A Novel Algorithm for Fully Automated Ontology Merging Using Hybrid Strategy

C.R. Rene Robin
Department of Computer Science and Engineering, Jerusalem College of Engineering
Chennai-100, Tamil Nadu, India
E-mail: crrenerobin@gmail.com
Tel: 91-9840222341; Fax: 91-44-22461904

G.V. Uma
Department of Information Science and Technology
Anna University, Chennai-25, Tamil Nadu, India
E-mail: gvuma@annauniv.edu

Abstract

Ontology is a formal representation of knowledge by a set of concepts within a domain and relationship between those concepts. With hundreds of developers emerging everyday there are always more than one ontology for a domain. In this case different conceptualizations of same domain exist. To create a common repository of knowledge base and to remove overlaps in existing ontologies we go for Ontology Merging. Two or more Ontologies are merged to build bigger ontology for extended knowledge. This paper deals with an algorithm used for merging of ontologies automatically using a hybrid strategy. It consists of four sub strategies such as Lexical Matching, Semantic Matching, Similarity Check and Heuristics Functions. The user’s only job is to give the owl files as input and a merged owl file will be produced as output.

Keywords: Ontology, Ontology Merging, Lexical Matching, Semantic Matching, Similarity Check and Heuristics functions

1. Introduction

Since T. Berners-Lee, the creator of the Internet, proposed the semantic Web concept [Wang Jin et. al 2006], a series of relative research work has begun fully. Taking ontology as the basic tool to describe the information semantics on the Web has become a consensus of many researchers. Various investigations and applications based on ontology have emerged [Navigli R et. al]. At the same time, research oriented to semantic Web, aiming at semantic information processing, has draw a lot of people’s attention in information-search area.

An ontology [Gruber 1993] defines a common vocabulary for researchers who need to share information in a domain. It includes machine-interpretable definitions (semantics) of basic concepts in the domain and relations among them. Classes are the focus of most ontologies. Classes describe concepts in the domain. A class can have subclasses that represent concepts that are more specific than the superclass. Slots describe properties of classes and instances. So developing an ontology includes defining classes in the ontology, arranging the classes in a taxonomic hierarchy, defining slots and
describing allowed values for these slots, filling in the values for slots for instances [A. Gomez-Perez]. Despite efforts and experience in developing ontologies, there is no agreement on a methodology for building ontologies. The ontology development methodology problem has been investigated in several projects. First attempt was by [T. Gruber] that specifies some principles and criteria for the design of ontologies. We can create a knowledge base by defining individual instances of these classes filling in specific slot value information and additional slot restrictions.

Ontology merging [Ghidini et al.] is the process of creating a new single coherent ontology from two or more existing source ontologies related to the same domain. The new ontology will replace the source ontologies. A merging task resembles a construction of a new ontology. A development starts with defining the domain and a lexicon of a typical, common vocabulary. It forms a base for a hierarchy of concepts – they are divided into classes and proper relations are attached. There are different approaches to hierarchy construction: top-down starts with the most general concepts; bottom-up begins with the most detailed helps to avoid mistakes during modeling large domains [3]. A new algorithm for ontology algorithm has been developed, which integrates some important features in matching to achieve high quality results that will help in searching and exchanging information between ontologies. This paper illustrates in detail the main components of the proposed algorithm and explains how they interact with each other. It describes the hybrid strategies that are used and shows how the system can aggregate different results which are produced by different strategies.

The organization of the paper is as follows. In the next section, we reviewed the existing work on ontology merging. In Section 3, we discuss proposed algorithm for ontology merging using hybrid strategies. In Section 4, we tabulate the performance evaluation between the expected performance of the proposed algorithm and the existing algorithm. And the conclusion section summarizes the paper.

2. Related Work
Ontology merging [Pinto et al] is defined as “Combining different ontologies with the same subject domain and creating a unified ontology”. Kalfoglou and Schorlemmer (2002) distinguished ‘merging’ from mapping by their definition on ontology merging as “…product of this merge will be, at the very least, the intersection of the two given ontologies” and “…the engineer is in charge of making decisions that will affect the merging.”. Stumme, G. and Mädche, A. (2001a) presented FCA–MERGE, a bottomup technique for merging ontologies based on a set of documents. It consists of three steps of the technique: the linguistic analysis of the texts; the merging of the two contexts and the computation of the pruned concept lattice; and the semi-automatic ontology creation phase which supports the user in modeling the target ontology. The disadvantage of the technique is, it does not mention if the ontologies have the same/similar/complementary/orthogonal subject.

ONION [Mitra, P. Wiederhold, G. and Kersten, M.L. 2000] is a merging approach which was implemented by Stanford University’s Database Group. It provides articulation rules for resolving terminological heterogeneity and enables knowledge interoperability that will lead to a bridging of the semantic gap between different ontologies. ONION uses both lexical and graph-based techniques to suggest articulations. The method of finding lexical similarity between concept names uses dictionaries and semantic-indexing techniques based on co-occurrence of words in a text corpus. It uses the linguistic matcher to identify every possible pair of concepts in ontologies and assigns a similarity score to each pair. It then compares the similarity score with a threshold to determine whether or not to accept it; if it does, an articulation rule is generated. After linguistic matchers are presented, a structure-based matcher starts looking for further matches. Articulation rules express the relationship between concepts belonging to the ontologies. There are several semantic relationships with a built-in meaning: (SubClassOf; PartOf; AttributeOf; InstanceOf; ValueOf). The disadvantage of this approach is that it does not use a normalization process. The way of selecting matching elements is also not specific.
Prompt [Noy and Musen, 2000] is an algorithm for ontology merging and alignment embedded in Protégé 2000. It starts with the identification of matching class names. Based on this initial step an iterative approach is carried out for performing automatic updates, finding resulting conflicts, and making suggestions to remove these conflicts. This approach provides interactive suggestions to the users. It solves mismatches at terminological and scope of concept level, and it helps alignment by providing possible edit points and it supports repeatability. But it is not automatic which means every step requires user interaction.

Chimaera [McGuinness, D.L. Fikes, R. Rice, J. and Wilder, S. 2000] is a semi-automatic or interactive tool for merging ontologies. The engineer is in charge of making decisions that will affect the merging process. This tool starts by analyzing the ontologies to be merged. It automatically finds linguistic match merges, and if it cannot find any matching terms, it gives the user control over any further action. In fact, it is similar to PROMPT, as both are embedded in ontology editing environments and offer the user interactive suggestions. It solves mismatches at terminological and scope of concept level, and it helps alignment by providing possible edit points and it is not repeatability. But it is not automatic which means everything requires user interaction. It is very similar to PROMPT.

Mohammad Mustafa Taye [2010] on his finding concluded that Ontology matching involves determining the semantic heterogeneity between two or more domain specifications by considering their associated concepts. The process requires matching, but makes use of more expressive relations between ontology concepts (partOf, subsumes, etc.). It is generally agreed that such alignment not capable to be done manually beyond a certain complexity, size or number of ontologies. Therefore, in order to decrease the trouble of manual establishment and maintenance of alignments, automatic- or semi-automatic-techniques have to be developed.

3. Proposed Ontology Merging Process
The system proposed uses mixed strategies. Even if one of the strategies fails to acknowledge the match the other strategies will. This makes the proposed system better. It is fully automatic. The user’s only job is to give the owl files as input and a merged owl file will be produced as output. The strategies considered are Lexical Matching, Semantic matching using Wordnet, Similarity Checking of properties and Heuristic functions.

3.1. Proposed Ontology Merging Algorithm
It takes two owlfiles as input. Our merging process starts from the top in one owl file and from bottom in other. This way we can eliminate a lot of unnecessary comparison. This way makes more sense as it inherits the properties of the super most class, so while comparing if two owl files are even a little similar they will possess at least few similar properties. For comparing class names, Lexical Analysis and Semantic is used. If this has a match it means that the classes are same. So after a Similarity Check Heuristic is called. If Lexical and Semantic fails, it checks every class of the owlfile 1 with the class of owlfile 2 and saves the value of the classes and p value in an intermediate value. p value is the ratio of similarity between two classes. After which heuristic function is called. This process is repeated for every class of owlfile 2. In this process the output file owlfile 3 is initialized with owlfile 1 any addition from owlfile 2 is made in owlfile 3. As in merging it should have all values that are in both owl files. At last the owlfile 3 is returned as merged file.

**Input:** owlfile o1, owlfile o2  
**Output:** owlfile o3

**Function OntologyMerging(owlfile o1, owlfile o2)**

```java
String c1,c2,p3[];
bool l,s;
```
float p;
owlfile o3=copy of owlfile o1;
c1=MainClass(owlfile1);
c2=LeftMostLeafNode(owlfile2);
foreach Class c2 in owlfile o2
{
  if(!(l=LexicalAnalysis(c1.name,c2.name)))
    s=SemanticAnalysis(c1.name,c2.name);
    if(l||s)
      p=SimilarityCheck(c1,c2,p3[]);
    else
      foreach Class c1 in owlfile o1
      {
        p=SimilarityCheck(c1,c2,p3[]);
        Create a intermediate file f1
        Store the similarity value p with the class names
        c1=class with highest p value
        c1=c1.nextClass;
        if c1 is null
        break;
      }
    HeuristicFunction(l,s,p,c1,c2,p3[],o3,f1);
  }
  c2=c2.previousClass;
return owlfile o3;
}

3.1.1. Lexical Matching
This process includes two process, Stemming and string compare. Two class names are given to this module. It performs stemming which gives the root word as the output which is then compared in string compare.

Function LexicalAnalysis(String a,String b)
{
  String r1,r2;
  r1=Stemming(a);
  r2=Stemming(b);
  if(StringCompare(r1,r2))
    return true;
  else
    return false;
}

3.1.2. Semantic Matching
This module takes two class names as input. In this module the string comparison is made using the meaning of the words. For finding the meaning of the words Wordnet is used. Wordnet is a large lexical database of English. It consists of synonyms, and conceptual semantic and lexical relations. One of the class name is given as input to the wordnet which retrieves all possible synonyms of the word. Every synonym is compared with the other class name using lexical analysis.
Function SemanticAnalysis(String c1, String c2)
{
    String s[];
    bool m;
    s = Wordnet(c1);
    m = false;
    foreach string w in s
    {
        m = LexicalAnalysis(w, c1);
        if (m)
            break;
    }
    return m;
}

3.1.3. Similarity Checking of properties
This module takes two classes as input. Each of their properties is stored in an array. Every property of
the class is compared with the other. To perform the comparison again lexical and semantic analysis is
used. The number of similarities found is calculated and divided by the total number of properties. This
value is useful for determining how much a class is similar to the other class.

Function SimilarityCheck(Class c1, Class c2, String p3[])
{
    String p1[], p2[], p3[];
    int a, c = 0; float p;
    bool m, n;
    p1 = c1.properties;
    p2 = c2.properties;
    a = max(p1.length, p2.length);
    foreach String q in p1
    {
        foreach String e in p2
        {
            m = LexicalAnalysis(q, e);
            if (!m)
                n = SemanticAnalysis(q, e);
            if (m || n)
            {
                p3[c] = q;
                c++;
            }
        }
    }
    p = c / a;
    return p;
}
3.1.4. Heuristic Functions
This module is the final module which writes into the output file. This module checks if the lexical and semantic match is found. If found, that class is written into the file along with the properties which has been passed through similarity checking module. If no match found, then using the value calculated by the similarity checking module the most related classes are found. To find which class is the super class the number of properties is compared. Whichever has the highest is the subclass of the other.

```java
Function HeuristicFunction(bool l, bool s, float p, class c1, class c2, String p3[], owlfile o3, File f1)
{
  Class c;
  if(l||s)
    Write into the output file the class c1 with properties p3
  Read intermediate file f1
  c=the class in the row which has the highest p value
  d=class in the column
  search c in the owlfile o3
  if(c.no_of_properties > d.no_properties)
    Write into the output file d as c’s subclass with its properties
  else
    Write into the output file d as c’s superclass with its properties
}
```

3.2. Description of the Proposed Algorithm
In **OntologyMerging**, we pass the files to be merging o1 and o2. In this module, a owlfile o3 is created which is said to be the output file. This file has a copy of o1. As we are only merging two ontologies all values present in both the ontologies must be present in the final output too. So, we add the classes or properties from o2 to o1.

We start comparing classes from top of o1 and bottom of o2. We go for this procedure considering the principle that “if the leaf node has no similarity with the super most class then the ontologies completely differ”. For every class c2 in o2 Lexical Analysis of the c1 in o1 and c2 is performed. If the lexical analysis fails to yield a match Semantic Analysis is performed which also takes the values c1 and c2. If one of Lexical or Semantic Analysis produces match, Similarity Check module is called by passing c1 and c2. If a match is not found, for every class c1 in o1 Similarity Check is performed on c1 and c2. An intermediate File f1 is created in which the similarity values p along with the class names are stored. p value is the ratio of similarity between two classes. The class with the highest p value is chosen as c1. Next class of c1 is initialized to c1. Until c1 is null the process continues. When c1 is null, Heuristic Function is called. c2 is initialized to the previous class (higher level class or next leaf node) and the process continues till the last class of o2.

Lexical Analysis takes two string as input. In this case its class names. Stemming is performed for the class names to obtain the root value r1 and r2. The root value of the class names is compared using String Compare method. If a match is found the function returns Boolean value true or false. Semantic Analysis takes two string values. It uses WorldNet as a database to identify the synonyms of the class names. In this module, class name c1 is searched in the WorldNet. This gives all synonyms of c1, which is stored in s[], a array of strings. Those synonyms and c2 is passed for Lexical Analysis for comparison. If a match is found Boolean value true is returned or false value is returned. Similarity Check takes class c1, c2 and an array of string p3 which is used to store the properties of the merged class. In this module the properties of the c1 is stored in p1 and c2 in p2. The maximum of number of properties in p1 and in p2 is assigned to a integer variable ‘a’. Every property in p1 is compared with every other property in p2 using Lexical and Semantic Analysis. If a match is found the counter
variable c is incremented and the property is stored in p3. This module returns the value p which is
counter value divided by a. This value defines the similarity between the two classes. Heuristic
Function takes the result from Lexical Analysis l, Semantic Analysis s and Similarity Check along with
the classes, p value, its properties p3, intermediate file f1 and the output file o3. If Lexical or Semantic
match is found the class c1 is written into output file with properties p3. Else it reads the intermediate
file f1 and identifies the class in the row with highest p value which is stored in c. The class in
the corresponding column is stored in d. The class c is searched in o3. If the properties of c are more in
number, d is assigned as c’s super class. If the properties of d are more in number, c is assigned as d’s
super class. Accordingly the class with its properties is written into the output file o3.

4. Comparison between Existing Algorithms and the Expected Performance of the
Proposed Algorithm
The expected performance of the proposed algorithm for ontology merging is compared with the
existing algorithm such as Chimaera, ONION and PROMPT. The strengths of the proposed algorithm
are it is fully automatic, there is no user interaction, it uses four different strategies and even if one of
the strategies fails to acknowledge the match the other strategy will accomplish the task.

Table 1: Comparison between existing algorithms and the expected performance of the proposed algorithm

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Parameters</th>
<th>CHIMAERA</th>
<th>ONION</th>
<th>PROMPT</th>
<th>PROPOSED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User Interaction</td>
<td>More</td>
<td>More</td>
<td>More</td>
<td>Nil</td>
</tr>
<tr>
<td>2</td>
<td>String Matching</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>Semantic Matching</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>Instance Matching</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>Structure Matching</td>
<td>Yes</td>
<td>IDL, XML-Based</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>Ontology Structure</td>
<td>Ontologua</td>
<td>Concept,</td>
<td>High Value</td>
<td>OWL</td>
</tr>
<tr>
<td>7</td>
<td>Matching Selection is</td>
<td>---</td>
<td>Properties</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>based on</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>Normalization</td>
<td>No</td>
<td>Semi Automatic</td>
<td>No</td>
<td>Semi Automatic</td>
</tr>
<tr>
<td>9</td>
<td>Automation</td>
<td>Semi Automatic</td>
<td>Wordnet</td>
<td>Semi Automatic</td>
<td>Fully Automated</td>
</tr>
<tr>
<td>10</td>
<td>Additional Resources</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td></td>
</tr>
</tbody>
</table>

5. Conclusion
The proposed algorithm brings together techniques in Lexical Matching, Semantic Matching,
Similarity Check and Heuristics functions in order to provide a fully-automatic merging framework for
the purpose of improving semantic interoperability in heterogeneous systems. Such ontology alignment
means linking entities of source ontology with those of target ontology based on different features of
these ontologies and using different strategies. The proposed algorithm explores four different
measures of similarity based on strings, linguistics, heuristics and structure, to support Merging. The
merging task was performed manually for a long time, to simplify the process of merging semi
automatic tools like Ontolingua, Chimaera were developed with lots of human intervention. The
proposed algorithm using hybrid strategies increased the hope among the ontology engineers that it is
possible to have a fully automatic ontology merging tool to merge ontologies irrespective of its size.
The proposed algorithm uses mixed strategies. Even if one of the strategies fails to acknowledge the
match the other strategies will. This makes the proposed system better. It is fully automatic. The user’s
only job is to give the owl files as input and a merged owl file will be produced as output.
References


