgraphs in the ontologies without considering concept names- performs very poorly and is inadequate.

Therefore, we propose a structure-based matcher that is called after the matches generated by a linguistic matcher is available. If the linguistic matcher has matched nodes, "A" and "B" in the ontology-graphs, the structural matcher looks to match their children (also parents), "C", and "D", if they have not already been matched. If a substantial percentage (above the threshold supplied) of the parents of "C" have been matched with those of "D", and the children of "C" have been matched with those of "D", then an articulation rule matching "C" and "D" is generated.

3.2.2 Inference-based Heuristics

An inference engine can reason with the rules available with the ontologies and any seed rules provided by an expert ontologies to generate matches between the ontologies. For example, a rule:

```
(=> (InstanceOf X O1.LuxuryCar)
    ((InstanceOf X O2.Car) AND
     (O2.PriceOf Y X) AND
     (O2.UnitOf X "$") AND
     (ValueOf X Z) AND
     (> Z 40,000)))
```

which says that any instance of $O1.LuxuryCar$ is an instance of $O2.Car$, that has a price greater than $\$40,000$.

4 Experiments & Results

We have implemented the linguistic methods and the structural methods in our articulation generator (using Java as the programming language). We experimented with three sets of ontologies represented in RDF[2]:

1. Ontologies (avg. 30 nodes) constructed manually to represent a domestic airlines (terminology used on United Airlines website) and a airforce ontology (terminology used in the US Air Force).
2. Ontologies (avg. 50 nodes) constructed manually from the NATO government web-sites representing each web-page associated with an department of the government as a node. The edges in the ontology graph were derived from the links between the pages.

We measured the accuracy of the generated match by comparing the results generated by the automated matcher with those expected by the expert. Any match deleted by the expert was taken to be a false positive and lowered the precision figures, and a match added by the expert that the automated generator failed to find lowered the recall. We summarize the results of the several experiments below:

- A purely structural method which requires exact concept-name match, like that has been used in existing tools, fails to generate even 50% of the matches expected by the expert. This result is not surprising since despite having useful information, the structure of the ontologies used hardly encode sufficient semantics to use them solely for ontology alignment.
- Adding linguistic heuristics gave significantly better results, especially, the corpus-based heuristic provided we supplied the matcher with a good representative set corpus of documents from the applicable domain.
- However, a multi-strategy approach works best. On the average about 75% of the matches were generated, with less than 5% false positives that the expert indicated was not correct. The linguistic method generates on the average about 60-70% of the matches (recall with 95% precision). Adding the structural matcher, boosts the matches by 5-10%. The human expert provided us with the other 30% of the rules that were not generated automatically.

The performance of the algorithm depends upon several parameters:

- Thesaurus-based Method: A general purpose thesaurus results in very poor results. Domain-specific thesauri produce better results but might not be available.
- Corpus-based Method: A corpus-based method produced better results than the thesaurus-based method. In the aircraft example, solely employing the thesaurus-based method produced a 30% recall (at 90% precision). A corpus-based method, where we obtained a corpus by searching the web with a few key-words from the domains, boosted the match to 60%. Combining the two, we obtained a recall of 70%.
- Scalability: Initially, we tried the corpus-based method with a preprocessing step of collecting the corpus and building up the word-context vectors. The linguistic matcher, while matching the ontologies, constructed the word similarities as needed. However, for a test case with 300 nodes in each ontology took an hour to run on a Pentium III machine with 256M memory. It becomes clear that for larger ontologies, the algorithm does not scale well if we compute the word similarities while matching the ontologies. For the algorithm to scale, not only, do we need to build the corpus and construct the word-context vectors a priori, but also pre-compute the similarity of all pairs of words in the corpus. The corpus-based method can then be thought of as equivalent to a lookup based method, where the word-similarity matrix is constructed from the words in the corpus. This variation of the corpus-based method scaled well and for our ontologies finished within a couple of minutes at worst.
- Quality: The quality of the matches were very dependent on the quality of the corpus available. We experimented with corpuses of size 50 pages, 100 pages, 200 pages and 1000 pages. Corpuses of size 50-100 pages resulted in low recall figures for the matches. A size of 200 webpages often proved adequate to generate a recall of 70%, although in most cases having a corpus of 1000 matches increased the recall, it was less than a few percentages.

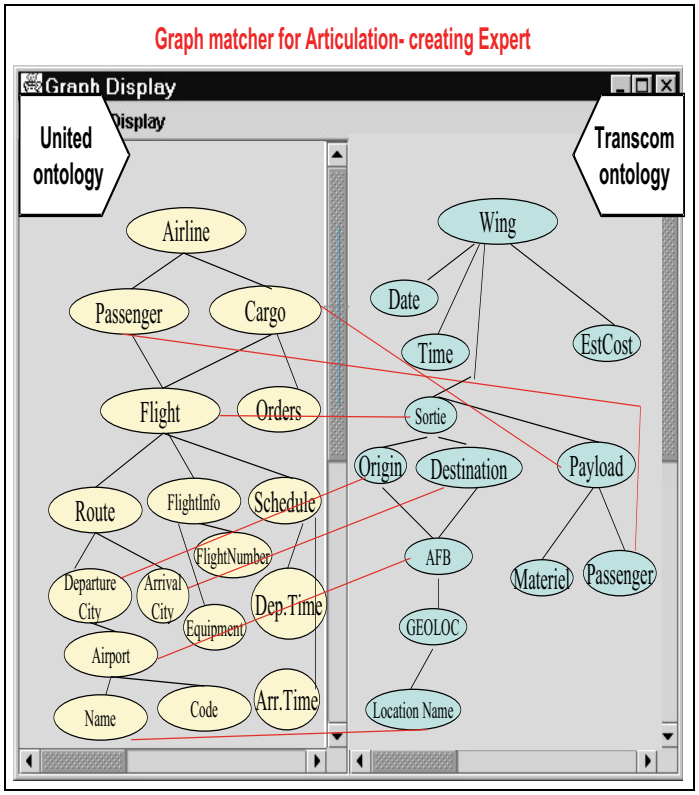

Figure 2. Example of an articulation of United Airlines Ontology with TRANSCOM Ontology

In Figure 2, we show two ontologies - the United Ontology and the TRANSCOM Ontology and the matches generated. We used a hybrid method that uses WordNet as a thesaurus, and a corpus generated by searching google. For example, the page "http://www.etrackcargo.com/Help/Agents/Fieldwas part of the corpus. The confidence scores of the matches are as follows when the threshold was set to 0.7:

Table 1. The matches between the United and TRANSCOM Ontologies

Term in United.ont	Term in TRANSCOM.ont	Confidence Score
Passenger	Passenger	1.0
Cargo	Payload	1.0
Departure Time	Time	0.90
Arrival Time	Time	0.88
Arrival City	Destination	0.79
Name	Location Name	0.75
Departure City	Origin	0.72
Airport	Airforce Base	0.71
Flight	Sortie	0.70

If the threshold was set to a lower value 0.60, we introduced false positives like $(Match\ Airline\ Destination\ 0.61)$. Further lowering the threshold to 0.50 introduces more false matches $(Match\ FlightNumber\ Sortie\ 0.52)$, $(Match\ Equipment\ Materiel\ 0.54)$. Only the first two matches were generated using a word-relator that consults WordNet. We ran the word-relator with a depth value of 1. That is, the relator looks into the definition of the two words for similar words but does not proceed any further recursively. The match between $Cargo$ and $Payload$ was not higher than 0.7 using the corpus-based

word-relator and would not have been suggested. Thus, we see that a hybrid method gives us a better accuracy than any one method alone.

In this example, we see that with a threshold value of 0.7, we generate all the desired matches and no false matches - the ideal solution. However, acheiving a 100all cases. From our experiments, we see that setting a threshold of 0.7 gives the most number of matches with the a 95the matches are false positives. Therefore, we suggest that for an unknown application or an unknown corpus, when running the first time, the matching threshold be set to 0.7. If not satisfied with the results the expert can then increase or decrease the threshold to get better matches.

5 Conclusion

We discussed several heuristic methods to produce simple matching rules between concepts in ontologies that are being aligned. We see that a multi-strategy method based on intial linguistic-similarity followed by structural matching generates matches between ontologies with reliable accuracy. The work of an expert who then validates the suggested rules or supplies new rules is substantially reduced by the automated component.

REFERENCES

[1] 'Wordnet - a lexical database for english. http://www.cogsci.princeton.edu/ wn/', Technical report, Princeton University.

[2] 'Resource description framework(rdf) model and syntax specification, w3c recommendation http://www.w3.org/tr/rec-rdf-syntax', (1999).

[3] M. D. Siegel C. H. Goh, S. E. Madnick. Semantic interoperability through context interchange: Representing and reasoning about data conflicts in heterogeneous and autonomous systems http://citeseer.nj.nec.com/191060.html.

[4] H. Chalupsky, 'Ontomorph: A translation system for symbolic knowledge', in *KR 2000*, pp. 471–482. Morgan Kaufmann Publishers, (Apr 2000).

[5] A. Doan, P. Domingos, and A. Y. Halevy, 'Reconciling schemas of disparate data sources: A machine-learning approach', in *SIGMOD 2002*, (2001).

[6] J. Jannink, *A Word Nexus for Systematic Interoperation of Semantically Heterogeneous Data Sources*, Ph.D. dissertation, Stanford University, 2000.

[7] J. Madhavan, P. A. Bernstein, and E. Rahm, 'Generic schema matching with cupid', in *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pp. 49–58. Morgan Kaufmann, (2001).

[8] D.L. McGuiness, R.Fikes, J. Rice, and S. Wilder., 'The chimaera ontology environment', in *Seventh National Conference on Artificial Intelligence (AAAI-2000)*, (2000).

[9] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm, in *Proceedings of the Twelfth International Conference on Data Engineering, San Jose, CA*. IEEE Computer Society, (February 2002).

[10] N.F. Noy and M.A. Musen, 'Prompt: Algorithm and tool for automated ontology mergin and alignment', in *Seventh National Conference on Artificial Intelligence (AAAI-2000)*, (2000).

[11] D.E. Oliver, Y. Shahar, E.H. Shortliffe, and M.A. Musen, 'Representation of change on controlled medical terminologies', in *Proc. AMIA Conference*, (Oct. 1998).

[12] Yannis Papakonstantinou, Hector Garcia-Molina, and Jeffrey D. Ullman, 'Medmaker: A mediation system based on declarative specifications', in *Proceedings of the Twelfth International Conference on Data Engineering, February 26 - March 1, 1996, New Orleans, Louisiana*, ed., Stanley Y. W. Su, pp. 132–141. IEEE Computer Society, (1996).

[13] Hinrich Schuetze, 'Dimensions of meaning', in *Supercomputing*, pp. 787–796, (1992).

[14] L. L. Yan, R. J. Miller, L. M. Haas, and R. Fagin, 'Data-driven understanding and refinement of schema mappings', in *ACM SIGMOD*, (2001).