Sharing Ontology Schema Information for Web Service Integration

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

Abstract

In spite of many standards efforts, Web services with similar or compatible functionalities often have heterogeneous semantics. One reason is the disparate ontologies used for service descriptions. In order to compare and compose Web services, the ability to merge different ontologies is essential. This paper describes an approach to align numerous, independently designed ontologies. Our approach is a completely automated one, without the need for prior agreement on semantics. It incorporates WordNet and heuristic reasoning and infers new knowledge by self-learning. Our system provides a solid base for the seamless integration of Web services.

1. Introduction

In traditional business, great potential value will be added if Web service applications are to be integrated. Originally business partners needed to predefine the terminology of their interaction using EDI standards such as ebXML [15]. Therefore, the automation activities in the Web were tightly coupled. Nowadays Web services can be considered the next generation of EDI with extra capabilities because they also allow sharing tasks and automating processes. As they are based on simple and non-proprietary standards (i.e., UDDI for discovery, WSDL for description, BPEL4WS for coordination, WSCI for choreographed interactions, and SOAP for communication), Web services promise to increase interoperability and reuse, and lower the costs of software integration and data-sharing with partners.

There is no doubt users can obtain some value in accessing a single Web service through a semantically well-founded interface. However, a greater value is derived through enabling a flexible composition of services, which will not only create new services, but also potentially add value to preexisting ones [1]. Therefore, the seamless composition of distributed Web services becomes important. On the other hand, because there is no agreed-upon global ontology, Web services from different providers are usually featured by heterogeneous semantics. A challenging but worthwhile goal, then, is to be able to share schema information from different ontologies.

In this paper, we introduce a system (**PUZZLE**) implementing an approach to merge/align distributed and independently designed ontologies. In [2] the main technique for semantic mapping between two ontology concepts relies on simple string and substring matching. We extend that work to incorporate: further linguistic analysis; contextual analysis based on the properties of the concepts in the ontology and the relationships among these concepts; extended use of WordNet [4] to include the search of not only synonyms but also antonyms, plurals, hypernyms, and hyponyms; use of the Java WordNet Library API [13] for performing run-time access to the dictionary; integration of heuristic knowledge into the contextual analysis phase; and reasoning rules based on the relationships among ontology concepts and each concept's property list. A set of experiments and corresponding evaluation have been carried out to show the promising result.

The paper is organized as follows. Related work is discussed in Section 2. Section 3 overviews the **PUZZLE** system, whose details are described in Section 4. Section 5 evaluates the experiments conducted and gives some analysis. Finally, conclusion and future work are mentioned in Section 6.

2. Related Work

Much research has been carried out in ontology matching, mostly using one of two approaches [10]: instance-based and schema-based. All of the following systems belong to the latter, except for GLUE [6].

GLUE 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 an insignificant number of instance or no instances at all.

In [14], a method is investigated for agents to develop local consensus ontologies to help in communications within a multiagent system of B2B agents. This work shows the potential brought by local consensus ontologies in improving how agents conduct B2B Web service discovery and composition. It also explores the influence of a lexical database in ontology merging. However, it does not take into consideration the properties of ontology concepts.

PROMPT [9] 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.

COMA [11] provides an extensible library of matching algorithms, a framework for combining results, and evaluation platform as well. According to their evaluation, COMA is performing well in terms of precision, recall and overall measures. Although being a composite schema matching tool, COMA does not integrate reasoning and machine learning techniques.

Similarity Flooding [8] 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.

Cupid [12] 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. As a result, there are many limitations to its application, because a tree cannot represent multiple-inheritance, an important characteristic in ontologies.

For HELIOS [5], 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.

S-Match [7] 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, like Cupid, S-Match uses a tree-structured ontology.

3. Overview of Our Solution

The goal of our work is to construct a correctly merged 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. Similarly, 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 merged ontology from a number of ontologies, we take two ontologies and merge them into a new one, and then we iteratively merge the resultant ontology with each additional one. We show that the order by which the ontologies are to be merged will not affect the final result—the only difference lies in the intermediate resultant ontologies. If we choose the original ontologies in an optimal order, we will have a monotonically incremental intermediate ontology for each step. However, the eventually merged ontology will be exactly the same regardless of the order we choose. We will explain next our method for merging two ontologies.

We represent an ontology using a directed acyclic graph. In order to merge two ontologies, G_1 and G_2, we try to relocate each concept (node) from one ontology into the other one. Figure 1 shows this merging procedure.

First we introduce the *relocation value* of a target concept C against any other concept C'. A relocation value is a value from 0 to 1, reflecting the likelihood of correctly relocating a concept. As the equation below indicates, a relocation value is calculated as the weighted sum of the values from linguistic matching and contextual matching.

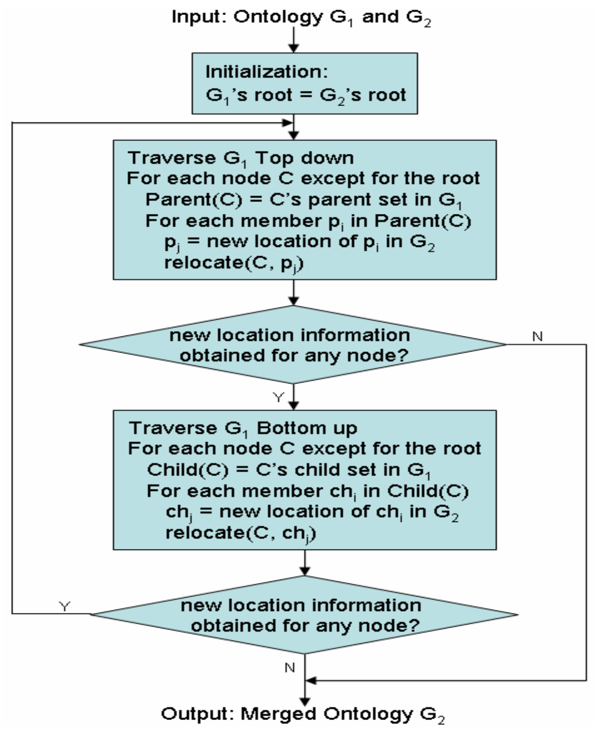
$$\textit{relocation value} = \mathrm{w}_{\mathrm{linguistic}} * v_{\mathrm{linguistic}} + \mathrm{w}_{\mathrm{contextual}} * v_{\mathrm{contextual}}$$

When trying to match concepts, we consider both linguistic and 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. That is, $w_{\text{linguistic}}$ is set to 0.7 and $w_{\text{contextual}}$ is set to 0.3 in the above equation. 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.

From all the candidate concepts in the destination graph *G*, we build a list of candidate concepts for each type of relationship of *C* (see details in Section 4.1). Within each list, we calculate the relocation value of *C* against each concept in that list, and then choose the one producing the highest value. After we finish processing all candidate lists, we have sufficient information to be able to relocate *C*.

4. Details of the PUZZLE System

As the flow chart in Figure 1 indicates, the relocation for all concepts as a whole will be repeated until no new information regarding the ontology schema is found. Firstly, we adopt a top-down 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 relocated into the suitable place(s) in the destination graph G_2 before the relocation of *C* itself. Based on the information from both *C* and its parent set, we relocate *C* in G_2. After every concept has been relocated, we traverse G_1 again, bottom-up this time, trying to obtain new information about the relocation of *C* according to the new location(s) of its child(ren). Then we repeat the above process until there is no more information found for any concept. The idea behind this repeating relocation process is that the correct new location of each concept depends on both its linguistic and contextual features, while the latter comprises a concept's properties and its relationship(s) with other(s). The most important relationship mentioned here is the *subclass-superclass* relation. Therefore, the new locations of both parent(s) and child(ren) are critical in determining where to relocate a concept. Moreover, the newly obtained information about the relocation of parent(s)/child(ren) will affect the relocation of child(ren)/parent(s). As a result, we need this iterative process to guarantee the most suitable new location of each concept.



Input: Ontology G_1 and G_2
Initialization: G_1's root = G_2's root
Traverse G_1 Top down
For each node C except for the root
Parent(C) = C's parent set in G_1
For each member p_i in Parent(C)
p_j = new location of p_i in G_2
relocate(C, p_j)
new location information obtained for any node?
N
Y
Traverse G_1 Bottom up
For each node C except for the root
Child(C) = C's child set in G_1
For each member ch_i in Child(C)
ch_j = new location of ch_i in G_2
relocate(C, ch_j)
new location information obtained for any node?
Y
N
Output: Merged Ontology G_2

Figure 1. Top level procedure of PUZZLE system

4.1. Linguistic Matching

The linguistic factor reflects how the ontology designer wants to encode the meaning of a concept by choosing a preferable name for it. **PUZZLE** uses both string and substring matching techniques when performing linguistic feature matching. Furthermore, we integrate WordNet by using the JWNL API in our system. In this way, we are able to obtain the synonyms, antonyms, hyponyms, and hypernyms of an English word, which has been shown to increase the accuracy of linguistic matching dramatically. In addition, WordNet performs some preprocessing, e.g., the transformation of a noun from plural form to singular 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 $v_{\text{linguistic}} = 0$;
- *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 $v_{\text{linguistic}} = 1$;

- *sub-match*: N_C is either a postfix or a hypernym of $N_{C'}$, with the matching value $v_{\text{linguistic}} = 1$;
- *super-match*: $N_{C'}$ is either a postfix or a hyponym of N_C, with the matching value $v_{\text{linguistic}} = 1$;
- *leading-match*: the leading substrings from N_C and $N_{C'}$ match with each other, with the matching value $v_{\text{linguistic}}$ equaling the length of the common leading substring divided by the 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*: the matching value $v_{\text{linguistic}} = 0$.

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.

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). The latter is not expressed explicitly in any formula. Instead, we integrate the relationship factor into our system by three reasoning rules specified in Section 4.3.

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 $v_{\text{contextual}}$ between them.

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

v_{required} and $v_{\text{non-required}}$ are the similarity values calculated for the *required* property list and *non-required* property list, respectively. w_{required} and $w_{\text{non-required}}$ are the weights assigned to each list. In this paper, we choose 0.7 and 0.3 for w_{required} and $w_{\text{non-required}}$. v_{required} and $v_{\text{non-required}}$ are calculated by the same procedure. (In future research we will investigate the sensitivity of our results to this choice of weights.)

Suppose the number of properties in two property lists (either *required* or *non-required* ones), 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 $v_{\text{linguistic}} \geq$ threshold; 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 reasoning. 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 $v_{\text{linguistic}} \geq$ threshold; 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*.

After we find all the possible matching models in the above order, we can calculate the similarity value *v* between the property lists as

$$v = (v_1 * w_1 + v_2 * (w_2 + w_2' * f_1) + v_3 * (w_3 + w_3' * f_2))/n_1$$

where:
- the value range of *v* 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.

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 relationship axioms [3]: *subClassOf*, *equivalentClass*, *disjointWith*, *sameIndividualAs*, *differentFrom*, *subPropertyOf*, *equivalentProperty*, *inverseOf*, *transitiveProperty*, *functionalProperty*, and *inverseFunctionalProperty*. The first three axioms will be used as follows.

The *subClassOf* axiom will represent *subclass-superclass* relationship. The *equivalentClass* axiom will be used for specifying the *equivalentclass* relationship. As for *sibling* relationship, there is no direct support from OWL axioms. However, the *disjointWith* axiom is a good choice, given the condition that each ontology is reasonably designed. That is, we make an assumption that under a same parent class, all the siblings within the same level will be disjoint with each other. Otherwise, a new *superclass* should be added for those siblings with intersection.

4.3. Reasoning Rules

PUZZLE uses three domain-independent rules, each regarding the relationship among ontology concepts, to incorporate the reasoning into our system. As mentioned in Section 3, the merging of two ontologies is basically the relocation of every concept from one ontology into the other one. By the use of reasoning rules, new information about the relocation of a concept is inferred. This self-learning greatly increases the accuracy of our system. In that sense, our system is a self-adaptive one. In addition, these rules are applied to concepts from different ontologies. Therefore, we refer to them as *inter-ontology reasoning*. As mentioned before, a concept's relationships with others play an important role in relocating itself into suitable place(s). This role is not reflected in the form of formula, but we embody it via the reasoning rules discussed in this section.

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. $v_{\text{contextual}} \geq$ threshold

  We denote the corresponding concepts $n_i(A)$ and $n_j(B)$ by $n_i(A) \xleftrightarrow{p} n_j(B)$;
- $P(n_i(A))$ is a subset of $P(n_j(B))$ if and only if
  i. $s_i \leq s_j$, and
  ii. $v_{\text{contextual}} \geq$ threshold

  We denote the corresponding concepts $n_i(A)$ and $n_j(B)$ by $n_i(A) \xrightarrow{p} n_j(B)$;
- $P(n_i(A))$ is a superset of $P(n_j(B))$ if and only if
  i. $s_i \geq s_j$, and
  ii. $v_{\text{contextual}} \geq$ threshold

  We denote the corresponding concepts $n_i(A)$ and $n_j(B)$ by $n_i(A) \xleftarrow{p} n_j(B)$;
- Other.

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

[**Rule 1**] This rule straightforwardly states 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, considering the possibility of the existence 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 markedly.

- 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. $n_{i2}(A) \xleftrightarrow{p} n_{k2}(B)$ and (the names of $n_{i2}(A)$ and $n_{k2}(B)$ have either an *exact-match*, or a *leading-match* with $v_{\text{linguistic}} \geq$ threshold)
  2. $n_{i2}(A) \xrightarrow{p} n_{k2}(B)$ and the name of $n_{k2}(B)$ is a *sub-match* of the name of $n_{i2}(A)$
  3. $n_{i2}(A) \xleftarrow{p} n_{k2}(B)$ 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 from two ontologies have no direct relationships between them, we consider a third ontology to see if it can provide a 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. Essentially, the idea of a semantic bridge is implemented via this rule.

- 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. $n_{i1}(A) \xleftrightarrow{p} n_{k1}(B)$ and (the names of $n_{i1}(A)$ and $n_{k1}(B)$ have either an *exact-match*, or a *leading-match* with $v_{\text{linguistic}} \geq$ threshold)
  2. $n_{i1}(A) \xrightarrow{p} n_{k1}(B)$ 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 the 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. Experiments and Discussion of Results

In this section we describe a set of experiments conducted with the purpose of evaluating the ability of **PUZZLE** to generate a correctly merged ontology. We collected a set of ontologies and evaluated **PUZZLE** in terms of precision, recall, and merging convergence. We discuss the results obtained.

5.1. Experimental Setup

- Test ontologies
  A collection of sixteen ontologies for the domain of "Building" were constructed by graduate students in computer science at our university and used for evaluating the performance of **PUZZLE**. The characteristics of these ontology schemas are summarized as: having between 10 and 15 concepts with 19 to 38 properties and 31 to 49 relationships among the concepts.

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 align their local ontologies to form a merged one.

5.2.1. Evaluation of the Resultant Ontology. To decide whether a correctly merged ontology is obtained, we asked two ontology experts to carry out a manual mapping and we compared their results with ours. Both *precision* and *recall* measurements are applied in the evaluation during the process of merging ontologies one at a time. The evaluation result is shown in Figure 2. 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".

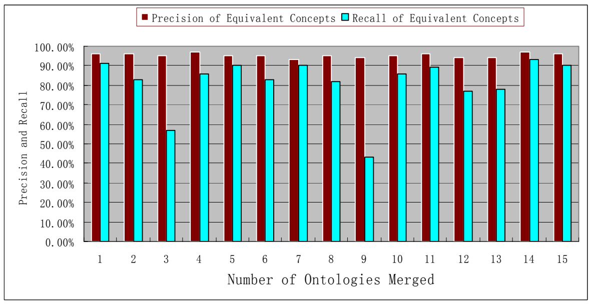


Figure 2. Precision and recall measurements of the resultant ontology

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 4 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 3. 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.

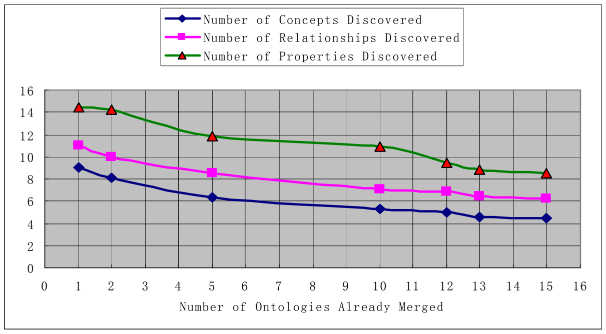


Figure 3. Merging convergence experiment

6. Conclusion and Future Work

Ontology matching is an important foundation in the Semantic Web interoperation. In this paper, we present the **PUZZLE** system, a schema-based approach combined with inter-ontology reasoning, which self-learns to merge/align 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 reasoning with several matching techniques, and incorporates inter-ontology reasoning to implement the idea of semantic bridge. **PUZZLE** provides a solid base for the seamless integration of Web services. A set of experiments showed a promising result from this system.

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 state-of-art ones in ontology matching, by using more general ontology libraries.

References

[1] M.P. Singh, and M.N. Huhns, *Service-Oriented Computing Semantics, Processes, Agents*, 1st edn. Wiley, Chichester, England, 2005.

[2] L. Stephens, A. Gangam, and M.N. Huhns, "Constructing Consensus Ontologies for the Semantic Web: A Conceptual Approach", *World Wide Web Journal*, *Vol. 7*, *No. 4*, Kluwer Academic Publishers, Norwell, MA, USA, 2004, pp. 421-442.

[3] W3C, "OWL Web Ontology Language Reference", http://www.w3.org/TR/owl-ref, 2004.

[4] A.G. Miller, "WordNet: A Lexical Database for English", *Communications of the ACM*, *Vol. 38*, *No. 11*, ACM Press, New York, NY, USA, 1995, pp. 39-41.

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

[6] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy, "Learning to match ontologies on the Semantic Web", *The VLDB Journal (2003)*, *Vol. 12*, Springer-Verlag, New York, NY, USA, 2003, pp. 303-319.

[7] F. Giunchiglia, P. Shvaiko, and M. Yatskevich, "S-Match: an algorithm and an implementation of semantic matching", *Proceedings of the 1st European Semantic Web Symposium*, *Vol. 3053*, Springer-Verlag, New York, NY, USA, 2004, pp. 61-75.

[8] S. Melnik, H. Garcia-Molina, and E. Rahm, "Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching", *Proceedings of the 18th International Conference on Data Engineering*, IEEE Computer Society Press, Washington DC, USA, 2002.

[9] N.F. Noy, and M.A. Musen, "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment", *Proceedings of the17th National Conference on Artificial Intelligence (AAAI 2000)*, AAAI Press, Menlo Park, CA, USA, 2000.

[10] E. Rahm, and P.A. Bernstein, "A survey of approaches to automatic schema matching", *The VLDB Journal (2001)*, *Vol. 10*, Springer-Verlag, New York, NY, USA, 2001, pp. 334-350.

[11] H. Do, and E. Rahm, "COMA – A system for flexible combination of schema matching approaches", *Proceedings of the 28th VLDB Conference*, Springer-Verlag, New York, NY, USA, 2002.

[12] J. Madhavan, P.A. Bernstein, and E. Rahm, "Generic Schema Matching with Cupid", *Proceedings of the 27th VLDB Conference*, Springer-Verlag, New York, NY, USA, 2001.

[13] JWNL, "Java WordNet Library – JWNL 1.3", http://sourceforge.net/projects/jwordnet/, 2003.

[14] A. Williams, A. Padmanabhan, and M.B. Blake, "Local Consensus Ontologies for B2B-Oriented Service Composition", *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, *Session: Ontologies*, ACM Press, New York, NY, USA, 2003, pp. 647-654.

[15] R.L. Zavala Gutiérrez, and M.N. Huhns, "On Building Robust Web Service-Based Applications", *Extending Web Services Technologies: The Use of Multi-Agent Approaches*, Kluwer Academic Publishing, Norwell, MA, USA, 2004.