Building Application Ontologies from Descriptions of Semantic Web Services

Xia Wang, Tomas Vitvar, Manfred Hauswirth, and Doug Foxvog
Digital Enterprise Research Institute (DERI)
National University of Ireland Galway, Ireland
{firstname.lastname}@deri.org

Abstract

Different ontologies used in semantic web services fields raise numerous interoperation and communication problems with respect to service discovery, composition, and execution. The current approaches for ontology mediation often failed due to their lack of sufficient semantic expressiveness and reasoning capability. In this paper[1], we present a novel approach allowing ontologies to provide self-contained semantics for service applications. We show how desired application ontologies can be generated using a new merging algorithm for service ontologies. We also show some experimental results and compare them to the output of the PROMPT ontology merging tool.

1. Introduction

When developing a semantic web service (SWS) [21], an *application ontology* is necessary for the description of a service. This application ontology is a knowledge foundation not only for defining particular services, but also fostering the understanding between machines with services and services with services. Unfortunately, uniform application ontology for similar services does not exist.

Currently, developers of SWS often describe their services locally and independently by using their own ontologies. As a result, different ontologies are barriers between heterogeneous services, and cause numerous interoperation communication problems. Especially, during the processes of service discovery [9] and service composition [11], the diversity of service ontologies makes it difficult and mismatched often.

One of the solutions to this problem is to build an uniform *application ontology*. However, to manually create an ontology is difficult and time-consuming work. Because building an ontology usually requires highly professional knowledge, and in large-scale environments ontologies may be incomplete and likely to change often. Moreover, defining ontologies generally duplicates effort which leads not only to wasted human labor, but also increased difficulties of service interoperation since ontologies are often created based on developers' domain knowledge without using standards. Therefore, automatically generating an ontology with standards is desired.

Aiming to solve the above problems, we propose a novel method to build a uniform application ontology by merging the existing heterogeneous ontologies of semantic service descriptions. The potential benefits of a uniform application ontology are: 1) Preventing repeated work; 2) Formalizing ontology generation; 3) Standardization of ontology manipulations; 4) Better service development; 5) Overcoming many service heterogeneities caused by improper ontology matches;

The solution of this paper is similar to learn a domain ontology from text or semantic web [14], [17], and [24]. Unfortunately, these work can not be reused in the SWS context directly, as they mainly focus on building ontologies from texts, and rely heavily on Natural Language Processing and inductive learning. Semantic services are, however, defined by well-structured and schema-based ontology languages, such as OWL-S [25] or WSMO [4], which are not natural language descriptions. In a summary, our semantic services ontologies merging algorithm 1) imports individual ontologies, 2) uses WordNet as a knowledge base to disambiguate word senses and extract concepts relations, 3) follows a new rule-based ontology merging algorithm to build application ontologies, and 4) cleans up inconsistencies, if any.

2. Related Work

Building an application ontology can be regarded as service ontologies merging problem in SWS context.

Several heuristics were described to identify corresponding concepts in different ontologies, e.g., comparing the names of two concepts, comparing the natural language

[1]This work is supported by the Science Foundation Ireland Grant No. SFI/02/i131, and the EU projects Knowldege Web (FP6-507482).

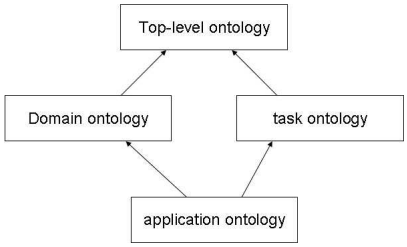

Figure 1: Different Levels of Ontologies and Their Relationships

definitions of two concepts by linguistic techniques, and checking the closeness of two concepts in a concept hierarchy [15]. Recently, other approaches relying on syntactical and semantic matching heuristics were proposed, including ONIONS [6], Chimaera [5], PROMPT [18], and FCA-merge [22]. Just taking PROMPT as an example, it is a semi-automated method for ontology merging and alignment embedded in Protege-2000 developed by the Stanford Medical Informatics Group. Under continuous user interaction of selecting suggested operations, it merges (including merging classes, meta-classes, slots, bindings between a slot and a class, or deep/shallow copying a class from one ontology), incorporates changes, and finds conflicts in result ontologies.

In the literature, there is a consensus on the lack of automated methods for acquiring domain ontologies [14], [17], [24], and [10]. There are some similar work on ontology merging supported by WordNet, which is used to enrich domain concepts by finding an appropriate WordNet concept and their concept relations. As Cho et al. [12] introduced an ontology merging technique using WordNet based on a horizontal approach (for mapping between ontologies through integrated similar concepts on the same level) and a vertical approach (creating rules from similarity measures between concepts at different levels).

In summary, the shortcomings of the current approaches are: 1) lack of a standard for merging and dealing with conflicts, 2) semi-automation when human involvement is needed, 3) only light-weight ontologies support, i.e., not considering concept axioms and constraints sufficiently well, if at all, and 4) too few contributions to further usage, such as creating a domain knowledge base providing powerful reasoning capabilities.

3. Levels of Ontologies

In work [8], different generality levels of ontologies were suggested as shown in Fig. 1. Our work only distinguishes *generic ontologies*(which describe very general concepts which are independent of particular problems or domains), *domain ontologies* $\mathcal{DO}$ (which describe vocabularies related to generic domains by specializing concepts introduced in top-level ontologies.), and *application ontologies* $\mathcal{AO}$ (which defines concepts in applications often corresponding to roles played by domain entities while performing certain activities in applications.). In the context of SWS, several major generic ontologies for semantic services have been proposed, such as OWL-S and WSMO; and application Ontology may involve several domain ontologies to describe a certain application, as $\mathcal{AO} \subset \bigcup_{i=1}^{m} DO_i,\ i, m \in N$.

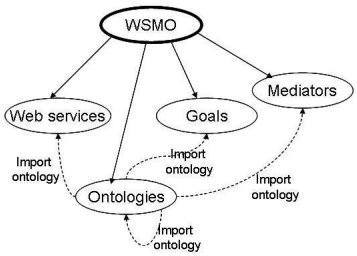

Figure 2: Upper WSMO Elements

We also extend a term *service ontology* (noted as $\mathcal{SO}$) as a piece of ontology used by a single service for service description, then there holds $\mathcal{SO} \subseteq \mathcal{AO}$. Despite any two $\mathcal{SO}$s describing the same domain intersect or not, the idea of building an application ontology is $\mathcal{AO} \doteq \bigcup_{i=1}^{n} SO_i,\ n \to \infty$.

4. Generic Ontologies of SWS

Without losing generality we use the WSMO as semantic Web service ontology model for descriptions of our scenarios. WSMO was devised as an ontology to describe various aspects related to semantic web services. There are four top-level elements in WSMO: *Ontologies* provide the terminology used by the other WSMO elements to describe the relevant aspects of the domains of discourse; *Web Services* describe the computational entities providing access to services that deliver some value in a domain, mainly including capabilities and interfaces; *Goals* represent user desires, for which fulfillment could be sought by executing a web service; and *Mediators* describe elements that overcome inter-operability problems (at the data, process, and protocol levels) between different WSMO element. As depicted in Fig. 2, element *Ontologies* is imported by the other three elements in WSMO as concept reference.

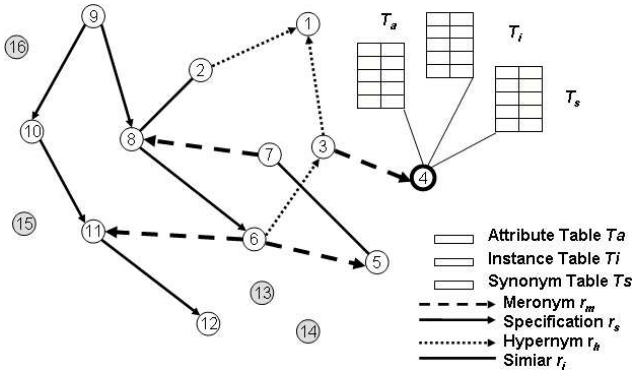

Figure 3: Conceptual Ontological Semantic Net

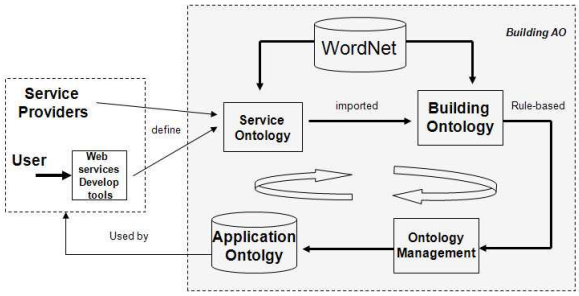

Figure 4: Application Ontology Builder

5. Conceptual Model of Application Ontologies

5.1 Definition of Ontologies

In the SWS context, we reused and simplified the ontology definitions of the University of Karlsruhe [13].

Definition 1 [Ontology with Datatypes]: An ontology with datatypes is a structure $O := (C, \leq_C, T, R, \sigma_R, \leq_R, A, I)$, consisting of a set of concepts C aligned in a hierarchy $\leq_C$, a set of relations R with $\leq_R$, the signature $\sigma_R : R \rightarrow C \times C$, a set of datatypes T with type transition functions, a set of attributes A and instances I, and the signature $\sigma_A : A \rightarrow C \times T$. For a relation $r \in R$, we define its domain and its range by $dom(r) := \pi_1(\sigma_R(r))$ and $range(r) := \pi_2(\sigma_R(r))$.

This definition is extended by four concept relations with fuzzy weights, see Section 5.2. For measuring of ontology similarity $sim(\mathcal{O}_1, \mathcal{O}_2) \in [0, 1]$, we consider concepts, concept relations, types and attributes. The concept instance similarity is out of this paper.

5.2 Ontological Concept Semantic Net

Extended with multiple concept relations, the application ontology can not be structured as a tree of concepts limited by semantic categories or simply single concept relation. We represent our application ontology as a semantic net.

Therefore, application ontology $(\mathcal{AO})$ is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{c_1, c_2, ..., c_n\}$ is a node set of ontological concepts with their concept type set c_T, and $\mathcal{E} = \{e_1, e_2, ..., e_m\}$ is an edge set, in which each edge is denoted by a binary relation of two concepts.

In Fig. 3, a semantic net with 16 connected nodes and 4 isolated nodes (in *grey* color) is depicted. The semantic net has 4 concept relations, *Specialization/Generalization*(*is-a* relation depicted as a solid direct edge, r_s), *Holonym/Meronym* (*partOf/memberOf* relation depicted as a dashed direct edge, r_m), *Hypernym/Hyponym* (*kindOf* relation depicted as a dotted direct edge, r_h), and *Similar* (including *Synonym*, r_i) depicted as a line annotated with a similarity value $sim(c_1, c_2) \in (0, 1]$. If $sim(c_1, c_2) = 1$, then the concepts $c_1,\ c_2$ are synonyms. In addition, a concept c_i may have an attribute table T_{Ai}, instance table T_{Ii}, and synonym table T_{Si}.

6. Building Application Ontologies

In order to build meaningful a $(\mathcal{AO})$, a knowledge base, a thesaurus or dictionary is necessary. WordNet is a quite mature conceptual thesaurus with hierarchally structured organisation and multiple semantic relations between concepts. It is a good choice also due to its broad concept structure of synonym sets.

6.1 Process of Building $(\mathcal{AO})$

We assume there are *k* semantic services from the same application domain as the inputs of $(\mathcal{AO})$ builder. As depicted in Fig. 4, the building process is iterative including following steps.

1. Initialize $\mathcal{AO}_0$ as an empty set, $\emptyset$.
2. Import a $SO_i,\ 1 \leq i \leq k$; re-organise it into the ontology structure defined by Definition 1.
3. For each concept of $\mathcal{SO}_i$, extract its relative concept information from WordNet by the *WCEA* algorithm of Section 6.2; add the retrieved information to $\mathcal{SO}_i$, then get a new $\mathcal{SO'}_i$ with more concept relations.
4. Merge $\mathcal{SO'}_i$ and $\mathcal{AO}_{i-1}$ using the *WOMA* algorithm defined in Section 6.3 to obtain a new $\mathcal{AO}_i$.
5. Manage $\mathcal{AO}_i$, including cleaning concept conflicts, updating and storing (see Section 6.4).
6. Repeat steps 2 to 5, until $i = k$, then stop.

The output of this process is an application ontology $\mathcal{AO}_k$, which can be referred to by other service providers or users to create new applications as depicted in Fig. 4.

6.2 WordNet Concept Extraction Algorithm

We assume that a $\mathcal{SO}$ has j concepts, $SO_i = \{C_1, C_2, ..., C_j\}$, $j \in N$. The WordNet Concept Extraction Algorithm (*WCEA*) aims to extract more concept information from WordNet. It involves following steps: (1) perform word sense disambiguation (WSD) on as many concepts as possible (according to Agirre and Rigau's [2] alogrithm), (2) retrieve two kinds of information of the disambiguated concepts from WordNet, which are synonym words and relation words, by executing *add()* operations, e.g., *addSynonym()* and *addRelation()*. In the following, we elaborate on two major points of the *WCEA* algorithm:

1. In the WordNet, many words are polysemous or homonymous (e.g., "Bank"). However, in a specific service, a single sense of a word is intended. Word Sense Disambiguation (WSD) [26] is the task of identifying the intended meaning of a given *target word* from its context. For example, a $SO = \{travel, traintravel, flight, ...\}$, 3 synsets of the word *travel* are returned by the search engine of WordNet 2.1, viz., *(travel, traveling, travelling; #1), (change of location, travel; #2) and (locomotion, travel; #3)*. Thus, the issue is which word sense is intended in this travel service ontology.

   In our work, we use concept density to consider WSD algorithm as discussed in [2, 3, 7]. It was to resolve the lexical ambiguity of nouns by finding the combination of senses from a set of contiguous nouns that maximizes the conceptual density among senses [1].

   As in Fig. 5, word W has four senses and several context words $\{W_1, W_2, W_3, W_4\}$. Each sense of the word belongs to a sub-hierarchy of WordNet. The dots in the sub-hierarchies represent the senses of either the word to be disambiguated (W) or the words in the context. Conceptual density will yield the highest density for the sub-hierarchy containing more senses of those, relative to the total amount of senses in the sub-hierarchy. The sense of W contained in the sub-hierarchy with highest conceptual density will be chosen as the sense disambiguating W in the given context. The sense 2 would be chosen as in Fig. 5.

   Given a concept c, at the top of a sub-hierarchy, and given $nhyp$ (mean number of hyponyms per node), the conceptual density for c, when its subhierarchy contains a number m (marks) of senses of the words to

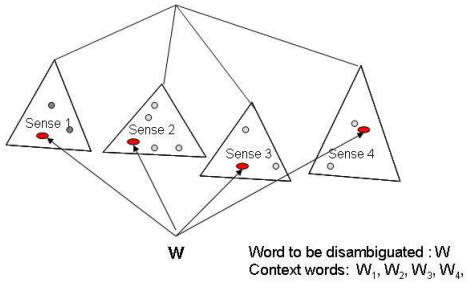

Figure 5: Senses of a word in WordNet [2]

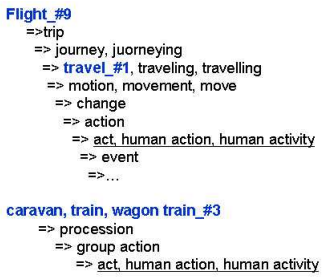

Figure 6: Partial lattice for the example concept

disambiguate, is given by

$$CD(c, m) = \frac{\sum_{i=0}^{m-1} nhyp^{i^{0.2}}}{descendants_c} \quad (1)$$

where the parameter was found that the best performance was attained consistently when the parameter was near 0.2 [2].

For the above travel scenario, first, we adjust it to a new $SO' = \{train\ travel, travel, flight\}$, that $travel(3)$ as the *target word* should be in the middle (the number in parentheses indicates how many senses the word has), the other two words $\{train(6), flight(9)\}$ as window context (meanwhile we remove redundancy words as "travel" from "train travel"), and the target word window is 3. Next, by the WSD algorithm, these three noun concepts are represented by a lattice, with their senses and hypernyms (as Step 1), see Fig. 6.

Then, the conceptual density of concepts of word window are computed referring to WordNet (as Step 2). In our case, only $(act, human\ action, human\ activity)$ has underneath 2 senses to be disambiguated and a sub-hierarchy size of 31 leading to a conceptual density of 0.25806. It is selected as a concept with high density (as Step 3) and, in turn, selects the senses below it as the correct senses of the respective words (as Step 4).

Next, the WSD algorithm proceeds to compute the density for the remaining senses in the lattice, and continues to disambiguate words (back to Step 2, 3 and 4).

When no further disambiguation is possible, the senses left for w are processed and the result is presented (as Step 5). For our case, no more word can be disambiguated. Finally, two words are disambiguated, that is, $\{travel\_\#1, flight\_\#9\}$.

Failure to disambiguate of all concepts in word context occurs in cases of this algorithm. This problem can be resolved by repeated invocation of the algorithm during the ontology building process.

2. After concept disambiguation, the meaning of some concepts are specified. Here, operations *addSynonym()* and *addRelation()* can then be considered.

   Also, *addRelation()* includes 4 relations defined in Section 5.2. Concept association is the criterion for selecting the kind of relation to be added. If two concepts co-occur in one relation in WordNet, and such relation is not defined by the service ontology yet, then add it.

   For example, in the above case we add the synonyms of concept $travel\_\#1$ and $flight\_\#9$, and note that there already has an *is-a* relation between $flight\_\#9$ and $travel\_\#1$ in service ontology, then $addRelation(r_s)$ is not necessary here.

6.3 WordNet-based Ontology Merging Algorithm

When importing a $\mathcal{SO}_j$ and merging it into the current $\mathcal{AO}_i$, the existing structural description of the global ontology merging process [16] can be re-used and extended with WordNet. Preprocessing steps before executing the WordNet-based Ontology Merging Algorithm (WOMA) are as follows:

1. Import a $\mathcal{SO}_j$ and clean words (word stemming and removing); transform $\mathcal{SO}_j$ into semantic net format; organize concept attributes or instances into respective tables; and clean concepts relations.

2. After WSD, to extract concept information of $\mathcal{SO}_j$ from WordNet by executing operations, as *add(synonym)*, *add(attribute)*, *add(instance)*, *delete(relationR)*, and *add(relationR)*.

After these preprocessing, the new $\mathcal{SO'}_j$ is organized as $SO'_j = \{C_{x,0}, C_{y,0}, ..., C_{z,0}, C_{x,y}, ..., C_{0,y}, ..., C_{0,x}\}$ (where x, y, z are variables), the order of which bases the size of *in-degree* (as the left subscript) and *out-degree* (as the right subscript) of concept nodes. That is, concept nodes with 0 *out-degree* are isolated concepts, then concepts are the parents of them, until the concept with 0 *in-degree*. Then, the ontology merging algorithm reads as follows:

- Copy current $\mathcal{AO}_i$.
- Merge concepts of $\mathcal{SO'}_j$ into $\mathcal{AO}_i$, denoted as $\mathcal{AO}_{i+1} = \mathcal{AO}_i \cup SO'_j$.
- Clean conflicts of $\mathcal{AO}_{i+1}$, as name conflicts or taxonomy conflicts.

We assume C_1 and C_2 are two concepts of ontology $\mathcal{AO}_i$ and $\mathcal{SO}_j$, and $C_1 \in \mathcal{AO}_j$, $C_2 \in \mathcal{SO}_i$. Several rules must be followed during the merging process:

Rule 1: If two lexically identical concepts C_1 and C_2 have different types, and there is no transition function between their types, then they are different concepts, $copy(C_2)$ to $\mathcal{AO}_i$; or else they are possibly identical (further referring to Rule 3).

For example, a merged concept with its types would be $(COUNTY\_AREA.acre, \{SquareMeter, Acre\})$, because there is $(1acre = 4,047squaremeters)$.

Rule 2: If C_1 and C_2 are lexically identical, and $T_{a2} \cap T_{a1} = \emptyset$, then they are not identical, and C_2 is added as a new concept.

Rule 3: C_1 and C_2 are not lexically identical, if $T_{a2} \neq \emptyset$, and $T_{a2} \subseteq T_{a1}$, then C_1 and C_2 are synonym and execute **Rule 6**; if $T_{a2} \cap T_{a1} \neq \emptyset$, then C_1, C_1 has an ancestor, but not necessarily latest one; if $T_{a2} \cap T_{a1} = \emptyset$, then C_2 is added as a new concept.

Rule 4 is similar to **Rule 3** to define concept instances.

Rule 5: If C_1 and C_2 are identical, then merge them and their concept tables, and copy C_2's relations from $\mathcal{SO'}_j$.

Rule 6: If C_1 and C_2 are synonym but not identity, then shallow copy concept C_2 by operation $addConcept(C_2, \mathcal{AO}_i)$.

Rule 7: If C_1 is *specific* ($\leq_C$ or $\leq_{\mathcal{R}}$) of C_2, and C_2 *specific* of C_3 in $\mathcal{SO}_j$, but C_1 is *specific* of C_3 in $\mathcal{AO}_i$, then delete the relation between C_1 and C_3, $addRelation(C_1, C_2)$ and $addRelation(C_2, C_3)$.

6.4. Cleaning Conflicts in Ontologies

After the merging and copying operations, it is necessary to check the consistency of the generated ontology and to clean the conflicts occurred. In the related work [19], two levels of semantic conflicts between ontologies were characterized as data level and schema level. Basically, two kinds of consistency will be checked in our work:

- Concept relations conflicts: Checking if there is a relation circle in a new ontology (by the loop check algorithm [19]); Checking if there is a relation conflict *with its original ontology* (by the unification algorithm [19]); Also, if necessary, adding new reasoned relations, e.g., synonym and similar.
- Concept entities conflicts: Are there more identical concepts? Merging the identical concepts into one; Are there different types for one concept? Keeping the

Figure 7: *AirReservation* and *CarRental* ontologies

diversity of types and saving types transition relation; Are there any concepts able to be united? Unite and denote similar concepts.

Based on our knowledge, most of conflicts arise due to the lack of merging rules/standards. Fortunately, some conflicts can automatically be solved by using WordNet as the knowledge reference and our merging rules 1–7.

7. Experiments and Evaluation

To evaluate our work, we created an ontology merging plug-in of PROMPT [18] in Protege environment, which is not only an appropriate tool to implement our merging algorithm, but also a test bed allowing to compare our algorithm to PROMPT's work.

The ontologies used in this experiments are *Air_reservations* and *Car_rental*, both are related to travel service taken from the PROMPT Tool. The *Air_reservations* ontology has 14 concepts, 9 sub-concept relations, 21 slots, and 0 instances in the left part of Fig. 7; while the *Car_rental* ontology has 9 concepts, 4 sub-concept relations, 21 slots, and 0 instances in the right part of Fig. 7.

The evaluation uses the metrics of Information Retrieval by calculating the *Recall*, *Precision*, and *F-Measure*, which all vary in $[0, 1]$. And *F-Measure* combines recall and precision with an equal weight in Equation 2 [23].

$$F - Measure = \frac{2 \times Recall \times Precision}{Recall + Precision} \qquad (2)$$

Our experiment is carried out on an IBM ThinkPad notebook with a 2.16 GHz processor, and Microsoft Windows XP with service package 2.

First, an experimental component for extracting concept information from WordNet is developed, which mainly

	AirReservation Ontology	AirReservation Ontology New	CarRetenl Ontology	CarRetenl Ontology New
Concet	14	14	9	9
Slots	21	21	21	21
instance	0	0	0	0
relations	9	10	4	5
Synonym	-	6	-	4

Figure 8: New ontologies after executing WCEA

	Numer of Merge	Numer of Copy	Numer of Conflictions	Precise	Recall	F-measure
Lexical	3	13	2	0.81	0.85	0.829
Lexical with Synonyms	3	-	1	0.85	0.85	0.85
WOMA	4	7	0	0.95	0.95	0.95
Manually	4	5	0	1.0	1.0	1.0

Figure 9: Results of ontology merging

re-uses WSD algorithm in the context of SWS as described in Section 6.2. For instance, after referring to WordNet, a new ontology *Air_reservations'* is produced, which is $\{Aircraft\_\#1, Flight\_\#9, Itinerary\_\#3(travelplan), Corporation\_\#1(corp), Individual, Tour\_operator, Customer\_\#1(client), check\_\#1(bankcheck, cheque), credit\ card, Award\_travel, Reservation, Payment\_record, Reservation\_record, Record\_\#8(checkbook, chequebook)\}$ with their synonyms, but no new relations are obtained. The summary of new ontology metrics is shown in Fig. 8.

Another component is an algorithm plug-in implementing the *ComparisonAlgorithmPlugin* interface of PROMPT, which makes it possible to compare our ontology merging algorithm with PROMPT's default algorithms and with manual results. For example, by carrying out the *Lexical Matching* function of PROMPT, 3 merging operations and 13 copy operations are suggested. By following the suggested operations, the merged ontology has 18 concepts, 32 slots, 19 sub-concept relations, and 0 instances.

Similarly, another PROMPT algorithm (*Lexical with Synonym* was performed. The results (including WOMA algorithm and manual) are recorded in Fig. 9. The numbers of merges, copies and conflicts are listed. By taking the manually built ontology as the Gold standard, precision, recall and F-Measure can be computed. Obviously, during the merging process, manually setting some synonyms can improve the merging result. That is why the *Lexical with Synonyms* algorithm is better than the *lexical* algorithm. Our WOMA algorithm apparently has a higher precision and recall because of having more concept information. This is graphically illustrated in Figs. 10.

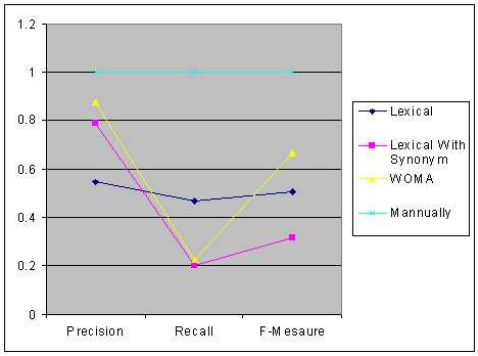

Figure 10: Comparison in precision and recall

8. Conclusion

In this paper, a novel algorithm for building application ontologies is proposed which takes WordNet as knowledge reference; a new semantic net architecture of ontologies is presented which maintains multiple concept relations and concept content; and two kinds of algorithms to merge semantic service ontologies are proposed. A realistic tool ($WSAO^{©}DERI$) to merge service ontologies is developed using our algorithm. It is evaluated by comparison with similar ontology merging algorithms. The experimental results show that the tool is reasonable and effective.

References

[1] E. Agirre and G. Rigau. A proposal for word sense disambiguation using conceptual distance. In *Conf. on Recent Advances in Natural Language Processing*, 1995.

[2] E. Agirre and G. Rigau. Word sense disambiguation using conceptual density. In *Proc. of the 16th Int'l Conf. on Computional Linguistics*, pp.16-22, Denmark,1996.

[3] S. Banerjee. An adapted lesk algorithm for word sense disambiguation using wordnet. In *Proc. of the Third Int'l Conf. on Intelligent Text Processing and Computational Linguistics*, pp.17-22, Mexico City, Mexico, February 2002.

[4] H. L. D. Roman, U. Keller. Web service modeling ontology. In *Applied Ontology*, 1(1):77-106, 2005.

[5] J. R. D.L. McGuinness, R. Fikes and S. Wilder. The chimaera ontology environment. In *AAAI-2000*, pp.1123-1124, Austin, TX 2000.

[6] A. G. G. Steve and D. Pisanelli. Integrating medical terminologies with onions methodology. In *Info. Modelling and Knowledge Bases VIII*, Amsterdam, IOS Press 1998.

[7] R. T. G.A. Miller, C. Leacock and T. Bunker. A semantic concordance. In *Proc. of Flairs 2000*, pp.219-223, Orlando, FL,May 2000.

[8] N. Guarino. Formal ontology and information systems. In *Conf. on Formal Ontologies in Information Systems*, pp.3–15,1998.

[9] D. J. Pathak, N. Koul and V. Honavar. A framework for semantic web services discovery. In *Proc. of the 7th ACM Int'l Workshop on Web Information and Data Management (WIDM 2005)*, ACM pp.45-50, 2005.

[10] G. V. K. Kotis and K. Stergiou. Towards automatic merging of domain ontologies: The hcone-merge approach. *Elsevier's Journal of Web Semantics (JWS)*, 4(1):60–79, 2006.

[11] F. Lecue and A. Leger. Semantic web service composition through a matchmaking of domain. In *European Conf. on Web Services (ECOWS'06)*, pp.171-180, 2006.

[12] H. K. M. Cho and P. Kim. A new method for ontology merging based on concept using wordnet. In *Advanced Communication Technology ICACT 2006*, 3:20-22, 2006.

[13] M. M. Ehrig, P. Haase and N. Stojanovic. Similarity for ontologies - a comprehensive framework. In *ECIS*, 2005.

[14] C. G. M. Sabou, C. Wroe and H. Stuckenschmidt. Learning domain ontologies for semantic web service descriptions. *Journal of Web Semantics*, 3(4):340–365, 2005.

[15] A. Maedche and S. Staab. Ontology learning for the semantic web. In *IEEE Intelligent Systems*, 16(2), March, 2001.

[16] Maedche02. Ontology learning for the semantic web. In *Kluwer Academic Publishers*, 2002.

[17] R. Navigli and P. Velardi. Learning domain ontologies from document warehouses and dedicated web sites. In *Computational Linguistics*, 30(2):151-179, 2004.

[18] N. Noy and M. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. In *Proc. of the National Conf. on Artificial Intelligence (AAAI)*, 2000.

[19] S. Ram and J. Park. Semantic conflict resolution ontology (scrol): An ontology for detecting and resolving data and schema level conflicts. *IEEE Transactions on Knowledge and Data Engineering*, 16(2):189–202, February 2004.

[20] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proc of the 14th Int'l Joint Conference on Artificial Intelligence*, 1995.

[21] T. S. S.A. McIlraith and H. Zeng. Semantic web services. In *IEEE Intelligent Systems*, 16(2):46-53, 2001.

[22] G. Stumme and A. Maedche. Fca-merge: Bottom-up merging of ontologies. In *17th Intl. Conf. on Artificial Intelligence*, pp:225-230, 2001.

[23] C. van Rijsbergen. Information retireval. In *Butterworths*, London, 1979.

[24] C. Y. W. Wu, A. Doan and W. Meng. Bootstrapping domain ontology for semantic web services from source web sites. In *TES*, pp.11-22, 2005.

[25] W3C. Web ontology language for web services. In *http://www.daml.org/services*, 2004.

[26] S. S. X. Li and S. Matwin. A wordnet-based algorithm for word sense disambiguation. *IJCAI*, pages 1368–1374, 1995.