A Schema-Based Approach Combined with Inter-Ontology Reasoning to Construct Consensus Ontologies

Jingshan Huang, Rosa Laura Zavala Gutiérrez, Benito Mendoza García, and Michael N. Huhns

Computer Science and Engineering Department, University of South Carolina, Columbia, SC 29208

{huang27, zavalagu, mendoza2, huhns}@engr.sc.edu

Abstract

As the Semantic Web gains attention as the next generation of the Web, the issue of reconciling different views of independently developed and exposed data sources becomes increasingly important. Ontology integration serves as a basis for solving this problem. In this paper, we describe an approach to construct a consensus ontology from numerous, independently designed ontologies. Our method has the following features: i) the matching is carried out at the schema level; ii) the alignment of the ontologies is performed without previous agreement on the semantics of the terminology used by each ontology; iii) both the linguistic and the contextual features of an ontology concept are considered; iv) WordNet is incorporated into the linguistic analysis phase; v) heuristic knowledge is integrated into the contextual analysis phase; and vi) reasoning rules based on the domain-independent relationships *subclass*, *superclass*, *equivalentclass*, *sibling*, and each ontology concept's property list are used to infer new relationships among concepts. We describe a set of experiments and provide an evaluation of the results that shows the accuracy of our system.

1. Introduction

A major goal of the envisioned Semantic Web is to provide an environment where data can be shared and processed by automated tools as well as by people (Berners-Lee, Hendler, and Lassila 2001). Suppose a user wants to compare information, e.g., price, rating, and location, of nearby daycare facilities. Such information may be on the Web, but it is not in a machine-readable form. The user would need to review and process all the data exposed in each provider's website in order to get the information needed. On the Semantic Web agents can carry out this task automatically.

The idea of intelligent software agents that freely surf the Web and make sense of the information they find and the fact that such information is organized, represented, and expressed in different ways, have created the need for developing tools and techniques in order for the agents to make use of that information. Common ontologies provide the infrastructure needed to add semantics to the data on the Web so that it can be understood by agents.

It is impractical to have a unique and global ontology that includes every concept that is or might be included as part of the Web. However, it is reasonable that there might be ontologies for specific domains and sub-domains of the Web, and even for individual Web pages. It is clear, then, that the challenge is to be able to align and use different ontologies.

In this paper, we describe **PUZZLE**, a system that constructs a consensus ontology from numerous, independently designed ontologies. Our work is an extension of (Stephens, Gangam, and Huhns 2004) where the main idea is that any pair of ontologies can be related indirectly through a *semantic bridge*, consisting of many other previously unrelated ontologies, even when there is no direct relationship between the pair.

In (Stephens, Gangam, and Huhns 2004) the main technique for semantic mapping between two ontology concepts relies on simple string and substring matching. We extend this work to incorporate: further linguistic analysis; contextual analysis based on the properties of the concepts in the ontology; extended use of WordNet (Miller 1995) to include the search of not only synonyms but also antonyms, plurals, hypernyms, and hyponyms; use of the Java WordNet Library API (JWNL 2003) for performing run time access to the dictionary, instead of having to initialize the synsets a priori; integration of heuristic knowledge into the contextual analysis phase; and reasoning rules based on the domain-independent relationships *subclass*, *superclass*, *equivalentclass*, *sibling*, and each ontology concept's property list to infer new relationships among concepts.

Existing research efforts incorporate some of these features, but none has investigated them in combination. The combination addresses the major challenges described in (Stephens, Gangam, and Huhns 2004): different terminology for similar concepts and inconsistent relationships among concepts.

Our methodology is appropriate when there are a large number of small ontologies. Furthermore, in the case where the information is available through the Web, we assume that sites have been annotated with ontologies

(Pierre 2000), which is consistent with several visions for the Web (Berners-Lee, Hendler, and Lassila 2001).

The rest of the paper is organized as follows. Section 2 briefly discusses related work in ontology matching. Section 3 gives an overview of the **PUZZLE** system, whose details are described in Section 4. Section 5 reports the experiments we conducted and analyzes the results. Section 6 concludes.

2. Related Work

A lot of research work has been carried out in ontology matching. There are two approaches to ontology matching (Rahm and Bernstein 2001): instance-based and schema-based. All of the systems mentioned below belong to the latter, except for GLUE.

GLUE (Doan et al. 2003) introduces well-founded notions of semantic similarity, applies multiple machine learning strategies, and can find not only one-to-one mappings, but also complex mappings. However, it depends heavily on the availability of instance data. Therefore, it is not practical for cases where there is not a significant number of instances or no instance at all.

For HELIOS (Castano et al. 2004), WordNet is used as a thesaurus for synonyms, hyponyms, hypernyms, and meronyms. However the thesaurus has to be initialized for each domain for which it is used. If additional knowledge or a different domain is needed then the user has to input the respective terminology interactively.

PROMPT (Noy and Musen 2000) is a tool making use of linguistic similarity matches between concepts for initiating the merging or alignment process, and then use the underlying ontological structures of the Protege-2000 environment to inform a set of heuristics for identifying further matches between the ontologies. PROMPT has a good performance in terms of precision and recall. However, user intervention is required, which is not always available in real world application.

Cupid (Madhavan, Bernstein, and Rahm 2001) combines linguistic and structural schema matching techniques, as well as the help of a precompiled dictionary. But it can only work with a tree-structured ontology instead of a more general graph-structured one, which introduces many limitations to its application, because a tree cannot represent multiple-inheritance, an important characteristic in ontologies.

COMA (Do and Rahm 2002) provides an extensible library of matching algorithms, a framework for combining results, and an evaluation platform. According to their evaluation, COMA performs well in terms of precision, recall, and overall measures. Although it is a composite schema matching tool, COMA does not integrate reasoning and machine learning techniques.

Similarity Flooding (Melnik et al. 2002) utilizes a hybrid matching technique based on the idea that similarity spreading from similar nodes to the adjacent neighbors. Before a fix-point is reached, alignments between nodes are refined iteratively. This algorithm only considers the simple linguistic similarity between node names, leaving behind the node property and inter-node relationship.

S-Match (Giunchiglia, Shvaiko, and Yatskevich 2004) is a modular system into which individual components can be plugged and unplugged. The core of the system is the computation of relations. Five possible relations are defined between nodes: equivalence, more general, less general, mismatch, and overlapping. Giunchiglia et al. claim that S-Match outperforms Cupid, COMA, and SF in measurements of precision, recall, overall, and F-measure. However, as Cupid does, S-Match uses a tree-structured ontology.

In (Williams, Padmanabhan, and Blake 2003), a method is investigated for agents to develop local consensus ontologies to help in communications within a multiagent system of B2B agents. They show the potential brought by local consensus ontologies in improving how agents conduct B2B Web service discovery and composition. They also explore the influence of a lexical database in ontology merging.

3. Overview of Our Solution

The most important differences between **PUZZLE** and the systems mentioned in Section 2 are that **PUZZLE**:

- Requires no user intervention and is automated;
- Represents an ontology as a graph instead of a tree;
- Integrates WordNet by using the JWNL API;
- Applies heuristic knowledge during linguistic matching;
- Reasons with additional relations during context matching.

The goal of our work is to construct a consensus ontology from numerous independently designed ontologies. The main idea of our approach is that any pair of ontologies, G_1 and G_2, can be related indirectly through a semantic bridge consisting of other previously unrelated ontologies, even when there is no direct relationship between G_1 and G_2. The metaphor is that a small ontology is like a piece of jigsaw puzzle. It is difficult to relate two random pieces of a jigsaw puzzle until they are constrained by other puzzle pieces. Furthermore, for the semantic bridge between a given pair of ontologies G_1 and G_2, the more ontologies the semantic bridge comprises, the better the semantic match between G_1 and G_2.

In order to construct a consensus ontology from a number of ontologies, we take two ontologies and merge them into a new one, then we iteratively merge the resultant ontology with each additional one. We will explain next our method for merging two ontologies.

Suppose that original ontologies are built according to OWL Full specification (W3C 2004). Internally, our system represents an ontology using a directed acyclic graph G (V, E), where V is a set of ontology concepts (nodes), and E is a set of edges between two concepts, i.e., $E = \{(u, v) \mid u$ and v belong to V and u is a *superclass* of $v\}$. In addition, we assume that all ontologies share "#Thing" as a common "built-in" root. In order to merge two ontologies, G_1 and G_2, we try to relocate each concept from

```
PUZZLE Algorithm – merge(G1, G2)
Input: Ontology G1 and G2
Output: Merged ontology G2
Begin
    new location of G1 's root = G2 's root
    for each node C (except for the root) in G1
        Parent(C) = C's parent set in G1
        for each member pi in Parent(C)
            pj = new location of pi in G2
            relocate(C, pj)
        end for
    end for
end
```

Figure 1. **PUZZLE** Algorithm

one ontology into the other. We adopt a width-first order to traverse G_1 and pick up a concept C as the target to be relocated into G_2. Consequently, C's parent set *Parent(C)* in the original graph G_1 has already been put into the suitable place(s) in the destination graph G_2 before the relocation of C itself. The pseudocode in figure 1 describes the top level procedure of our algorithm.

The *relocate* function in the above algorithm is used to relocate C into a subgraph rooted by p_j. To obtain the correct relocation, we need to consider both the linguistic feature and the contextual feature of these two concepts (described in sections 4.1. and 4.2. respectively). The pseudocode for the *relocate* function is shown in figure 2. Notice that there is a recursive call to itself within *relocate*.

```
relocate(N1, N2)
Input: nodes N1 and N2
Output: the modified structure of N2 according to information from N1
begin
    if there exists any equivalentclass of N1 in the child(ren) of N2
        merge N1 with it
    else if there exists any subclass of N1 in the child(ren) of N2
        Children(N1) = set of such subclass(es)
        for each member ci in Children(N1)
            add links from N2 to N1 and from N1 to ci
            remove the link from N2 to ci
        end for
    else if there exists any superclass of N1 in the child(ren) of N2
        Parent(N1) = set of such superclass(es)
        for each member pi in Parent (N1)
            recursively call relocate(N1, pi)
        end for
    else
        add a link from N2 to N1
    end if
end
```

Figure 2. *relocate* Function

This recursive procedure is guaranteed to terminate because the number of the nodes within a graph is finite, and the worst case is to call *relocate* repetitively until we hit a node without child.

4. Details of the PUZZLE System

When trying to match concepts, we consider both the linguistic and the contextual features. The meaning of an ontology concept is determined by its name and its relationship with other concept(s). In this paper, we assume that the linguistic factors contribute 70 percent and the contextual factors contribute 30 percent in concept matching. The former is greater than the latter, because in our experiments, the input ontologies have less contextual information. Therefore, we do not want the contextual factors to dominate in the matching process. Notice that these weight values can always be customized according to different application requirements. For example, when merging diverse ontologies, i.e., ones with rich linguistic but poor contextual information versus ones with poor linguistic but rich contextual information, appropriate weight values can be applied accordingly.

4.1. Linguistic Matching

The linguistic factor reflects how the ontology designer wants to encode the meaning of the concept by choosing a preferable name for it. Our **PUZZLE** system uses both string and substring matching techniques when performing linguistic feature matching. Furthermore, we integrate WordNet by using JWNL API in our software. In this way, we are able to obtain the synonyms, antonyms, hyponyms, and hypernyms of an English word, which is shown to increase the accuracy of the linguistic matching dramatically. In addition, WordNet performs some preprocessing, e.g., the transformation of a noun from plural form to single form.

We claim that for any pair of ontology concepts C and C', their names N_C and $N_{C'}$ have the following mutually exclusive relationships, in terms of their linguistic features.

- *anti-match*: N_C is a antonym of $N_{C'}$, with the matching value $v_{name} = 0$;
- *exact-match*: either N_C and $N_{C'}$ have an exact string matching, or they are the synonyms of each other, with the matching value $v_{name} = 1$;
- *sub-match*: N_C is either a postfix or a hypernym of $N_{C'}$, with the matching value $v_{name} = 1$;
- *super-match*: $N_{C'}$ is either a postfix or a hyponym of N_C, with the matching value $v_{name} = 1$;
- *leading-match*: the leading substrings from N_C and $N_{C'}$ match with each other, with the matching value v_{name} = length of the common leading substring/length of the longer string. For example, "active" and "actor" have a common leading substring "act", resulting in a *leading-match* value of 3/6;
- *other*.

When relocating C, we perform the linguistic matching between C and all the candidate concepts. For each candidate concept C', if an *exact-match* or a *leading-match* is found, we put C' into C's candidate *equivalentclass* list; if a *sub-match* is found, we put C' into C's candidate

subclass list; and if a *super-match* is found, we put *C'* into *C*'s candidate *superclass* list. Then we continue the contextual matching between *C* and each concept in the three candidate lists to make the final decision.

Notice that using a synonym as the candidate of *equivalentclass* is an approximate approach, because each word could have multiple senses. We are making an assumption that different ontologies deal with similar domain (otherwise it is of little significance to align them). Therefore, in most cases, it is suitable to regard one concept's synonym(s) as a possible *equivalentclass* concept. Also, the approach to put a *sub-match* concept into another's candidate *subclass* list is approximate. In some cases it is not correct, e.g., "firstname" is not a *subclass* of "name". However, this approach does provide a lot of useful information and possible correct relationships in many cases. Similarly, *leading-match* sometimes does not offer accurate help as we expect. Because we are not considering linguistic matching alone, this kind of bias brought by *leading-match* is tolerable and under control.

4.2. Contextual Matching

The context of an ontology concept *C* consists of two parts, its property list and its relationship(s) with other concept(s). We discuss this next in detail.

4.2.1. Property List Matching

Considering the property lists, *P(C)* and *P(C')*, of a pair of concepts *C* and *C'* being matched, our goal is to calculate the similarity value $v_{Property}$ between them.

$$v_{Property} = w_{required} * v_{required} + w_{non\text{-}required} * v_{non\text{-}required}$$

$v_{required}$ and $v_{non\text{-}required}$ are the similarity values calculated for the *required* property list and *non-required* property list respectively. $w_{required}$ and $w_{non\text{-}required}$ are the weights assigned to each list. In this paper, we choose 0.7 and 0.3 for $w_{required}$ and $w_{non\text{-}required}$. $v_{required}$ and $v_{non\text{-}required}$ are calculated by the same procedure. We will explain next in detail how to obtain $v_{required}$, and from this point on, "property" means "required property" for concision purpose.

Suppose the number of properties in two property lists, P_1 and P_2, is n_1 and n_2 respectively. Without loss of generality, we assume that $n_1 \leq n_2$. There are three different matching models between two properties.

1. *total-match*
   - The linguistic matching of the property names results in either an *exact-match*, or a *leading-match* with $v_{name} \geq 0.9$; and
   - The data types match exactly.

   Let v_1 = number of properties with a *total-match*, and $f_1 = v_1/n_1$. Here f_1 is a *correcting* factor embodying the integration of heuristic knowledge. We claim that between two property lists, the more pairs of properties being regarded as *total-match*, the more likely that the remaining pairs of properties will also hit a match as long as the linguistic match between their names is above a certain threshold value. For example, assume that both P_1 and P_2 have ten properties. If there are already nine pairs with a *total-match*, and furthermore, if we find out that the names in the remaining pair of properties are very similar, then it is much more likely that this pair will also have a match, as opposed to the case where only one or two out of ten pairs have a *total-match*.

2. *name-match*
   - The linguistic matching of the property names results in either an *exact-match*, or a *leading-match* with $v_{name} \geq 0.9$; but
   - The data types do not match.

   Let v_2 = number of properties with a *name-match*, and $f_2 = (v_1 + v_2)/n_1$. Similarly to f_1, f_2 also serves as a *correcting* factor.

3. *datatype-match*

   Only the data types match. Let v_3 = number of properties with a *datatype-match*.

According to the above definition, first, we try to find out all pairs of *total-match* and filter them out of the original properties, then in the remaining properties find out all pairs of *name-match* and filter them too, and finally in the rest of original properties find out all pairs of *datatype-match*. Now we can calculate the similarity value $v_{required}$ between the two property lists.

$$v_{required} = (v_1 * w_1 + v_2 * (w_2 + w_2' * f_1) + v_3 * (w_3 + w_3' * f_2))/n_1$$

where:

- the value range of $v_{required}$ is from 0 to 1;
- w_i (i from 1 to 3) is the weight assigned to each matching model. We use 1.0 for *total-match*, 0.8 for *name-match*, and 0.2 for *datatype-match*;
- w_i'(i from 2 to 3) is the *correcting* weight assigned to the matching models of *name-match* and *datatype-match*. We use 0.2 and 0.1 respectively;

Notice that all the thresholds and arguments in the formulas mentioned in this section are based on trial-and-error.

4.2.2. Relationships among Concepts

Given any two ontology concepts, we can have the following five mutually exclusive relationships between them:

- *subclass*, denoted by $\subseteq$
- *superclass*, denoted by $\supseteq$
- *equivalentclass*, denoted by $\equiv$

- *sibling*, denoted by $\approx$ and
- other, denoted by $\neq$

OWL Full provides eleven axioms (W3C 2004): *subClassOf*, *equivalentClass*, *disjointWith*, *sameIndividualAs*, *differentFrom*, *subPropertyOf*, *equivalentProperty*, *inverseOf*, *transitiveProperty*, *functionalProperty*, and *inverseFunctionalProperty*. The first two axioms will be used to represent the *subclass-superclass* and *equivalentclass* relationships respectively.

4.3. Reasoning Rules

Based on the linguistic and contextual features, **PUZZLE** uses three domain-independent rules, each regarding the relationship among ontology concepts, to incorporate the reasoning into our system. These rules are applied to concepts from different ontologies. Therefore, we refer to them as *inter-ontology reasoning*.

Suppose we have three ontologies A, B, and C, each of which is designed according to the OWL Full specification. Furthermore, let *n(A)*, *n(B)*, and *n(C)* be the sets of concepts in A, B, and C respectively, with $n_i(A)$, $n_j(B)$, and $n_k(C)$ be the individual concept for each set (*i* from 1 to $|n(A)|$, *j* from 1 to $|n(B)|$, and *k* from 1 to $|n(C)|$), and $P(n_i(A))$, $P(n_j(B))$, and $P(n_k(C))$ be the property list for each individual concept.

Consider the property lists $P(n_i(A))$ and $P(n_j(B))$, let s_i and s_j be the set size of these two lists. There are four mutually exclusive possibilities for the relationship between $P(n_i(A))$ and $P(n_j(B))$:

- $P(n_i(A))$ and $P(n_j(B))$ are consistent with each other if and only if

  i. *Either* $s_i = s_j$ *or* $|s_i - s_j|/(s_i + s_j) \leq 0.1$, and
  ii. $v_{Property} \geq 0.9$

  We denote the corresponding concepts $n_i(A)$ and $n_j(B)$ by $n_i(A) \overset{p}{\longleftrightarrow} n_j(B)$;

- $P(n_i(A))$ is a subset of $P(n_j(B))$ if and only if

  i. $s_i \leq s_j$, and
  ii. $v_{Poperty} \geq 0.9$

  We denote the corresponding concepts $n_i(A)$ and $n_j(B)$ by $n_i(A) \overset{p}{\longrightarrow} n_j(B)$;

- $P(n_i(A))$ is a superset of $P(n_j(B))$ if and only if

  i. $s_i \geq s_j$, and
  ii. $v_{Property} \geq 0.9$

  We denote the corresponding concepts $n_i(A)$ and $n_j(B)$ by $n_i(A) \overset{p}{\longleftarrow} n_j(B)$;

- $P(n_i(A))$ and $P(n_j(B))$ have other relationship which will not be considered in our system.

Rule 1 and 2 consider two ontologies, A and B.

[**Rule 1**] This rule is straightforward, claiming that the *superclass/subclass* relationship of a class is transferable to its equivalent class(es).

- Preconditions:
  $n_i(A) \equiv n_k(B)$ and ($n_i(A) \subseteq n_j(A)$ or $n_i(A) \supseteq n_j(A)$)
- Conclusion:
  $n_k(B) \subseteq n_j(A)$ or $n_k(B) \supseteq n_j(A)$

[Rule 2] If two classes share the same parent(s), then their relationship is one of: *equivalentclass*, *superclass*, *subclass*, and *sibling*. For example, if we know that two classes have similar names and similar property lists, we still cannot conclude that they must be equivalent to each other, because of the possibility of badly designed ontologies. However, if we also know that these two classes have the same parent(s), then the probability of them being equivalent will increase substantially.

- Preconditions:
  $n_{i1}(A) \supseteq n_{i2}(A)$ and $n_{k1}(B) \supseteq n_{k2}(B)$ and
  $n_{i1}(A) \equiv n_{k1}(B)$ and
  1. $n_{i2}(A) \overset{p}{\longleftrightarrow} n_{k2}(B)$ and (the names of $n_{i2}(A)$ and $n_{k2}(B)$ have either an *exact-match*, or a *leading-match* with $v_{name} \geq 0.8$)
  2. $n_{i2}(A) \overset{p}{\longrightarrow} n_{k2}(B)$ and the name of $n_{k2}(B)$ is a *sub-match* of the name of $n_{i2}(A)$
  3. $n_{i2}(A) \overset{p}{\longleftarrow} n_{k2}(B)$ and the name of $n_{k2}(B)$ is a *super-match* of the name of $n_{i2}(A)$
  4. None of above three holds
- Conclusion:
  1. $n_{i2}(A) \equiv n_{k2}(B)$
  2. $n_{i2}(A) \supseteq n_{k2}(B)$
  3. $n_{i2}(A) \subseteq n_{k2}(B)$
  4. $n_{i2}(A) \approx n_{k2}(B)$

Rule 3 considers three ontologies, A, B, and C.

[Rule 3] If two classes have no direct relationship between them, we will refer to a third one, in order to find out the semantic bridge between the original two. In theory, the more ontologies the semantic bridge comprises, the more likely we can succeed in discovering the hidden relationships that are not obvious originally.

- Preconditions:
  $n_{i1}(A) \equiv n_{j1}(C)$ and $n_{j2}(C) \equiv n_{k2}(B)$ and
  $n_{k1}(B) \subseteq n_{k2}(B)$ and $n_{j1}(C) \subseteq n_{j2}(C)$ and
  1. $n_{i1}(A) \overset{p}{\longleftrightarrow} n_{k1}(B)$ and (the names of $n_{i1}(A)$ and $n_{k1}(B)$ have either an *exact-match*, or a *leading-match* with $v_{name} \geq 0.8$)
  2. $n_{i1}(A) \overset{p}{\longrightarrow} n_{k1}(B)$ and the name of $n_{k1}(B)$ is a *sub-match* of the name of $n_{i1}(A)$

3. $n_{i1}(A) \xleftarrow{p} n_{k1}(B)$ and the name of $n_{k1}(B)$ is a *super-match* of the name of $n_{i1}(A)$
4. None of above three holds

- Conclusion:
  1. $n_{i1}(A) \equiv n_{k1}(B)$
  2. $n_{i1}(A) \supseteq n_{k1}(B)$
  3. $n_{i1}(A) \subseteq n_{k1}(B)$
  4. $n_{i1}(A) \approx n_{k1}(B)$

5. Evaluation and Discussion of Our Results

Using a set of local ontologies designed by students, we evaluated our system in terms of precision, recall, and merging convergence. The purpose of the evaluation was to determine whether or not **PUZZLE** generates a consensus ontology.

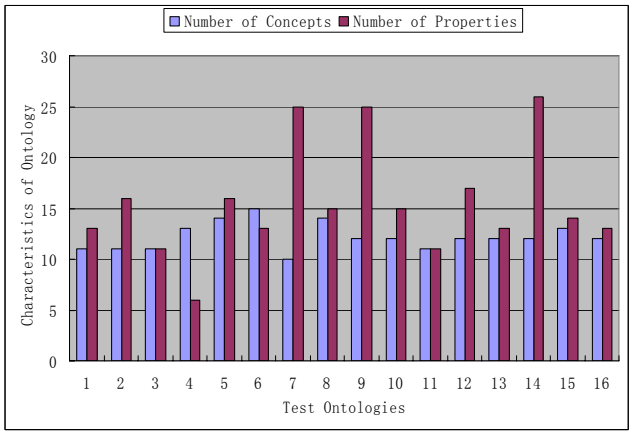

Figure 3. Characteristics of the Test Ontologies

5.1. Experimental Setup

- Configuration of the experimental platform
  Pentium 4 1.8GHz processor/512 MB RAM/40 GB hard disk/Windows XP Professional 2002 with SP2
- Programming environment
  JBuilder 9.0 with J2SE 1.5.0
- Test ontologies
  Sixteen ontologies for the domain of "Building" were constructed by graduate students in computer science and engineering at our university and used for evaluating the performance of the **PUZZLE** system. The characteristics of these ontology schemas can be found in figure 3. They had between 10 and 15 concepts with 6 to 26 properties.

5.2. Experimental Results and Analysis

Our experiments simulate having sixteen agents, each of which has a local ontology and is willing to communicate with the other agents. They try to reconcile their local ontologies to form a consensus one.

5.2.1. Evaluation of the Resultant Ontology

To decide whether a consensus ontology is obtained, we asked two ontology experts to carry out a manual mapping and we compared their results with ours. A random order was chosen during the process of merging ontologies one at a time, and both human and our system carried out the merging according to that same order, then both *precision* and *recall* measurements were applied in the evaluation. These two measurements refer to the total number of concepts with relationship of *subclass*, *superclass*, *equivalentclass*, and *sibling* up to the point at the end of each round of merging. The evaluation result is shown in figure 4. Notice that this result is not statistically valid but indicative. Both measurements reflect a promising result, except when we merged the third and the ninth ontologies. We checked the original ontologies and found out that a reason for the unsatisfactory result is due to unreasonably designed ontologies. For example, in one of the ontologies, "HumanBeing" and "InsectSpecie" are the only properties of the concept "LivingThing".

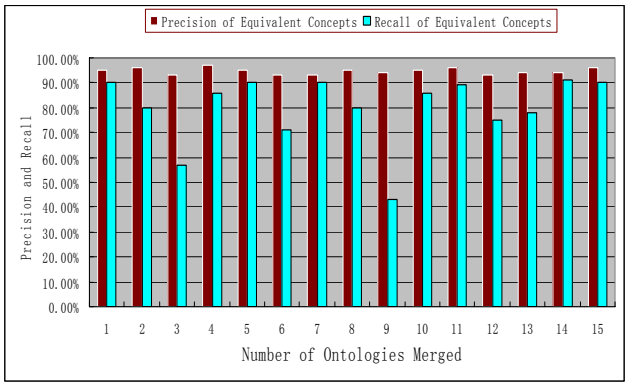

Figure 4. Precision and Recall Measurements of Resultant Ontology

5.2.2. Analysis of Merging Convergence

One hypothesis is that as each additional ontology is merged into a consensus one, there should be fewer new items (concept, relationship, or property) added to the consensus. To test this hypothesis, the following experiment has been conducted. We calculated the number of newly discovered information when the first, second, fifth, tenth, twelfth, thirteenth, and fifteenth ontologies were merged. Figure 5 shows the results of this experiment, which verifies the hypothesis.

Out of the 16 ontologies we had available for our experiments, we considered all possible combinations of the order by which they could be merged, in order to remove any bias that might be introduced by the presence of unusual ontology samples. This is a huge number; for example, there are 1680 combinations when the second ontology is to be merged, and 25000 for the fifth one. It is impossible to try all these orders. Our solution is that if the population size is less than or equal to 30 we try all

possible orders, otherwise we randomly choose a sample space of size 30.

A monotonically decreasing pattern is shown in figure 5. As the number of ontologies already merged increases, the number of concepts, relationships, and properties learned from additional ontologies decreases. We believe that the number of new items will eventually converge to zero, although the sixteen ontologies we have available for this experiment are not enough to verify this belief.

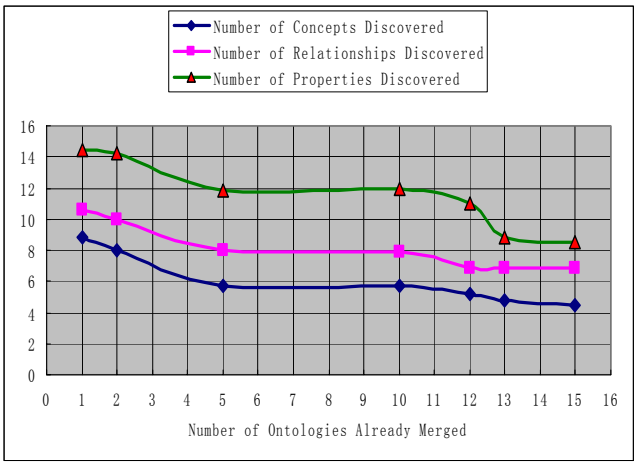

Figure 5. Merging Convergence Experiment

5.2.3. Other Features of PUZZLE

- **PUZZLE** removes redundant *is-a* links that are already specified by the transitivity of the *superclass-subclass* relationship.
- Our use of WordNet increases the accuracy. For instance, none of the original ontologies mentioned the relationship between the concepts "Monument" and "Structure". However, **PUZZLE** found out that the concept "Monument" is a *subclass* of the concept "Structure", which is quite reasonable and is an additional piece of information added to the merged ontology.

6. Conclusion and Future Work

Ontology matching is a critical operation in the Semantic Web. In this paper, we presented the **PUZZLE** system, a schema-based approach combined with inter-ontology reasoning, which reconciles ontologies for applications within a single domain. This completely automated matching is carried out at the schema level, without a previous agreement over the different terminology semantics. **PUZZLE** considers both linguistic and contextual features of an ontology concept, integrates heuristic knowledge with several matching techniques, and incorporates the reasoning among ontologies. A set of experiments showed a promising result from this system. Note that the resultant ontology represents a consensus model of a domain, but not necessarily a correct model. The possible incorrectness comes from the unreasonable (wrong) design of original ontologies.

Several remaining tasks are envisioned. We plan to adopt machine learning techniques to obtain more accurate results; take into consideration other relationships such as *partOf*, *hasPart*, *causeOf*, and *hasCause*; integrate the OWL Validator into our system; analyze the time complexity of the algorithm; and test our system against other well-known ones in ontology matching, by using more general ontology libraries.

Acknowledgements

Thanks to Dr. Larry M. Stephens for providing the source code and ontology files from a prior project; thanks to Prof. Ronald D. Bonnell and Dr. Csilla Farkas for encouraging their students to participate in the experiment. We also thank Goradia Hrishikesh and Jiangbo Dang for discussions about the system.

References

Stephens, L.; Gangam, A.; and Huhns, M. N. 2004. Constructing Consensus Ontologies for the Semantic Web: A Conceptual Approach. *World Wide Web Journal*, Vol. 7, No. 4, pages 421 - 442: Kluwer Academic Publishers.

W3C 2004. OWL Web Ontology Language Reference. *http://www.w3.org/TR/owl-ref.*

Miller, A. G. 1995. WordNet: A Lexical Database for English. In *Communications of the ACM*, Vol. 38, No. 11, pages 39 - 41: ACM Press.

Castano, S.; Ferrara, A.; Montanelli, S.; and Racca, G. 2004. Matching Techniques for Resource Discovery in Distributed Systems Using Heterogeneous Ontology Descriptions. In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC04)*, Vol. 1, pages 360 - 366: IEEE Computer Society Press.

Doan, A.; Madhavan, J.; Dhamankar, R.; Domingos, P.; and Halevy, A. 2003. Learning to match ontologies on the Semantic Web. *The VLDB Journal* (2003), Vol. 12, pages 303 - 319: Springer-Verlag.

Giunchiglia, F.; Shvaiko, P.; and Yatskevich, M. 2004. S-Match: an algorithm and an implementation of semantic matching. In *Proceedings of the 1st European Semantic Web Symposium*, Vol. 3053, pages 61 - 75: Springer-Verlag.

Melnik, S.; Garcia-Molina, H.; and Rahm, E. 2002. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In

Proceedings of the 18th International Conference on Data Engineering: IEEE Computer Society Press.

Noy, N. F., and Musen, M. A. 2000. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proceedings of the17th National Conference on Artificial Intelligence (AAAI 2000)*: AAAI Press.

Rahm, E., and Bernstein, P. A. 2001. A survey of approaches to automatic schema matching. *The VLDB Journal* (2001), Vol. 10, pages 334 - 350: Springer-Verlag.

Do, H., and Rahm, E. 2002. COMA – A system for flexible combination of schema matching approaches. In *Proceedings of the 28th VLDB Conference*: Springer-Verlag.

Madhavan, J.; Bernstein, P. A.; and Rahm, E. 2001. Generic Schema Matching with Cupid. In *Proceedings of the 27th VLDB Conference*: Springer-Verlag.

Berners-Lee, T.; Hendler, J.; and Lassila, O. 2001. The Semantic Web. *Scientific American*, Vol. 284, No. 5, pages 34 - 43: Scientific American, Inc.

JWNL 2003. Java WordNet Library – JWNL 1.3 *http://sourceforge.net/projects/jwordnet/*.

Pierre, J. M. 2000. Practical Issues for Automated Categorization of Web Sites. In *Electronic Proceedings of ECDL 2000 Workshop on the Semantic Web* (http://www.ics.forth.gr/proj/isst/SemWeb/program.html).

Williams, A.; Padmanabhan, A.; and Blake, M. B. 2003. Local Consensus Ontologies for B2B-Oriented Service Composition. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, Session: Ontologies, pages 647 - 654: ACM Press.