Lexicon based Algorithm for Domain Ontology Merging and Alignment

Tomasz Boiński and Henryk Krawczyk
Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology,
11/12 Gabriela Narutowicza Street, 80-233 Gdańsk-Wrzeszcz, Poland

Keywords: Ontology Matching, Algorithm.

Abstract: More and more systems contain some kind of knowledge describing their field of operation. Such knowledge in many cases is stored as an ontology. A need arises for ability to quickly match those ontologies to enable interoperability of such systems. The paper presents a lexicon based algorithm for merging and aligning of OWL ontologies. The proposed similarity levels are being presented and the proposed algorithm is being described. Results of test showing the algorithm quality are presented.

1 INTRODUCTION

Merging and aligning of domain ontologies is a complex process. A set of predefined procedures is needed for proper integration (Fig. 1).

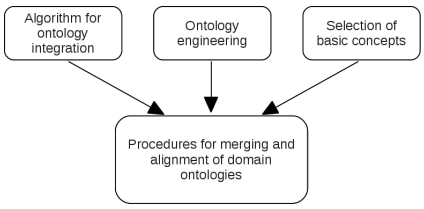

Figure 1: Procedures ensuring ontology integration.

Quality of ontology integration is influenced by three factors:

1. Quality and integrity of input ontologies – a key factor for proper ontology integration. Omitting typical mistakes (Goczyła, 2011) or basing ontology construction on well known guidelines (Gómez-Pérez et al., 2002) allows creation of consistent and flexible ontologies. This way integration introduces some new knowledge in the output ontology.
2. Usage of precise and well formed vocabulary, especially proper selection of core concepts (Boiński, 2012) determines possibility of performing integration seamlessly.
3. Methodology used during input ontology development – usage of one of the well known and accepted methodologies like Methontology (Fernandez et al., 1997), NeOn (NeOn Project, 2010) or methodology described by Noy and McGuiness (Noy et al., 2001) improves quality of the ontology (Boiński, 2012). All those methodologies take into account the need of future integration, which combined with usage of tools like Protg (Stanford University School of Medicine, 2010; Gennari et al., 2003; Noy et al., 2000) or OCS (Boiński, 2012), allows creation of consistent and formally correct ontologies.

With above criteria satisfied creation of lexicon based algorithm for ontology integration becomes possible.

In the following chapters of this paper proposed similarity measures and the algorithm itself will be presented. Some results showing it's correctness will be presented.

2 LEVELS OF SIMILARITY BETWEEN ONTOLOGY ELEMENTS

Integration of knowledge represented by two ontologies can be based on ability to compare elements, i.e. classes, individuals, relations and properties, of those ontologies. A need for measures of semantic and syntactic similarity between concepts thus arises. Those measures will lay fundamentals for the algorithm described in the next chapter.

Measures proposed in this paper are derived directly from pragmatic approach to ontologies represented by both Hovy (Hovy, 1998) and Euzenat

Boiński T. and Krawczyk H..
Lexicon based Algorithm for Domain Ontology Merging and Alignment.
DOI: 10.5220/0004092903210326
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development* (KEOD-2012), pages 321-326
ISBN: 978-989-8565-30-3
Copyright © 2012 SCITEPRESS (Science and Technology Publications, Lda.)

and Volchev (Euzenat and Valtchev, 2004). Proposed solution extends them with possibilities introduced by modern lexicons like WordNet (Fellbaum, 1998). Four mutually complementary and supplementary levels of similarity are proposed:

1. Lexical similarity P_{lex} of classes $\mathcal{K}_i(i = 1,2)$ and individuals $\mathcal{B}_i(i = 1,2)$

   The basic measure of similarity between concepts. Classes and individuals are being looked up in WordNet dictionary. Their similarity and mutual relation is being derived from WordNet structure. Basing on this measure one can determine whether concepts are identical, disjoint or logically encapsulating one another, meaning one can determine class hierarchy and membership of individuals. WordNet dictionary allows both direct string matching and synonyms look-up thanks to organisation of lemmas into so called synsets.

2. Semantic similarity P_{sem} of classes $\mathcal{K}_i(i = 1,2)$ and individuals $\mathcal{B}_i(i = 1,2)$

   When lexical similarity is not possible to calculate, the proposed algorithm uses semantic similarity as a secondary measure for similarity calculations. This level of similarity takes values from range $< 0;1 >$ and allows to determine whether concepts are similar, disjoint or logically encapsulating one another. As a base for this measure WordNet based Lin algorithm (Lin, 1998; Lin, 1993) is being used.

   Lin algorithm requires lemmas describing the concepts to be present in WordNet dictionary which not always is true. In that case Levenshtein edit distance (Levenshtein, 1966) is being used as secondary way of calculating similarity. In that case only similarity and disjointness of concepts can be calculated.

   To improve performance some strings, containing no information like pronouns and words shorter than three letters, are being eliminated from the lemma of the concept. WordNet contains only 305 noun entries of length two and less (0,26% of all nouns contained in the lexicon), moreover most of them are abbreviations and numbers in roman system.

   The border value the similarity that separates similar and disjoint concepts was based on results obtained from comparing human judgement (sim_h) with values obtained from Lin algorithm (sim_{Lin}) (Boiński, 2012).

   Pairs of high sim_{Lin} (*car* - *automobile*, *gem* - *jewel*, *journey* - *voyage*, *boy* - *lad*, *coast* - *shore*, *asylum* - *madhouse*, *magician* - *wizard*, *midday* - *noon* and *furnace* - *stove*) were all found very similar by humans. Similarity sim_h of pairs *tool* - *implement* and *brother* - *monk*, were lower than expected when compared to sim_{Lin}, however humans closely connected with English language (Northern Ireland resident, English teacher etc.) marked those pairs as highly similar.

   The most problematic were pairs *bird* - *cock* and *bird* - *crane*. In both cases second lemma is logically encapsulated by first lemma. According to the Lin algorithm both pairs have relatively high sim_{Lin}, 0.74 and 0.72 respectively. It is unfortunately not possible to determine which concepts encapsulates which basing only on that value. Most of the human testers decided however that in case when only a similarity value is available such concepts can be safely merged if no other piece of information is available.

   Remaining concept pairs were judged different by both humans and the Lin algorithm. Basing on that test it was decided, that $P_{sem} < 0,7$ means different concepts and $P_{sem} \geq 0,7$ means similar concepts. This observation is consistent with results obtained by other research groups, i.e. developers of Falcon-AO (Jian et al., 2005).

3. Similarity of comments P_{kom} attached to classes $\mathcal{K}_i(i = 1,2)$ and individuals $\mathcal{B}_i(i = 1,2)$

   Third level of similarity between ontology elements if similarity between comments attached to those elements. Comments are being treated as parts of bipartite graph, where words are mapped to nodes of the graph and similarity between them are mapped to edges between those nodes. Similarity between the comments is than reduced to problem of maximal assignments between two graphs which can be solved using Hungarian Method (Kuhn, 1955). Similarities between nodes are being calculated using aforementioned methods.

   Finally, the value of P_{kom} is calculated as ratio of achieved maximal assignment to number of elements in longer of the two comments (Equation 1). This ensures that the value of P_{kom} will always be within the range $< 0,1 >$.

$$P_{kom} = \frac{\sum_i P_{lr_i}}{max(|L|,|R|)} \tag{1}$$

4. Structural similarity P_{str} of classes $\mathcal{K}_i(i = 1,2)$ and individuals $\mathcal{B}_i(i = 1,2)$

   This level of similarity takes into account structural placements of given concept in regard of its nearest neighbours. Both the type and direction

of the relation are being taken into account. This similarity can be described by Equation 2:

$$P_{str} = \frac{\sum_i min(r_i)}{\sum_i max(r_i)} \quad (2)$$

where:
r_i – number of occurrences of relation i (where i = {subsumption, membership, equality, disjointness, union, intersection}) in which given concept takes part.
This level of similarity is used where no other possibility can be applied as it provides the lowest amount of information.

The basic similarity measure in the lexical similarity is being proposed. With constant development of many lexicons this approach seems to be more and more justified. In current 3.0 version of WordNet 155287 different nouns in English language with 206941 word–meaning pairs can be found (Princeton University, 2010). It is highly probable that most of the words used in concepts description will be found within that lexicon. This way connections between concepts can be derived from WordNet and thus enriching the output ontology. Further levels of similarity allows mapping concepts into one another even when they are not found in the used lexicon, making the proposed algorithm usable in general scenarios.

3 THE PROPOSED ALGORITHM

The proposed algorithm is based on lexical and semantic analysis of integrated ontologies and operates on ontologies stored in OWL language. Furthermore it was observed that in most ontologies concept names are usually represented by nouns and most of the information is either explicitly represented by classes and relations between them or can be easily derived and transformed into such representation. It was also assumed that input ontologies are representing the same domain. Strictness of such assumption depends on used lexicon and usually improves the quality of output ontology (Boiński, 2012).

The WordNet dictionary was proposed as a knowledge base allowing adding and extending relations in output ontology (Boiński, 2012). The proposed algorithm takes two ontologies as input and produces one ontology as its output. The output ontology can be either a merge (in terms of unification of URI's in OWL language) or an alignment (in terms of importing source ontologies and including only mapping between their elements).

The proposed algorithm goes as follows:

```
 1: program OntologyMerger {
 2:  function combineSubTrees(
              node_A, node_B, node_C) {
 3:   for all child_A of node_A do
 4:    for all child_B of node_B do
 5:     result = compare(child_A, child_B);
 6:     if result == EQUAL then
 7:      node = combine(child_A, child_B);
 8:      add node as child of node_C;
 9:      combineSubTrees(
              child_A, child_B, node);
10:     else if result == DISJOINT then
11:      add child_A as child of node_C;
12:      add child_B as child of node_C;
13:     else if result ==
              A_ENCAPSULATES_B then
14:      add child_A as child of node_C;
15:      place child_B in subtree of child_A;
16:     else if result ==
              B_ENCAPSULATES_A then
17:      add child_B as child of node_C;
18:      place child_A in subtree of child_B;
19:     end if
20:    end for
21:   end for
22:  }

23: function placeInSubTree(node, root) {
24:  for all child of root do
25:   result = compare(node, child);
26:   if result == EQUAL then
27:    newNode = combine(node, child);
28:    add newNode as child of root;
29:    combineSubTrees(
              node, child, newNode);
30:   else if result == DISJOINT then
31:    add child with subtree
              as child of root;
32:    add node with subtree
              as child of root;
33:   else if result ==
              NODE_ENCAPSULATES_CHILD then
34:    add node as child of root;
35:    placeInSubTree(child, node);
36:   else if result ==
              CHILD_ENCAPSULATES_NODE then
37:    add child as child of root;
38:    placeInSubTree(node, child);
39:   end if
40:  end for
41: }

42: input Ontology_A;
43: input Ontology_B;
44: output Ontology_C;
45: combineSubTrees(
              root_of_A, root_of_B, root_of_C);
46: return Ontology_C;
47: }
```

The main job of the algorithm is performed by two functions:

- combineSubTrees – which merges subtrees of node_A and node_B and adds the result node_C as its subtree (lines 2-22),
- placeInSubTree – which finds the best placement of node in subtree of root recursively looking for proper placement in terms of logical encapsulation (lines 23-42).

The basic concept behind both of those functions is the same, thus only the first one will be described in detail. The algorithm starts with reading two ontologies A and B (lines 42 and 43). Ontology C is the output of the algorithm and is returned when it finishes (line 46). In OWL ontologies there always is a common class *owl:Thing* that is the root of the ontology. The algorithm starts its work from this node of every ontology thus combining subtree of ontology A *owl:Thing* with subtree of ontology B *owl:Thing*. The result is being added to *owl:Thing* of ontology C (line 45) by the first function which takes three parameters:

- analyzed node of ontology *A*,
- analyzed node of ontology *B*,
- node of ontology *C* which should be the root for combination of subtrees of the rest of the parameters.

Note that it is not required for the ontology to be a tree. The ontology however is analysed at the level of a concept and its direct descendants and this fragment can be considered as such.

The algorithm starts with comparing every element of ontology *A* with every other element of ontology *B* at the same level of detail (loops starting at lines 3 and 4). The action performed depends on the result of the comparison:

1. if the concepts in both ontologies are determined to be equal (according to measures presented in chapter 2) they are merged together (or connected with equivalent property) (line 7), added to the output ontology (line 8). Finally their subtrees are being combined together into one tree attached to this new node (line 9).
2. if the concepts are determined to be different they are added with their subtrees to the output ontology (lines 10-13). No further analysis of those subtrees is being performed.
3. if meaning of node from ontology *A* is more general than meaning of node from ontology *B* (lines 13-16) than node from ontology *A* is added to the ontology (line 13) and the node from ontology *B* is placed within the subtree of node from ontology *A* (line 15). Lookup of proper place for this node is done via function *placeInSubTree*.
4. if meaning of node from ontology *B* is more general than meaning of node from ontology *A* (lines (linie 16-19), the operations are analogous to those in point 3.

The *compare(node_A, node_B)* function (lines 5 and 25) is used to determine whether concepts are similar, disjoint or logically encapsulating. It utilises similarity levels introduced in chapter 2. First it checks lexical and than semantic similarity. Furthermore, wherever possible, sibling and parent-children relations between compared concepts are determined. Finally, if two previous means did not prove concept similarity, comments and structural similarity is utilized according to Equation 3.

$$P_{sk} = w_1 P_{str} + w_2 P_{kom} \tag{3}$$

where:
P_{str} - structural similarity derived from type and number of relations in which analysed concepts take part (Equation 2),
P_{kom} - semantic similarity of comments attached to analysed concepts(Equation 1),
w_i - weights of aforementioned similarities, as a result of tests, their values where determined to be as follows: $w_1 = 0,3$; $w_2 = 0,7$.

Similar as in other cases it was assumed that $P_{sk} \geq 0,7$ means the concepts are identical and $P_{sk} < 0,7$ means the concepts are different.

The second function (*placeInSubTree*) is similar in the way of performing it's tasks. Calculation of similarity between elements is done in the same way and the main loop is similar. The difference is that instead of combining two subtrees it locates best logical placement of one node (first parameter) in the subtree of other (second parameter).

4 RESULTS

The proposed algorithm was tested using selected ontologies developed by Ontology Alignment Evaluation Initiative as input for EON Ontology Alignment Contest (Euzenat, 2004) and specially developed security ontology (Boiński, 2012).

4.1 OAEI Ontologies

The tests were performed using reference ontology (describing Bibtex structure), its modifications and one unrelated ontology. All ontologies used in the tests are publicly available at http://oaei.ontologymatching.org/2006/benchmarks/. The results obtained by the algorithm were than compared with those provided by the contests authors.

Table 1: Results of comparing concepts *InCollection* and *Chapter* with their modified equivalents *dcsqdcsqd* and *dzqndbzq*.

Lemma A	Lemma B	$max(P_{lex}, P_{sem})$	P_{kom}	P_{str}	P_{sk}	Result
InCollection	dcsqdcsqd	0,10	1,00	1,00	1,00	EQUAL
InCollection	dzqndbzq	0,00	0,75	0,67	0,73	EQUAL
Chapter	dcsqdcsqd	0,11	0,75	0,67	0,73	EQUAL
Chapter	dzqndbzq	0,00	1,00	1,00	1,00	EQUAL

All tests yielded positive results. The following scenarios were considered:

- merging with identical ontology – all classes were connected with final similarity equal 1.0. One additional connection was introduced (between *Address* and *Reference*) because of the domain overlap, as source ontologies provide no additional info stating disjointness of those two concepts,
- merging with completely different ontology – Bibtex ontology was combined with ontology describing food and wines, all concepts were correctly marked as different,
- merging with similar ontologies stored using more general dialects of OWL language – all elements of source and target ontologies were described as identical with similarity equal to 1.0. Similarly as in the first test one additional connection was introduced (between *Address* and *Reference*),
- merging with identical ontology with removed labels (comment and structural similarity only) – the ontology was merged with identical but with labels replaced by random, meaningless strings. The algorithm based its work solely on comments and structure of both ontologies. The algorithm calculated most of the connections right. The only problem was with connecting concepts *InCollection* and *Chapter* with their respective matches in modified ontology, as they are located within the same structure nad have similar comments („A part of a book having its own title." and „A chapter (or section or whatever) of a book having its own title." respectively). Thus matches to concepts *dcsqdcsqd* and *dzqndbzq* could not be guessed correctly and the algorithm marked all four concepts as similar. Details of the results of those calculations are presented by Table 1.

In all cases the algorithm produced satisfactory results proving that's it's useful for small, domain oriented ontologies.

4.2 Security Ontology

The security ontology[1] was created both manually and using the proposed algorithm. It consists of three modules:

- Risk Core Concepts module – created from integration of ontologies based on ENISA dictionary (ENISA, 2006; Enisa, 2010) (43 classes and 28 properties), NIST dictionary (Guttman and Roback, 1995) (70 classes and 23 properties) and chapter of Sommerville Book „Software Engineering" (Sommerville, 2006) (40 classes and 22 properties). After integration the ontology consists of 122 classes and 63 properties.
- Basic Security Concepts module – based on Avienis taxonomy (Avizienis et al., 2004) (269 classes and 91 properties).
- Safety and Security Requirements module – based on Firesmith taxonomy (Firesmith, 2005a; Firesmith, 2005b) (195 classes and 56 properties).

The ontologies included in the Risk Core Concepts module and later all three modules were merged using the proposed algorithm and the obtained ontologies were compared with results of manual integration.

In the first step Risk Core Concepts module was created where the algorithm performed 1170 comparisons between 153 classes. Of those comparisons only 13 was incorrect (1,11%). Some of them resulted from errors in source ontologies which were corrected. Later on the three modules were integrated into the Security ontology. The algorithm performed 1956 comparisons of which only 31 was incorrect (1,59%). The resulting ontology was very similar to the one obtained manually. Plus some of the errors in source ontologies were discovered during the automated integration and could be corrected.

5 CONCLUSIONS

In all test cases the proposed algorithm proved to be useful. The tests show its usability both in case of small and large domain ontologies with efficiency around 98%. In all cases however the algorithm was dependent on quality of input ontologies and external lexicons used during concept mapping. With the further development of such lexicons quality of obtained

[1] Available at http://ocs.kask.eti.pg.gda.pl and (as OWL) at http://kask.eti.pg.gda.pl

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

results can be enhanced and ontologies from wider field of domains can be merged.

The algorithm can be easily implemented as a lightweight library and used in any kind of application managing OWL or RDF ontologies. Such usage can further improve interoperability between systems in heterogeneous environment by enabling them to understand messages sent to each other and map them to local knowledge bases represented as OWL ontologies.

ACKNOWLEDGEMENTS

This work was funded by Nation Science Centre under the grant N N516 476440

REFERENCES

Avizienis, A., Laprie, J., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33.

Boiński, T. (2012). *Procedures for merging and alignment of domain ontologies, PhD thesis [in Polish]*. Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology.

ENISA (2006). Risk management: implementation principles and inventories for risk management/risk assessment methods and tools. Technical report.

Enisa (2010). Enisa: a European Union Agency - Glossary of Risk Management. [Online; przegldano 01-12-2010].

Euzenat, J. (2004). Eon ontology alignment contest. [Online; przegldano 27-01-2012].

Euzenat, J. and Valtchev, P. (2004). Similarity-based ontology alignment in OWL-lite. In *ECAI 2004: 16th European Conference on Artificial Intelligence, August 22-27, 2004, Valencia, Spain: including Prestigious Applicants [sic] of Intelligent Systems (PAIS 2004): proceedings*, page 333. Ios Pr Inc.

Fellbaum, C. (1998). *WordNet: An electronic lexical database*. The MIT press.

Fernandez, M., Gomez-Perez, A., and Juristo, N. (1997). Methontology: from ontological art towards ontological engineering. In *Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering*, pages 33–40.

Firesmith, D. (2005a). A Taxonomy of safety-related requirements. In *International Workshop on High Assurance Systems (RHAS'05)*.

Firesmith, D. (2005b). A taxonomy of security-related requirements. In *International Workshop on High Assurance Systems (RHAS'05)*. Citeseer.

Gennari, J., Musen, M., Fergerson, R., Grosso, W., Crubézy, M., Eriksson, H., Noy, N., and Tu, S. (2003). The evolution of Protégé: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123.

Goczyła, K. (2011). *Ontologie w systemach informatycznych*. Akademicka Oficyna Wydawnicza EXIT, Warszawa.

Gómez-Pérez, A., Corcho, O., and Fernandez-Lopez, M. (2002). *Ontological Engineering*. Springer-Verlag, London, Berlin.

Guttman, B. and Roback, E. (1995). *An introduction to computer security: the NIST handbook*. DIANE Publishing.

Hovy, E. (1998). Combining and standardizing large-scale, practical ontologies for machine translation and other uses. In *Proceedings of the 1st International Conference on Language Resources and Evaluation (LREC)*, pages 535–542. Citeseer.

Jian, N., Hu, W., Cheng, G., and Qu, Y. (2005). Falcon-AO: Aligning ontologies with falcon. In *Integrating Ontologies Workshop Proceedings*, page 85. Citeseer.

Kuhn, H. (1955). The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Levenshtein, V. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710.

Lin, D. (1993). Principle-based parsing without overgeneration. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 112–120. Association for Computational Linguistics.

Lin, D. (1998). An information-theoretic definition of similarity. In *Proceedings of the 15th International Conference on Machine Learning*, volume 1, pages 296–304. Citeseer.

NeOn Project (2010). NeOn Book. http://www.neon-project.org/nw/NeOn_Book. [Online; viewed 09-10-2010].

Noy, N., Fergerson, R., and Musen, M. (2000). The knowledge model of Protege-2000: Combining interoperability and flexibility. *Knowledge Engineering and Knowledge Management Methods, Models, and Tools*, pages 69–82.

Noy, N., McGuinness, D., et al. (2001). Ontology development 101: A guide to creating your first ontology.

Princeton University (2010). WordNet Database Statistics. http:// wordnet.princeton.edu/ wordnet/ man/ wnstats. 7WN. html. [Online; viewed 22-11-2010].

Sommerville, I. (2006). Software Engineering. 8th. *Harlow, UK: Addison-Wesley*.

Stanford University School of Medicine (2010). Stanford Center for Biomedical Informatics Research. http://protege.stanford.edu. [Online; viewed 09-10-2010].