Reconciling Agent Ontologies for Web Service Applications

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. Because there is still no agreed-upon global ontology, Web services supplied by different providers typically have individual and unique semantics, described by independently developed ontologies. The seamless connection of these distributed Web services for business-to-business applications depends heavily on reconciling the disparate semantics, possibly by integrating the ontologies. In this paper, we describe an approach to reconcile ontologies from distributed Web services. Our approach is totally automated, and features the following: i) alignment of the ontologies is performed without previous agreement on the semantics of the terminology in each ontology; ii) both linguistic and contextual features are considered; iii) the use of WordNet for linguistic analysis; iv) integration of heuristic knowledge for contextual analysis; and v) inference of new relationships by applying several rules based on domain-independent relationships and property lists. Experiments have been carried out to show the promising results of our system.

1 Introduction

Web service applications, such as supply-chain purchase orders and automated order enactment, have been shown to offer great potential value to businesses. Initial on-line automation activities were tightly coupled in the sense that business partners predefined the terms of their interaction using standards such as EDI and XML [12]. Recently, the emergence of Web services has led the software industry into a service-oriented approach to software development. Service-oriented computing is a loosely coupled methodology, based on the use of standard protocols (UDDI for discovery, WSDL for description, BPEL4WS for coordination, and SOAP for communication). The use of Web services provides greater flexibility with respect to the interoperability, reuse, and development of applications in a distributed environment.

Although there can be some value in accessing a single Web service through a semantically well-founded interface, a greater value is clearly 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 connection of distributed Web services becomes increasingly critical. However, due

to the lack of an agreed-upon global ontology, Web services from different providers typically have heterogeneous semantics. Agents that automatically reconcile ontologies, and thereby understand and integrate the information from different sources, would greatly facilitate Web service-based application interoperability.

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 implements an approach to construct a merged ontology from distributed and independently designed ontologies. We also explain the potential application of our system in Web service-based transactions. We assume that: 1) we are dealing with Web services for similar domains; 2) ontological representations have been derived from Web service documentations, e.g., WSDL and SOAP specifications; and 3) agents are willing to communicate with each other to reach consensus among 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 [10] to include the search of not only synonyms but also antonyms, plurals, hypernyms, and hyponyms; use of the Java WordNet Library API [9] for performing run-time access to the dictionary, instead of having to initialize the synonym set 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 rest of the paper is organized as follows. Section 2 briefly discusses related work in ontology matching. An overview of the **PUZZLE** system is given in Section 3. Section 4 describes the details of our system. Section 5 reports the experiments conducted and analyzes the results, and 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 [7]: instance-based and schema-based. All of the systems mentioned below belong to the latter, except for GLUE [8].

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 instances or no instances at all.

In [3], 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.

Cupid [5] 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 [11], 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 [4] 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.

3 Overview of Our Solution

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, *G1* and *G2*, can be related indirectly through a semantic bridge consisting of other previously unrelated ontologies, even when there is no direct relationship between *G1* and *G2*. 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 *G1* and *G2*, the more ontologies the semantic bridge comprises, the better the semantic match between *G1* and *G2*.

In order to construct a consensus 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 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. We adopt a breadth-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.

Firstly, we address the issue of 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 equation 1 below indicates, a

relocation value is calculated as the weighted sum of the values from linguistic matching and contextual matching.

$$relocation\ value = w_{linguistic} * v_{linguistic} + w_{contextual} * v_{contextual}. \quad (1)$$

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, $w_{linguistic}$ is set to 0.7 and $w_{contextual}$ is set to 0.3 in equation 1. 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.

We claim that there are five mutually exclusive relationships between any two concepts (see details in Section 4.2.2). 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

The following pseudocode describes the top level procedure of our algorithm.

```
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 p1 in Parent(C)
             pj = new location of p1 in G2
                relocate(C, pj)
          end for
    end for
  end
```

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). Notice that the latter is not explicitly expressed in any formula. Instead, we integrate the relationship factor into our system by the 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.

$$v_{\text{contextual}} = w_{\text{required}} * v_{\text{required}} + w_{\text{non-required}} * v_{\text{non-required}} . \quad (2)$$

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.

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 $v_{\text{linguistic}} \geq 0.9$; 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 $v_{\text{linguistic}} \geq 0.9$; 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*.

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

$$v = (v_1 * w_1 + v_2 * (w_2 + w_2' * f_1) + v_3 * (w_3 + w_3' * f_2))/n_1 . \quad (3)$$

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 [6]: *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. 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. $v_{\text{contextual}} \geq 0.9$

  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. $v_{\text{contextual}} \geq 0.9$

  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. $v_{\text{contextual}} \geq 0.9$

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

Rules 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, 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 dramatically increase.

- 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 $v_{\text{linguistic}} \geq 0.65$)
  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 relationships between them, we consider a third one 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.

- 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 $v_{\text{linguistic}} \geq 0.65$)
  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. $n_{i1}(A) \overset{p}{\longleftarrow} n_{k1}(B)$ 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 Our Results

First, we envision the following example application of **PUZZLE** in Web service-based transactions. In the domain of real estate, there might be many reasons why different agencies would like to communicate with each other. Consider the case where a real estate agent did not initially find any housing matching a client's requirements. It would be helpful if that agent directly connected to other agents and found something for the client, instead of sending the client away to find another agent. The ability of an agent to reach other potential suppliers would lead to better service, less work for the client, and ultimately happier clients. Another situation is that several agencies might want to put together a unified interface to users, so that all of them together offer a wider range of options to clients. In order to carry out communications among agencies without the need to agree on predefined data interchange formats, the agencies can benefit from automated ontology matching abilities as provided by **PUZZLE**.

In this section, we describe a set of experiments we conducted, whose purpose was to determine whether or not **PUZZLE** generates a consensus ontology. We evaluate **PUZZLE** in terms of precision, recall, and merging convergence.

5.1. Experimental Setup

- Test ontologies
  Three sets of ontologies in three different domains, i.e., "Building", "Human", and "Sports" were used for evaluating the performance of the **PUZZLE** system. They were constructed by graduate students in computer science and engineering at our university. There are 16 ontologies for the domain of "Building", having between 10 and 15 concepts with 19 to 38 properties and 31 to 49 relationships among the concepts. For the other 2 domains, no property was defined for any concept. We have 54 ontologies for the domain of "Human", with between 7 and 28 concepts; and 23 ontologies for the domain of "Sports", with between 4 and 22 concepts.

5.2. Experimental Results and Analysis

Our experiments simulate having a set of 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. 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 1. Due to the space limit, we only show the result for "Building" domain and omit the other two. Notice that this result is not statistically valid but indicative. Both measurements reflect a promising result, especially for "Building" domain. For "Human" and

"Sports" domain, the results are not as good as that of "Building" domain. The reason is straightforward. Although in Section 3 we mention that our experiment ontologies have less contextual information than linguistic one, we claim that contextual factor does play an important role in determining the mapping among ontology concepts. That is the reason we chose ontologies of both with and without properties in the experiment. The result verifies our claim.

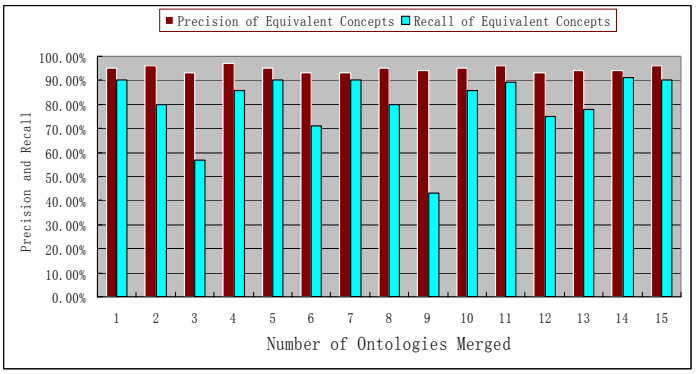

Figure 1. Precision and Recall Measurements of Resultant Ontology for "Building"

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 at certain points during the merging process. For different domain, the testing points chosen are different. For example, in "Building" domain we picked up the points when the first, second, fifth, tenth, twelfth, thirteenth, and fifteenth ontologies were merged. For the other two domains, please refer to figure 2-b and 2-c, which together with figure 2-a, show the results of this hypothesis-verifying experiment.

Out of the 16 ontologies in "Building" domain 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. The experiment data in "Human" and "Sports" domains was treated in the same way.

A monotonically decreasing pattern is shown in Figure 2-a. 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. In figure 2-b and 2-c, the similar monotonically decreasing pattern is found. However, the converge tendency is not so obvious, comparing to that in figure 2-a. In "Building" domain, when the last ontology was being merged, the number of newly discovered concepts is around 37% of that number when the 0^{th} ontology being merged, i.e., at the very beginning of the merging process. The corresponding percentages in the "Human" and "Sports" domains are 65% and 74%, respectively. This is again due to a lack of property information. In fact, sometimes it is even difficult for ontology experts to determine a potential mapping in the absence of a property list.

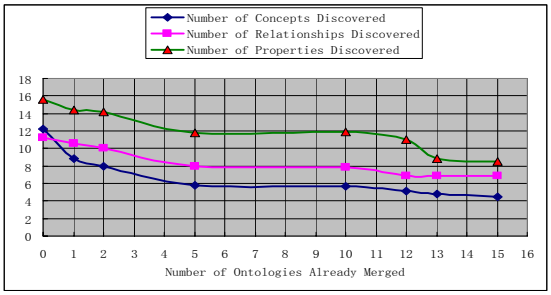

Figure 2-a. Merging Convergence Experiment for "Building"

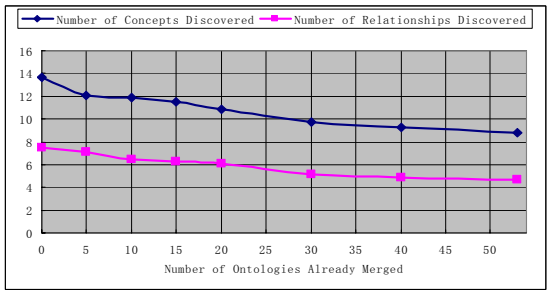

Figure 2-b. Merging Convergence Experiment for "Human"

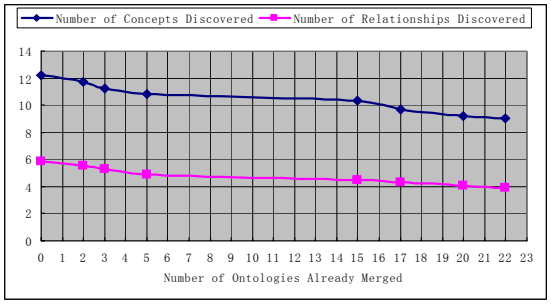

Figure 2-c. Merging Convergence Experiment for "Sports"

6 Conclusion and Future Work

Ontology matching is a critical operation in the Semantic Web, especially for business-to-business applications. In this paper, we presented the **PUZZLE** system, a schema-based approach combined with inter-ontology reasoning, which learns to reconcile 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.

Future work includes: adopting machine learning techniques to make agents more intelligent; considering other relationships, such as *partOf*, *hasPart*, *causeOf*, and *hasCause*; integrating the OWL Validator into our system; and testing our method against other well-known ones in ontology matching, by using more general ontology libraries.

References

1. Singh, M. P., and Huhns, M. N.: Service-Oriented Computing Semantics, Processes, Agents. 1st edn. Wiley (2005)
2. Stephens, L., Gangam, A., and Huhns, M. N.: Constructing Consensus Ontologies for the Semantic Web: A Conceptual Approach. In: World Wide Web Journal, Vol. 7, No. 4. Kluwer Academic Publishers (2004) 421 – 442
3. Williams, A., Padmanabhan, A., and Blake, M. B.: Local Consensus Ontologies for B2B-Oriented Service Composition. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems. ACM Press (2003) 647 – 654
4. Giunchiglia, F., Shvaiko, P., and Yatskevich, M.: S-Match: an algorithm and an implementation of semantic matching. In: Proceedings of the 1st European Semantic Web Symposium, Vol. 3053. Springer-Verlag (2004) 61 – 75
5. Madhavan, J., Bernstein, P. A., and Rahm, E.: Generic Schema Matching with Cupid. In: Proceedings of the 27th VLDB Conference. Springer-Verlag (2001)
6. W3C: OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref (2004)
7. Rahm, E., and Bernstein, P. A.: A survey of approaches to automatic schema matching. In: The VLDB Journal, Vol. 10. Springer-Verlag (2001) 334 – 350
8. Doan, A., Madhavan, J., Dhamankar, R., Domingos, P., and Halevy, A.: Learning to match ontologies on the Semantic Web. In: The VLDB Journal, Vol. 12. Springer-Verlag (2003) 303 – 319
9. JWNL: Java WordNet Library – JWNL 1.3. http://sourceforge.net/projects/jwordnet/ (2003)
10. Miller, A. G.: WordNet: A Lexical Database for English. In: Communications of the ACM, Vol. 38, No. 11. ACM Press (1995) 39 – 41
11. Castano, S., Ferrara, A., Montanelli, S., and Racca, G.: 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. IEEE Computer Society Press (2004) 360 – 366
12. Zavala Gutiérrez, R. L. and Huhns, M. N.: On Building Robust Web Service-Based Applications. In: Extending Web Services Technologies: The Use of Multi-Agent Approaches. Kluwer Academic Publishing (2004)