MOD - A Multi-Ontology Discovery System

Le Duy Ngan, Tran Minh Hang, and Angela Eck Soong Goh

School of Computer Engineering, Nanyang Technological University (NTU), Singapore
{ledu0001, A0261836A, ASESGOH}@ntu.edu.sg
http://www.ntu.edu.sg

Abstract. The rapid development of semantic web services has led to a demand for a semantic web service discovery mechanism. Though such discovery systems have been developed, there is a lack of support for matching semantic web services which use different ontologies. This paper introduces a general framework for the discovery of web services that use both the same ontology as well as different ontologies and describe experiments to test the algorithm. Contributions include a discovery algorithm and a novel concept similarity matching algorithm to eliminate mismatches and to increase the accuracy by using syntactic, properties, domain, and neighborhood similarity matching. The experimental results confirm the viability of the discovery system.

1 Introduction

Web services discovery is the most important task in the web services model. Discovery is the process of finding web services which satisfy certain requirements. Researchers have developed discovery systems [5-8,13,16-19] to match web service requesters against web service providers. Current discovery systems are adequate when the web service requester and provider use the same ontology. Unfortunately, research has not focused much on the situation where a web service requester and provider use different ontologies. Since the web service requesters and providers operate independently, they usually define their own ontologies to describe their services. In the real world, a web service provider can provide an exact service to the requester even though both services use different ontologies. Therefore, a discovery system that supports web services using different ontologies is extremely important.

Previous work on web services discovery includes LARKS (Language for Advertisement and Request for Knowledge Sharing) [17-19], a project based on a collaboration between Toshiba and Carnegie Mellon University [6,7], a Matchmaker from TU-Berlin [5], and systems by Li and Horrocks [8], Paolucci [13] etc. These web service discovery systems only support matching web services using the same ontology. This assumption implies that if different ontologies are used, matching cannot be carried out. As mentioned, this is a major limitation.

Cardoso and Sheth [3] have addressed this problem. However, their method of matching web services based on concepts, syntax, and their properties is inadequate, and has been improved upon by Oundhakar et al [12]. The discovery technique is based on METEOR-S [9] which is a web service discovery infrastructure. This

infrastructure provides a facility to access registries that are based on business domains and grouped into federations. The matching is divided into syntactic and semantic matching. In semantic matching, Oundhakar's algorithm has improved on Cardoso's work by considering the coverage and the context information of concepts, thereby improving on the matches and eliminating false matches. However, the system does not allow users to intervene during the matching process to restrict the matching results. This will be a significant problem since the numbers of web services from matching result is huge. Furthermore, the system does not consider the domain of two ontologies, from which the two concepts belong. The domain of the two ontologies is important in the matching process.

This paper introduces a general algorithm which supports matching web services using not only the same ontology but also different ontologies. Computing semantic similarity between concepts from different ontologies is the core component of the algorithm. The component will be introduced with a novel algorithm to eliminate mismatches and improve matching by using syntactic, properties, domain, and neighborhood concept similarity matching. The system is named Multi-Ontology Discovery (termed MOD) system. In the discussion to follow, we assume that the building and maintaining of the ontologies are outside the scope of the paper.

Our work is based on OWL-S [4] which is a semantic web service description language. OWL-S uses OWL [20] ontology to describe the service. A semantic web service defined in OWL-S includes four basic classes, namely *Service*, *ServiceProfile*, *ServiceModel*, and *ServiceGrounding*. For matching, only *ServiceProfile* is used because it contains the description about the service to be discovered. The *ServiceProfile* describes three basic types of information, namely, contact information, functional description of the service, and additional properties. The contact information contains a textual description providing a brief description of the service. It describes what the service offers, or what service is requested. The functional description of the service is the most important declaration used for matching. It specifies the parameters of the inputs, outputs, preconditions, and effects of the service. The Profile also declares operations of the web service.

The rest of the paper is organized as follows. Section 2 introduces the MOD matching algorithm. Section 3 presents main components which are used in MOD. Section 4 describes experiments and results, followed by the conclusions in section 5.

2 MOD Algorithm

The matching algorithm is divided into four stages namely input, output, operation and user-defined matching. The four-stage set up is similar to TUB [5] but MOD supports matching web services using different ontologies.

2.1 Definition of Semantic Similarity

In order to introduce concept similarity, we define similarity distances. The definition is based on the experience in [12]. Assume that A, B, C, and D are concepts of an

ontology. Figure 1 describes the possible relationships between concepts. The similarity distances between two concepts are defined as follows:

- Exact similarity (A, A): is the most accurate match. It happens when the two descriptions are semantically equivalent. The similarity degree for this match is given a value 1 and this is the highest similarity.
- Subsumes similarity (A, B): B is subsumed by A. As A is a direct superclass of B, it is more general than B. This match is less accurate than exact match. The similarity degree for this match is 0.75.
- General similarity (A, D): A is more general than D but A is not a direct superclass of D. We assume that there are x levels (with x≥1) between A and D, then the similarity degree for this match is $0.75 - 0.01*2^{x-1}$.
- Invert-Subsumes similarity (B, A): B is more specific than A and B is a direct subclass of A. The similarity degree for this match is 0.3.
- Specific similarity (D, A): this is the inverse of the general match. A is more general than D but A is not a direct super class of D. We assume that there are x levels (with x≥1) between A and D, then the similarity degree for this match is 0.3 – 0.05*x.
- Fail similarity (A, C): A and C are not related in the ontology. With reference to figure 1, we also have Fail (B, C).

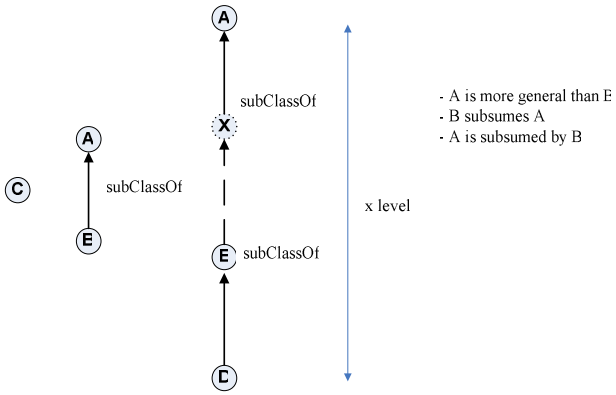

Figure 1. The relation of concepts

Similar to [12], we use levels of two concepts to compute general and specific similarity. We reduce the similarity by 0.01 for parents, and 0.02 for grandparent, and so on for super-concepts. With sub-concepts, we reduce by multiples of 0.05 for each level. The maximum number of levels to be considered is 6.

2.2 MOD Matching Algorithm

The MOD algorithm divides the matching process into four-stage and each stage is independent of the other and the final result is based on the sum of the individual stages.

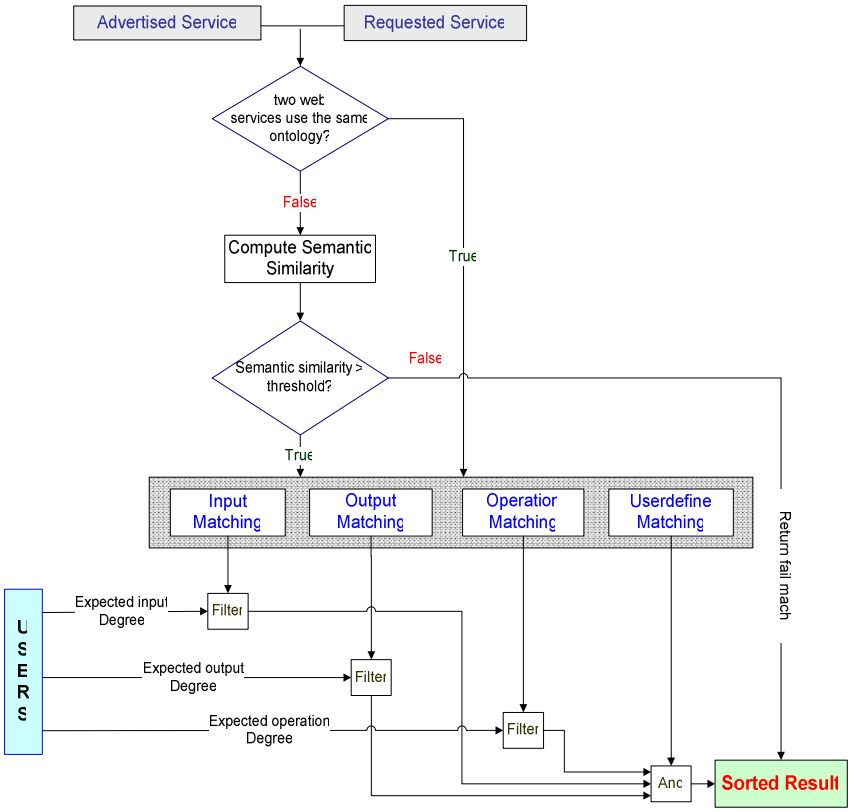

Figure 2. Matching algorithm

The matching algorithm is described in figure 2. *reqService* and *advService* denote the requested and advertised service respectively. First, the two web services are checked if they use the same ontologies. If they do, the four-stage matching algorithm [5] is used. If they do not, the "compute semantic similarity" component computes the concept similarity between concepts. If the semantic similarity is less then a threshold which was defined by the user, then the matching fails. Otherwise, the four-stage matching algorithm is used with input, output, or operation matching. The algorithm is described in more detail in section 3. The four-stage algorithm first checks the user-defined matching. If the matching fails, the result of matching two services also fails and the matching process finishes. Otherwise, the matching process continues with input, output, and operation matching. The final matching result will be composed of the input, output, and operation matching.

2.2.2 Input Matching

In the input matching, we determine how inputs of the advertised services are satisfied by the inputs of the requested services. The input matching operates as follows: for each input of the advertised service, the algorithm tries to find an input of the requested service that is the most similar (the highest degree match). If none of the advertised input can match with the requested input, the input matching returns fail. Thus, the matching of the two web services will fail. The input matching algorithm is elaborated as follows:

```
inputMatch (inputAdv, inputReq){
 if (inputAdv = empty_list){
 return 1; //exact match.
 }
 degreeInput = 0;
 numInput = 0; //number of input of advertised service
 for all inputAdvElement in inputAdv do{
  numInput = numInput +1;
  degreeMax =0;
  for all inputReqElement in inputReq do{
   if(checkOntRelation(inputAdvElement,inputReqElement)){
    degreeMatch =
    computDegree(inputAdvElement,inputReqElement);
   }
   else {
    degreeMatch =
    computSimilarity(inputAdvElement,inputReqElement)
   }
   if (degreeMatch >degreeMax){
    degreeMax= degreeMatch;
   }
  if degreeMax = 0 then return fail;
   degreeInput = degreeInput + degreeMax;
  }
 degreeInput = degreeInput/numInput;
 if degreeInput < threshold then return fail;
 else return degreeInput;
}
```

The number of input of advertised and requested service may be none or many. If the advertised service does not contain any input, then the input matching returns an exact match. *inputAdvElement* and *inputReqElement* denote elements of the input list of advertised and requested service respectively. *degreeMatch* is the degree of the matching between an input pair (an input of advertised service and an input of the requested service). *degreeMax* is the maximum degree of an input of advertised service with every input of the requested service. In other words, *degreeMax* is the maximum of *degreeMatch* for every input of advertised service. *degreeInput* is the final result of input matching. *degreeInput* will be compared against a threshold whose value depends on the specific domain. If it is less than the threshold, then the matching fails. Otherwise, the matching returns the *degreeInput*. *checkOntRelation*() is the function that checks if the two concepts have a hierarchical relationship.

computDegree() is the function that computes the semantic degree of two concepts when they are related hierarchically. *computSimilarity*() is the function that compute the semantic similarity between two concepts from different ontologies. The algorithms for the *computSimilarity*() functions will be introduced in section 3.

2.2.3 Output, Operation, User-defined, and Final Result Matching

Both output and operation processing are carried out in the same manner as input. But in output matching, all output from the requested service will be matched with each output in the advertised service. In operation matching, service category operation of the requested and service category operation of the advertised service are matched. The results of output matching and operation matching are the degree of output matching and operation matching.

In user-defined matching, the users can declare some more constraints or rules to restrict the matching and increase the accuracy of the results of matching. For example, a requester wishing to buy a computer describes a web service with price as input and configuration of the computer as output. They may also declare more constraints relating to the computer manufacturer. These constraints or rules will be matched against the advertisement. Users may decide not to define any constraints or they may define many constraints. We assume that the result of a constraint matching is either "match" or "not match". In short, the result of user-defined matching is as follows: Match if every constraint is satisfied; Fail if at least one constraint fails. Of course, it is possible to define the distance of this matching but it is outside the scope of work.

The final matching result will be composed of the input, output, and operation matching.

$$S = \frac{(w_1 * inputMatch + w_2 * outputMatch + w_3 * operationMatch) * userMatch}{w_1 + w_2 + w_3}$$

w_1, w_2, and w_3 are defined by users. After matching with all advertisements from the database, we will sort by the degree of similarity and return the matched list to the requester.

3 Semantic Similarity between Concepts from Different Ontologies

This section introduces the main component of MOD which was mentioned in section 2, namely, computing semantic similarity of two concepts from different ontologies. Determining the semantic similarity of concepts from ontologies is necessary in information retrieval and information integration fields related to ontologies. MOD focuses on matching inputs, outputs, and operations because in the semantic web, these parameters are in fact ontology concepts. The concept similarity algorithm

includes four main components: syntactic similarity, properties similarity, domain similarity (or context similarity), and neighborhood similarity.

$$CS = \frac{w_s * synSim + w_p * proSim + w_c * conSim + w_n * neiSim}{w_s + w_p + w_c + w_n} \quad (1)$$

where w_s, w_p, w_c and w_n are weights defined by users depending on application. The following sections provide details of the five main matching components.

3.1 Syntactic Similarity

Each concept in an ontology is labeled (concept name) and has a short text (concept description) which describes it. The syntactic similarity computes the similarity between the concept names and concept descriptions of the two concepts, respectively. The concept names in OWL are defined as a word or a set of words. There are some techniques to compute word similarity such as n-gram [2,15,21] and token Matcher [14] but most of these only consider a word as a string of characters. The semantic relationship of the words, which is very important in computing word similarity, has been ignored. To overcome this drawback, WordNet [10] is used.

WordNet can only be used when the words which we would like to compute similarity are stored in its database. The WordNet database only stores root and single words. Therefore, before using WordNet, we must have a preprocessing process to convert words to their roots. If both concept names contain only one word, we can use directly WordNet to compute similarity words. In cases when there is more than one word in the concept name, we will find the most similar terms between synonym sets. WordNet is also used to compute similarity between two concept descriptions in the same way as computing concept names similarity when the latter contains more than one word. The syntactic similarity is a weighted average of concept names and concept descriptions similarity.

3.2 Property Similarity

A concept may have one or more properties. Similar to concept, a property also has a name and description. In addition, it contains range and cardinality. To compute the property similarity, all this information of the two properties should be matched. The method to compute property name and description similarity is syntactic similarity which was introduced in section 3.1.

Range of a property is either a primitive data type or another concept. If both ranges are primitive data types then the similarity between two ranges is as shown in table 1. If one range property has a primitive type and the other has a concept, the ranges are incompatible; therefore the range similarity is 0. If two range properties are concepts, the matching is carried out recursively as with the computing of two concepts.

Table 1. Similarity when range is primitive data type

		Range of Property 1				
Range of Property 2		Integer	Long	Float	Decimal	String
	Integer	1	0.9	0.8	0.7	0.3
	Long	0.9	1	0.8	0.7	0.3
	Float	0.8	0.8	1	0.7	0.3
	Decimal	0.7	0.7	0.7	1	0.3
	String	0.3	0.3	0.3	0.3	1

Cardinality of property permits the specification of the exact number of elements in a relation. For OWL, the cardinality values are limited to 0 or 1. We simply define the Cardinality similarity as follows: Cardinality similarity =1 if the cardinality of two properties is equal; Cardinality similarity =0 if the cardinality of two properties are different

The final property similarity is the combination of the three components: syntactic, range, and cardinality similarity.

3.3 Domain Similarity

Domain which is also known as a context of the ontology is important to the computation of concept similarity. For example, assume there is an Animal ontology which has concept Fish. Another Food ontology also has concept Fish. We assume that the properties of the two concepts Fish are the same. If we do not consider the domain of the two ontologies, the concept similarity of two concepts from the two ontologies will be 1. However, the Fish concept has different meaning in the two ontologies: one refers to an animal; the other refers to food. Therefore, the similarity distance of the Fish concept from the two ontologies should be low. In short, domain must be considered in computing concept similarity.

To compute the domain similarity of the two concepts, we compute the similarity of the two roots from the two ontologies since the root represents the domain of the ontologies. The root is a special concept in an ontology, which does not have super concepts. In the real world, most OWL ontologies have one root, but there are ontologies with more then one root. In this situation, we must find the root which the concept belong to. The paper only solves the simple and common situation when the ontology has one root.

3.4 Neighborhood Similarity

Neighborhood similarity computes the two concepts with respect to their neighborhood, namely, their super concepts and sub concepts. Figure 3 presents a matching of two concepts Network Node and Network Element. Both concepts have

super concepts Equipment; their sub concepts are Computer, Switch Equipment, and Computer, Central Hub, respectively.

The neighborhood similarity is computed using the following equation:

$$\text{neighborSim} = \sqrt{\text{supSim} * \text{subSim}}$$

where supSim and subSim are super concept similarity and sub concept similarity, respectively.

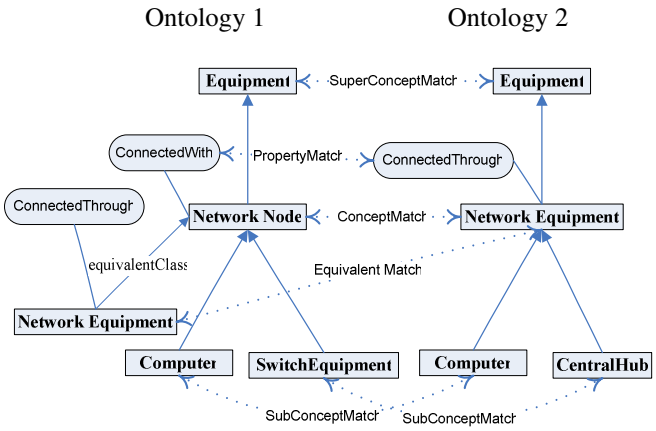

Figure 3. Comparing two NetworkNode concepts

A concept may have one or several super concepts. The super concept similarity is computed as follow:

$$\text{supSim}(C_P, C_R) = \frac{\text{allSubSim}(C_P, C_R)}{n}$$

where n is the number of pairs of matched super concepts. C_P and C_R are concepts from the two ontologies used by provider and requester, respectively. allSupSim is similarity of the matched super concepts. As shown in equation (2), the super concepts are matched one to one such that the average super concept match is maximized. This is done by using a recursive formula as follows:

$$\text{allSupSim}(CP, CR) = \text{Max}(\text{allSupSim}(CP - SupP, CR - SupR)) + \text{conceptSimilarity}(SupP, SupR)) \quad (2)$$

A concept may have one or more sub concepts. subSim is computed in the same way as supSim.

4 Experiment and Discussion

In this section, an experiment to determine concept similarity using two real world ontologies is introduced. This is followed by a description of how two web services which use the two ontologies are matched.

4.1 Experiments with Concept Similarity

For testing the concept similarity algorithm, we employ two ontologies from the real world taken from I3CON [11], the Information Interpretation and Integration Conference. Figure 4 shows two ontologies from two different OWL files: networkA.owl and networkB.owl. These two ontologies describe the nodes and connections in a local area network. networkA focuses more on the nodes themselves, while networkB encompasses the connections. Concept pairs were selected for testing arbitrarily but focused on pairs which have matching potential. The concept similarities are computed using equation (1). In most cases, w_i is set at 0.25. In some special cases, w_i is set differently. Table 2 shows the results of testing twelve concept pairs from the above two ontologies. The "expected results" column shows the "Ground Truth" result which is defined manually by human intelligence to determine closest fit. The "MOD Results" column shows the actual results which were obtained by using the proposed algorithm.

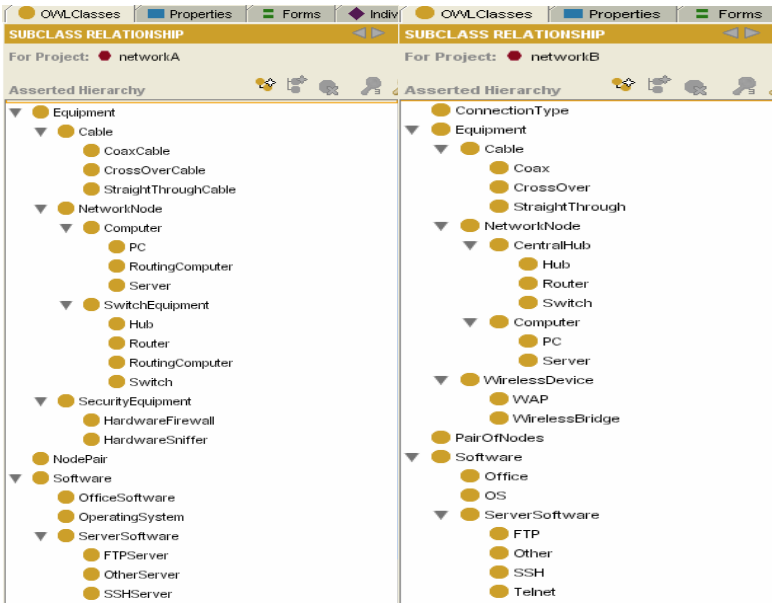

Figure 4. Structures of networkA.owl and networkB.owl in Protégé [1]

The details of matching are shown in figure 5. Each column pair in the figure shows the "expected" and MOD concept similarity. MOD results are computed based on the

four similarity components: syntactic, properties, neighborhood, and domain similarity.

Table 2. Similarity between concepts

Case	Concepts from NetworkA.owl	Concepts from NetworkB.owl	Expected Results	MOD Results
1	NetworkNode	NetworkNode	0.80	0.78
2	Equipment	Equipment	0.75	0.65
3	PC	Router	0.60	0.58
4	NodePair	PairOfNode	0.57	0.63
5	Hub	NetWorkNode	0.25	0.25
6	Computer	Computer	0.80	0.82
7	CrossOverCable	Cable	0.70	0.55
8	Router	WAP	0.35	0.59
9	Server	Router	0.45	0.67
10	CoaxCable	Coax	0.72	0.7
11	CrossOverCable	CrossOver	0.72	0.75
12	StraightThroughCable	StraightThrough	0.72	0.78

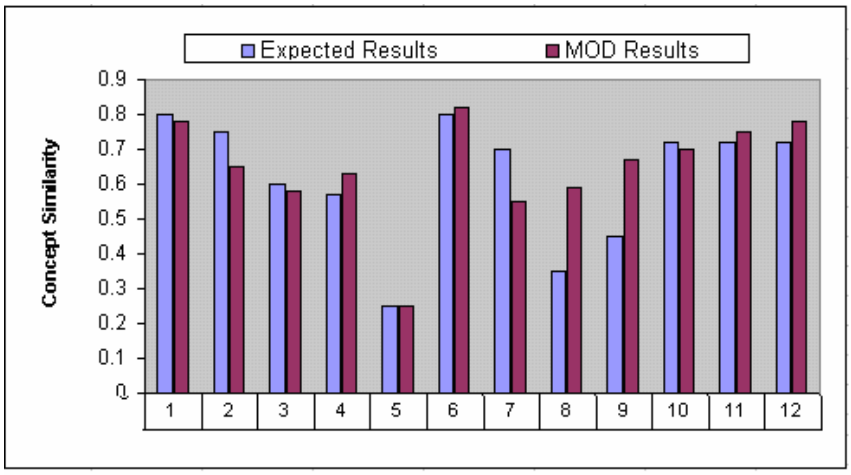

Figure 5. Graph represents concept similarities from Network A and Network B ontology

The syntactic matching results show that cases 1, 2, and 6 are 1 since their concept names are identical and the concept descriptions are nearly the same. The lowest syntactic similarity is in case 5 which involves two concepts: Hub and NetWorkNode. With the two concept names "Hub" and "NetWorkNode", WordNet returns a similarity is 0. But the similarity of the two concept descriptions is 0.2, so the syntactic similarity is 0.1. When no property is defined in one of the two concepts, their property similarity is 0. In this case, the algorithm will ignore the property dimension. In other words, the w_p in equation (1) is 0.

In most cases, the neighborhood similarity is high when their superclass and subclass similarity are high. This similarity is low in case 7 since CrossOverCable

concept in ontology networkA matched Cable concept in ontology networkB. However, in ontology NetworkA, the Cable concept is the superclass of CrossOverCable concept. Therefore, the neighborhood similarity of the two concepts is the similarity of the superclass pair (Cable, Equipment). The subclass similarity is ignored because CrossOverCable does not have a subclass. The pair (Cable, Equipment) has super/sub class relationship, but in neighborhood similarity matching, the algorithm only considers the syntactic, property, and domain similarity. Therefore, the neighborhood similarity between CrossOverCable and Cable is low. Domain similarity is the similarity of two root concepts of the two ontologies. Therefore, the domain similarities in the twelve cases are the same.

The similarity of two concepts is computed as the average of the four components: syntactic, property, neighborhood, and domain similarity. Therefore, the value of each component will affect the final result. The contribution of the four components is equal in most cases. However, there are some cases where one component carries more value than the other. The neighborhood similarity for the pair (CrossOverCable, Cable) is low and has 'negative' effect on the final result because the matching algorithm does not recognize that they have a superclass/subclass relationship. It is therefore up to the user to select a suitable weight w_i to place on each of the components.

4.2 Matching Web Services

We assume that there are two companies: Cisco is a network company which sells network components. BCS is a network company which buys network components. Cisco and BCS develop web services to sell and buy network components, respectively. The inputs of the web services are price; the outputs of the web services are network components. To develop web services, companies must develop price, e-business, and equipment ontologies. We assume that the price and e-business ontology are popular, so the two companies use the same price and e-business ontology. But the equipment ontology is not available. So, they develop different ontologies for the equipment as shown in figure 4. The web services of the two companies are shown in figure 6. In this experiment, we assume that the threshold of concept similarity is very small, so all the concept similarities are larger then the threshold.

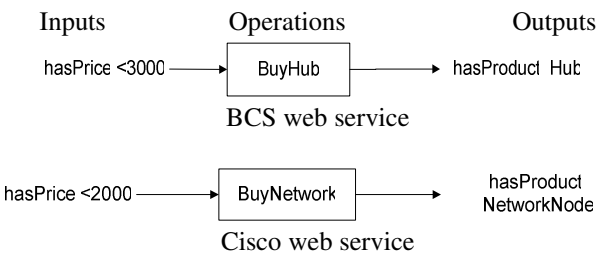

Figure 6. Web service requester and advertisement

Inputs of the web services are concepts from price ontology which is described in figure 7; We introduce concepts <3000, <2000, and <1000 that refer to prices that are less than 3000 units, 2000 units, and 1000 units, respectively.

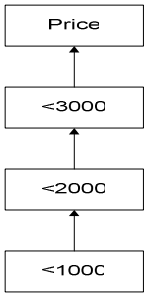

Figure 7. Ontology for price domain

Operations of web services are concepts from eBusiness ontology which is described in figure 8. The eBusiness provides Sell and Buy services. The Buy concept provides BuyComputer, BuyEquipment, and BuySoftware. BuyEquiment provides BuyHub, BuyRounter and BuySwitch.

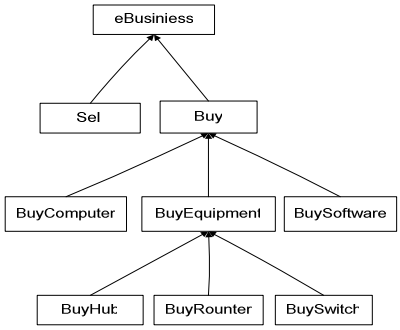

Figure 8. Ontology in eBusiness domain

With descriptions of the two web services and above ontologies, matching results are as follows:

- Input matching: This case is invert-subsumes since the input of the BSC service is a superclass of the input of Cisco service. Based on the definition in section 2, the input matching value is 0.3.
- Operation matching: This case is subsumes since the operation of the BSC service is the sub concept of the input of Cisco service. The operation matching value is 0.75.

- Output matching: In this case, the two web services use two concepts from different ontologies. Based on the results stated on section 4.1, the matching of two concepts Hub and NetworkNode is 0.25.
- User-defined: The user can define more rules or constraints to restrict the results. In this example, for simplicity, we assume that the user-defined matching is satisfied.

Based on the MOD algorithm, the similarity of the two web services is computed as follows:

$$S = \frac{(inpMatch + OutpMatch + OperMatch) * UserDefMatch}{3} = \frac{0.3 + 0.75 + 0.25}{3} = 0.43$$

The matching result of the two web services indicates that this is "subsume matching", that is, provider Cisco can satisfy the requester BCS but Cisco service is more general than BCS service.

5 Conclusions and Future Work

This paper has introduced an algorithm termed MOD which supports matching of web services using different ontologies. The algorithm is divided into four stages, namely, input, output, operation, and user-defined matching. The computing concept similarity which plays a key role in MOD has four main components, namely, syntactic, properties, domain, and neighborhood similarity. The experimental results confirm the viability of the discovery system. In the experiment, we used OWL-S ontologies to test the algorithm, but the MOD algorithm is general; therefore it can be used for web services using different semantic web service description languages such as DAML-S, and RDFS. However, the similarities defined in section 2 should not be the crisp sets; it should be defined as fuzzy sets. The domain similarity only considers the simple case when the ontology has one root. In the real word, the ontology may have more then one root. These problems will be considered in the future.

References

[1] Protégé - A tool for OWL editor.http://protege.stanford.edu/overview/

[2] Angell, R.; Freund, G., Automatic spelling correction using a trigram similarity measure, *Information Processing and Management*, (1983) **19**, 4, 255,

[3] Cardoso, J.; Sheth, A., Semantic e-Workflow Composition, *Journal of Intelligent Information Systems*, (2003) **21**, 3, 191 0925-9902, Kluwer Academic Publishers.

[4] DAML, Services Home Page, OWL-S markup of services.www.daml.org/services/owl-s/

[5] Jaeger, M. C.; Tang, S.; Liebetruth, C., The TUB OWL-S Matcher.Available at: http://ivs.tu-berlin.de/Projekte/owlsmatcher/index.html.

[6] Kawamura, T.; Blasio, J. D.; Hasegawa, T., et al., (2003), Preliminary Report of Public Experiment of Semantic Service Matchmaker with UDDI Business Register, *1st International Conference on Service Oriented Computing (ICSOC 2003)*, Trento, Italy, 208,

[7] Kawamura, T.; Blasio, J. D.; Hasegawa, T., et al., (2004), Public Deployment of Semantic Service Matchmaker with UDDI Business Registry, *3rd International Semantic Web Conference (ISWC 2004)*, **LNCS 3298**, 752,

[8] Li, L.; Horrocks, I., (2003), A software framework for matchmaking based on semantic web technology, *12th International World Wide Web Conference*, Budapest, Hungary, ACM Press, 331, 1-58113-680-3,

[9] LSDIS, METEOR-S: Semantic Web Services and Processes.http://lsdis.cs.uga.edu/projects/meteor-s/

[10] Miller, G., WordNet: An Electronic Lexical Database, The MIT Press: (May 15, 1998).

[11] NIST, I3CON Information Interpretation and Integration Conference.http://www.isd.mel.nist.gov/PerMIS_2004/index.htm

[12] Oundhankar, S.; K. Verma; Sivashanugam, K., et al., Discovery of web serivces in a Muti-Ontologies and Federated Registry Environment, *International Journal of Web Services Research*, (2005) **1**, 3,

[13] Paolucci, M.; Kawamura, T.; Payne, T. R., et al., (2002), Semantic Matching of Web services Capabilities, *1st International Semantic Web Conference (ISWC 2002)*, Sardinia, Italy, 333,

[14] Porter, M. F., An algorithm for Suffix Stripping, In *Morgan Kaufmann Multimedia Information And Systems Series*, Morgan Kaufmann Publishers: (1997); pp 313

[15] Salton, G., Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer, Addison-Wesley Ed. Massachusetts: (1988).

[16] Srinivasan, N.; Paolucci, M.; Sycara, K., (2004), Adding OWL-S to UDDI, implementation and throughput, *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, California, USA,

[17] Sycara, K.; J. Lu, M. K.; Widoff, S., Dynamic service matchmaking among agents in open information environments, *ACM SIGMOD Record (Special Issue on Semantic Interoperability in Global Information Systems)*,, (1999) **28**, No.1, 47,

[18] Sycara, K.; J. Lu, M. K.; Widoff, S., (March, 1999), Matchmaking among heterogeneous agents on the internet, *AAAI Spring Symposium on Intelligent Agents in Cyberspace*,

[19] Sycara, K.; Widoff, S.; M. Klusch, J. L., (2002), LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace, *Autonomous Agents and Multi- Agent Systems*, Vol.5, 173,

[20] W3C, Organization, OWL - Web Ontology Language Overview.http://www.w3.org/TR/owl-features/

[21] Zamora, E.; Pollock, J.; al, e., The Use of Trigram Analysis for Spelling Error Detection, *Information Processing and Management*, (1981) **6**, 17, 305,