Journal of Theoretical and Applied Information Technology
20th October 2013. Vol. 56 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AN EFFICIENT SEMANTIC WEB SERVICES SELECTION MODEL USING CLUSTERING

[1]SHYNU P.G.
[1]Asstt Prof., School of Information Technology & Engineering, VIT University, Vellore, India
E-mail: [1]pgshynu@gmail.com

ABSTRACT

Web Services are one of the fastest growing areas of information technology in recent years, also being a main motivating factor for internet computations in which, one of the services being, service discovery. Web service discovery is the process of finding appropriate services for the user defined tasks.Web Service clustering is a technique for efficiently facilitating service discovery. Most Web Service clustering approaches are based on suitable semantic similarity distance measure and a threshold. Threshold selection is essentially difficult and often leads to unsatisfactory accuracy. In this paper, a self-organizing based clustering algorithm called Taxonomy based clustering for taxonomically organizing semantic Web Service advertisements. A query matching method is also applied on these clusters to get more accurate and relevant results based for user requests. The system is tested and observed promising results both in terms of accuracy and performance.

Keywords: *Web Services Discovery, Composition, Ontology, Semantic Similarity, Clustering*

1. INTRODUCTION

Service discovery is one of the key problems that have been widely researched in the area of Service Oriented Architecture (SOA) based systems over the past decade. This problem is significant because all the other big problems in this area of research, such as service selection and composition, are intricately related to an efficient discovery mechanism. If the SOA middleware can discover services (over a set of service registries such as Universal Description Discovery & Integration (UDDI) accurately and fast then the problem of service selection and composition become relatively easy. However, the discovery problem is intrinsically difficult because most large scale SOA based systems (e.g., Web Services, Cloud Computing, etc) are dynamic and uncertain in nature. They are dynamic because the scope of services is ever changing – new services get added on to the system and old services get removed or modified. Also it is very difficult to model statistically this dynamism. This makes process of service discovery fundamentally hard as the middleware has to cope up somehow with both the dynamism and the uncertainty. The key operation that governs all kinds of service discovery approaches is service matchmaking over service descriptions. Matchmaking is essentially computation of similarity between a query description and a service description so that the middleware can decide whether a service is a candidate solution for the given query. However, matchmaking can be expensive due to two reasons: (i) the measure for similarity is intrinsically complex from a computational view point, and (ii) the query search space is extremely large resulting in a lot of unwanted but expensive similarity computation (i.e. comparisons). Hence, first, we need to choose a similarity measure for matchmaking that is simple and computationally fast, and second, we need an efficient way of pruning the query search space for service discovery. Moreover, both the similarity measure selection as well as the pruning strategy should entail optimal accuracy of query results in terms of precision (i.e., least inclusion of false results) and recall (i.e., least exclusion of true results). Similarity measures in most matchmaking researches is either based on semantic subsumption reasoning (using DL reasoners) over service descriptions written in languages such as OWL-S [2] or is based on statistical IR similarity models (such as vector space models, probabilistic models, information theory based models, etc). While semantic subsumption reasoning in worst case can be intractable and for most reasoners is not very fast similarity computation using IR techniques do not guarantee soundness and completeness. Also in some cases we need considerable training of the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

searching model before a relatively fair accuracy can be reached.

As for search space pruning strategies one very efficient approach is service clustering. In this approach services are clustered into functionally similar groups and then every cluster can be mapped to an index for a directed query processing. When a query comes in the middleware can guarantee the exclusion of any potential candidate service outside its corresponding cluster and hence, limit the search only to that cluster. Web service clustering algorithms can be classified into two general approaches: (i) similarity distance measure based and (ii) self-organization based. Both the approaches have the capacity to exploit semantic information of the service descriptions. Most distance-based approaches in service clustering strike a balance between the accuracy and the computational cost incurred by the algorithm. However, there are several significant limitations that can be observed in this approach. To begin with, for any sort of sample space having an independent, identical probability distribution (I.I.D) it is difficult to construct a classifier that classifies the samples into their correct clusters. One of the reasons is that the accuracy of the algorithm depends upon the order of sample selection during clustering. Secondly, distance-based algorithms need to assume the choice of good distance threshold for clustering. We claim that for the problem of semantic Web Service clustering we do not need a distance based approach.

This paper proposes a self-organizing based clustering algorithm called Taxonomy based clustering for taxonomically organizing semantic Web Service advertisements. The approach overcomes the problem of sample selection order as well as suboptimal choice of threshold. For the Taxonomy based clustering algorithm, it is also proposed a non-distance based subsumption matchmaking technique that does not require DL reasoners for computing subsumption. Instead it uses bit-based encoding technique that significantly reduces the comparison cost as compared to subsumption and other IR based techniques. In this way the algorithm guarantees a much higher accuracy than distance-based approaches. The system has been tested the proposed algorithm on both simulation based randomly generated test data and the standard OWL-S TC test data set and observed promising results both in terms of accuracy and runtime performance.

The rest of the paper is divided as follows: Section 2 discusses related work where standard service discovery infrastructure and service matching techniques have been introduced. Section 3, explains the taxonomy based clustering in detail. In Section 4, web service discovery as a query matching problem where the motivation behind the proposed clustering technique has been laid. Section 5 evaluates the proposed approaches and Section 6 concludes this paper.

2. RELATED WORK

All web service discovery problems have been treated as a special case of information retrieval problem by most researchers. The common principle is to cluster similar services in groups that are stored in backend system. Backend systems can be classified into three types: (i) index table based (such as UDDI, Jini, MSLP [3]), (ii) DHT-based (such as CHORD [1]), and (iii) taxonomical hierarchy based (such as capability graphs [4]). Currently, the most popular backend implementation for service discovery is UDDI. However, UDDI is at a very syntactic level. It is organized based on pre-defined thematic categories. However, the services are not categorized with respect to their functionality. This is a major drawback in terms of design as consumer access is mainly based on the functional attributes of Web services. Many researchers have proposed an extension to the existing UDDI structure by adding semantic descriptions to the services. The semantic description can be at three levels: (i) functional (such as OWL-S [2]), (ii) contextual (CCC/PP), and (iii) QoS (such as WSOL [5]). Service clustering can be performed at these three levels [6] using several approaches.

Web service clustering one of the most widely used method which is distance based. In this approach a semantic distance measure is first defined. This distance measure can be of two types: (i) keyword based [7] and (ii) ontology based [8]. In keyword based distance measures the similarity of two services is computed using a vector space model based on TF/IDF technique derived from IR research. As an alternative approach, there has been significant research on ontology based semantic distance measure [9,10]. Semantic distance

Journal of Theoretical and Applied Information Technology
20th October 2013. Vol. 56 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

measures can be classified into three categories: (i) taxonomic distance based [11,12], (ii) information content (IC) based [13], and (iii) concept property based [14]. In the case of service similarity measure a simple taxonomic distance based or IC based measure cannot give us precise results. This is because a Web service is a complex concept with independent multi-faceted dimensions like functionality, context and QoS. In general most research approaches choose a specific similarity distance measure. Then similarity is separately computed at the attributive level - namely, input (I), output (O), pre-condition (P), and post condition/result (R) [10]. Based on such computation a decision rule is applied that tells whether two services are similar or not with respect to a defined global threshold. The other alternative service clustering approach is self organization. The proposed clustering algorithm falls under this approach. Self-organization of Web services takes place on a conceptual plane (rather than a metric plane). The basis of clustering in this case is subsumptive matching instead of similarity computation. According to the model given in [10], there can be four types of Web service subsumptive match: (i) exact match (where the attribute/s of one service is same as or sub class of the attribute/s of another service), (ii) plug-in (where the attribute/s of the plug-in/replaceable service subsumes the attribute of the other service), (iii) subsumes (where the attribute/s of the subsuming service is a higher level generalization of the attribute/s of the other service), (iv) fail (where the attribute/s of one service has no subsumptive relation with the attribute/s of the other service). It has been pointed out by several researchers that in the context of service discovery a subsume match is considered the weakest match while an exact match is the strongest [4, 9, 15].

3. TAXONAMY BASED CLUSTERING

A self-organizing based clustering algorithm called Taxonomy based clustering for Web Service advertisements in proposed. The basic idea behind the clustering algorithm is to generate clusters over the sample space for each dimension independently (i.e., stratified approach). The stratified approach of clustering web services overcomes the problem of sample selection order as well as suboptimal choice of threshold. As a result, we get four different levels of cluster spaces – each corresponding to the four functional dimensions. We now define the scope of the problem which is to cluster web services according to their functionality for their efficient and accurate discovery.

As the proposed taxonomic web service clustering is essentially a subsumptive match based technique hence, we first need to compare any two given services with respect to a particular dimension (I, O, P, R) for three possibilities: (i) one is the ancestor/predecessor of the other (i.e., subsumption), (ii) both are sibling under a common parent, and (iii) both are mutually disjoint with no possible abstraction. For an example, assuming that two services *s1* and *s2* have their corresponding output dimension (i.e. O cluster space) to be *vehicle* and *SUV* (Fig 1). In such case both the services have a subsumptive relation with respect to the O dimension. Again, assuming that another service *s3* takes in *bus* as its output then *s3* and *s2* are siblings under the common parent *s1*

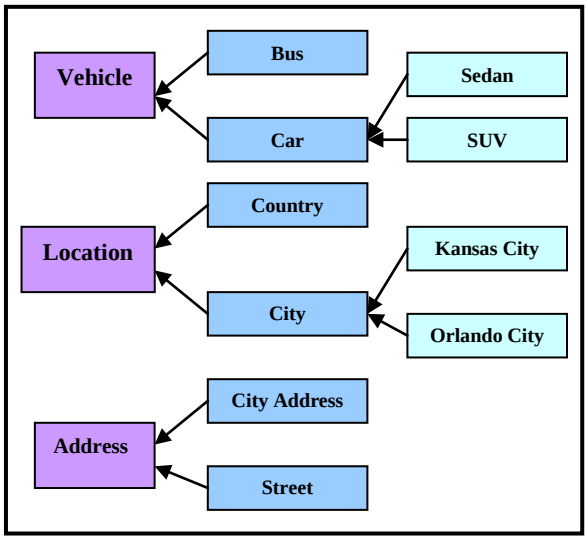

Figure 1. Vehicle, Location & Address Ontologies

3.1. Scope of Problem

Problem Definition: Given a finite set of web services generate a set of clusters such that if a member web service of a particular cluster has a particular class of functionality then all the other members of the same cluster must have same class of functionality subject to the constraints:

- The selection of web services from the sample space follows *I.I.D* (i.e., independent, identical distribution)
- The selection order of samples does not affect the accuracy of the clustering

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

process (no *effect of sample selection order*)
- Fuzzy clusters are not allowed (no probabilistic membership of sample within a cluster)

3.2. Stratification of Sample Space

Now we can formally define a cluster space as follows:

Definition 1: A g-subsumption is *s1.g* **K** *s2.g* = true iff (*s1.g* **T** *s2.g*) = *s1.g* where **T** is intersection over *g* dimension; g = {I, O, P, R}.

Definition 2: A g-type taxonomic cluster (denoted as $_T{}^g$) is a partial-order of web service (samples) over g-subsumption relation $\overset{g}{\boldsymbol{k}}$ with a unique maximum upper bound (or *least specific parent*) called the *root*.

Definition 3: A cluster space (denoted as *CS*) w.r.t to a particular functional dimension *g* is a well-defined finite set of *g-type taxonomic clusters* such that any member taxonomic cluster may have g-subsumption relation k *g* with one or more member taxonomic cluster via corresponding member samples.

The cluster spaces corresponding to the O dimension are called the O cluster space as the Type 1 component of a query is directed towards this cluster space. The cluster spaces corresponding to the I dimension are called the I cluster space as the Type 2 component of a query is directed towards this cluster space.

As the proposed taxonomic web service clustering is essentially a subsumptive match based technique hence, we first need to compare any two given services with respect to a particular dimension (I, O, P or R-spaces) for three possibilities: (i) one is the ancestor/predecessor of the other (i.e., subsumption), (ii) both are sibling under a common parent, and (iii) both are mutually disjoint with no possible abstraction

The order of sample selection over an *I.I.D* may have negative side-effect on the overall clustering performance. This can be shown via an example. Let us consider three services *s1*, *s2*, and *s3*. Let these services need to be clustered according to their output dimension (O).

It is given that *s1.o* = {*car, location*}, *s2.o* = {*vehicle, city, address*}, *s3.o* = {*SUV, street_address*}. The domain set for this example is: *{vehicle, location, address}* (Fig 1). From a semantic taxonomy point of view, *s1* and *s2* are siblings under a common abstraction {*vehicle, location*} while *s3* is sibling to this abstraction under a common abstraction {*vehicle*}.

3.3. Taxonomy Based Clustering Algorithm

The basic outline of the proposed taxonomy based clustering algorithm involves positioning a randomly selected sample (Web Service) from the given ample space and semantically positioning it in the cluster space by searching for the *most specific parents* (*MSP*) and the *least specific children (LSC)*. An important optimization can be made here by restricting the search for *LSC*s only to the children of the *MSP*s already discovered. Semantic positioning is based on subsumption matching between the sample and the already clustered samples in the cluster space. Thus, a sample after positioning either forms a new taxonomy (i.e. cluster) or joins an existing set of taxonomies.

The algorithm, *Taxonomy based Clustering* returns an instantiated CS when given the sample space (S). This main algorithm requires two functions: (a) *findMSP* for computing the MSP of a particular sample, and (b) *findLSC* for computing the LSC of a particular sample. For service matching the algorithm *SubsumptionMatch* uses the binary bit code representation with encoding algorithms presented in [16]. It returns 0 if there is no match, 1 if the first argument service subsumes the second argument service, and 2 if the argument services are sibling under a common abstract parent service. It may happen that the sample service does not find any MSP. In that case two things may happen: (a) the sample becomes a root or (b) the sample can be a sibling of one or more of the existing root services under a common root service.

ALGORITHM: Taxonomy Based Clustering

INPUT: sample space S = {s1, s2, s3 ….. sn}
OUTPUT: cluster space CS1...n

```
TaxonomicClustering(S){
 CS = NULL // initially CS is set as empty
  For count = 1 to n{
    sample = randomSelect(S);
     S = S – {sample};
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
        MSP = findMSP(sample, CS);
For 1 to MSP.size{ // PLSC: Potential LSC
        PLSC=PLSCS
        findLSC(memberOf(MSP)); }
        findLSC(sample, PLSC, CS);}
    Return CS;
}
```

ALGORITHM: findMSP

INPUT: sample, CS
OUTPUT: MSP of sample

```
findMSP(sample, CS){
sample.visited = false;
 For count = 1 to CS.size{
  node = extractMember(CS);
 // CASE A: When CS is empty (initial state)
   If node == NULL{ …}
     Else if (node != NULL &&
        node.visited =    false){
   // CASE B: When sample has no parents
   // 0 denotes the disjoint relation with respect to
        //the'g' dimension
   If (g-SubsumptionMatch(sample, node) == 0)
   {
        MSP = NULL;
        return MSP; }
 // CASE C: When sample gets at least one
 // 1 denotes parent-child relation with respect to
        the 'g'dimension
 Else if (g-SubsumptionMatch(sample, node) == 1)
{
    node.visited = true; // this node won't be
//selected again
  If (node.childrensize != 0){
    For count = 1 to node.childrensize{
        node = node.child[count];
        findMSP(sample, node);} }
  Else{
        sample.parent[parentsize + 1] = node;
         sample.parentsize ++;
        node.child[childrensize + 1] = sample;
        node.childrensize ++;
       }
    }
  For count = 1 to sample.parentsize{
     MSP = MSP S sample.parent[count];}
    return MSP;
   }
  Else if (node != NULL && node.visited == true){
     continue;}
   }
}
```

ALGORITHM: findLSC

INPUT: sample, member of sample MSP, CS
OUTPUT: LSC of sample for the member of sample MSP

```
findLSC(sample, member, CS)
{ // CASE A: When sample has a non-empty MSP
 If member != NULL{
For count = 1 to member.childrensize{
If(g-SubsumptionMatch(member.child[count],
            sample) == 1){
LSC = LSC S member.child[count];}}
insertMember(CS, sample);
return LSC;
}
// CASE B: When sample has an empty MSP
Else{
For count = 1 to CS.size{
node = extractMember(CS);
If node.visited == false{
node.visited = true;
  If (SubsumptionMatch(node, sample) == 1){
    LSC = LSC S node;}
  Else continue;}
Else continue;}
insertMember(CS, sample);
return LSC; }
}
```

3.4 Web Service Matching

The Taxonomy clustering is essentially a subsumptive match based technique hence, we first need to compare any two given services with respect to a particular dimension (I, O, P, R) for three possibilities-(i) one is the ancestor/predecessor of the other (i.e., subsumption), (ii) both are sibling under a common parent, and (iii) both are mutually disjoint with no possible abstraction. For an example, assuming that two services *s1* and *s2* have their corresponding output dimension (i.e. O cluster space) to be *vehicle* and *SUV* (Fig 2). In such case both the services have a subsumptive relation with respect to the O dimension. Again, assuming that another service *s3* takes in *bus* as its output then *s3* and *s2* are siblings under the common parent *s1*. Fast subsumption testing of a well defined feature domain can be done using several ontology encoding techniques (such as interval encoding and prime encoding [4]).

The encoding technique used in the proposed work is to use bit codes. It use a '1' bit for a unique bit position (representing a unique characteristic

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

property) and inherit all the '1's of the parents into their corresponding positions. Thus, a root concept may be coded 001 while its sole child will be coded 011.

In general, the coding is done following a topological sort over the ontological domain space starting from the root concepts till the leaf concepts. If there is a multiple inheritance then the codes of the parents are ORed. Thus, for a root parent having code 001 and another root parent having code 010 their children will have the code 111. A subsumption can then be tested by doing a AND to check whether one of the operands is produced. The resulting operand denotes the ancestor while the other operand denotes its descendant. Thus, in the last example, for the codes 010 and 111, an AND produces 010 (Figure. 2). Although bit encoding is fast yet the former may not be efficient for huge domain space. This is because the space complexity of bit encoding is linear to the size of the domain space (i.e. O(*n*) bits where *n* is the number of concepts). The approach that is taken here is to break the operands into sufficiently smaller bit strings and then use a divide and conquer methodology for testing the subsumption. For any newly selected sample web service out of the sample space the objective is to find: (a) the most specific parent services in the existing cluster space under which it can be classified, and (b) the least specific children services that can be subsumed by it. Here it uses a SGPS match-making algorithm (called SGPS-Matching) based on dimension subsumption [16]. For two service samples to functionally match each other each with respect to a particular dimension (*I*, *O*, *P*, *R*) we need to compare concepts for subsumption that comprise the dimension. More precisely, the binary codes of the parameters within a dimension (say O cluster space) of a particular service sample are ORed together (if the parameters are more than one). The code (simple or ORed) of one service is then ANDed with that of the other service.

4. WEB SERVICE DISCOVERY

Web service discovery can be posed as a query matching problem. A user provides a query that typically involves a desired service. Existing services that match the desired service need to be found out of a large collection of services. This requires a pair-wise query matching between the functional dimensions of the query and the services. According to OWL-S [2], the functional dimensions of services constitute four dimensions: (i) *Input (I)*, (ii) *Output (O)*, (iii) *Pre-condition (P)*, and (iv) *Postcondition/Effect/Result* (R). Query matching for service discovery is most often based on the web service match model proposed by Paolucci et al [4]. We first need to discriminate the basic types of queries in terms of the four dimensions IOPR that can be possible over a service space.

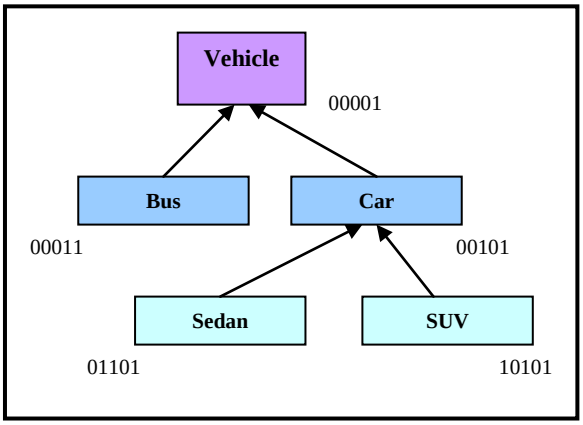

Figure 2: Bit-Encoding of Vehicle Ontology

4.1. Query Model for Service Discovery

Queries can be broadly classified into: (i) *simple queries*, (ii) *complex queries*, and (iii) *compound queries*. A simple query does not contain any conjunctive or disjunctive implication in its clauses. In other words, it constitutes only one query clause. An example would be *Q1*: "*find all services that provide vehicles*". A complex query constitutes a simple query whose clausal literals are constrained via the use of qualifiers. An example would be *Q2*: "*find all services that provide vehicles which operate within city X*". In this case the query has a simple query *Q1* whose clausal literal *vehicle* is qualified as: "*all vehicles which operate in city X*". Finally, a compound query is a partially ordered set (poset) of simple and/or complex queries logically associated via conjunctions and/or disjunctions. An example would be *Q3*: "*find all services that provide hospitals and find all services that provide names of neurologists attending all such hospitals and find all services that provide vehicle transportation to at least one of these hospitals*". A compound query is conventionally referred to as *tasks* in the literature [17]. Most work in service discovery and composition assume that queries are in the form of a task. However, this is a gross generalization and often leads to computationally expensive service discovery. This is because a

Journal of Theoretical and Applied Information Technology
20th October 2013. Vol. 56 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

consumer query is normally far from being a well-formed task. It is extremely difficult to articulate one's desire for a service by breaking it into concrete sub desires that are logically connected with each other so as to form a valid poset of desires. In [6] it is shown different types of queries that are used in service discovery. In the model proposed thereby a user initiated query is an ordered pair of two complex/simple sub queries in the form: <*input, query, desire*>.

Definition 4: *Type 2 Query* (in other words, the input required for the service) represents the information that is provided by the consumer to the system as a whole. Specifically, it is to search all services that have: (i) P dimension satisfied by the query, and (ii) I dimension satisfied by the query.
Definition 5: *Type 1 Query* (in other words, what the consumer desires as a service) represents the formal goal of the consumer (in other words). Specifically, it is to search all functionally similar services that have: (i) R dimension similar to desired R (or set of Rs), and/or (ii) O dimension similar to desired O (or set of Os).

4.2. Query Processing over Stratified Cluster Space

Initially when a query is received by the service discoverer it gets split into two parts: desire and input. For an example in Fig-3 it shows an initial query "*My name is Joe Smith. I live in Kansas City. I'd like to rent a car to go to Chicago*." This query can be formally split into Type 1 as: "<desire: *rent car*, restriction*: location* <source location: *Kansas City*, destination location: *Chicago*>>". In this example we see that the core desire is to rent a car while a location restriction is imposed over the desire as to where the source and destination location would be. Similarly, the original query is formally split into Type 2 as: "<input: *name* <value: *Joe Smith*>>". In the second step, Type 1 query is pushed into the O cluster space while Type 2 query is pushed into the I cluster space. The idea is to extract the set of all services from the O cluster space that satisfies (partially or completely) the Type 1 query while at the same time extracting the set of all services from the I cluster space that can take in the input (partially or completely) from Type 2 query.

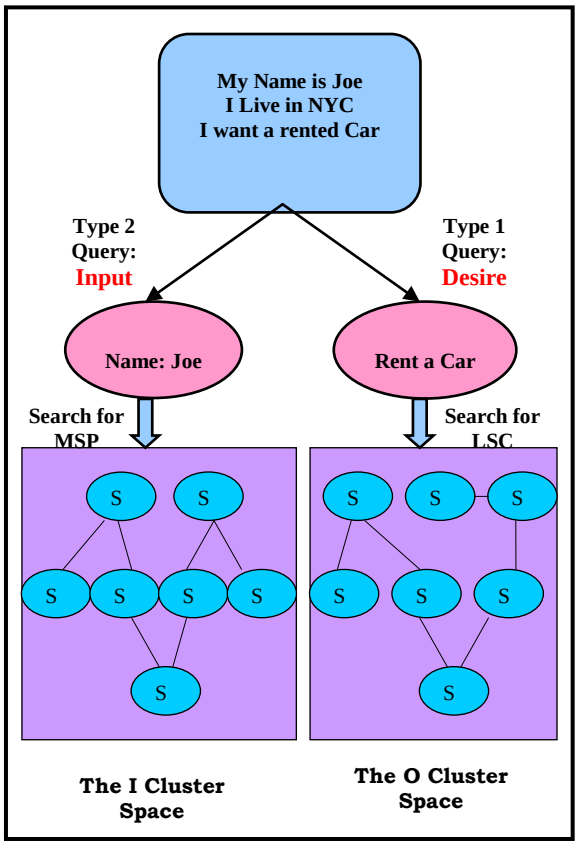

Figure 3. Matching Processing for Service Discovery

4.2.1Type 1 Query Processing

In order to extract services from the O cluster space the query processing algorithm tries to find the Least Specific Children (LSC) set of Type 1 query for a *strong solution set* (*SSS*). Once the LSC is found then all the services that are descendants of each member of the LSC within the O cluster space are included within the strong solution set. We choose the LSC (and subsequent descendants) because from a Type 1 query perspective we should be more interested in services that are guaranteed to produce the desired output. For the example discussed in this section we can have the set SSS as something like {*car rental service operating in Midwest, SUV rental service operating in Midwest, Sedan rental service operating in Midwest,…}* where all members guarantee the desired service. We also would like to keep a separate set of probabilistic (or *weak*) set of solutions for Type 1 query. In order to do that the query matching algorithm searches for the Most Specific Parent (MSP) set of Type 1 query and then includes the ancestor services of each of the MSP members into the *weak solution set* (*WSS*).

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4.2.2 Type 2 Query Processing

The same technique is applied over the I cluster space for the Type 2 query but in this case the *SSS* will include the MSP of Type 2 query (and all the member ancestor services) instead of LSC (which was the case for Type 1 query). This is so because for Type 2 query we are more interested in finding all those services that are guaranteed to take in the given input. Hence, a service requiring more specific input cannot guarantee execution all the time. For the given example the set of SSS for Type 2 query can be something like {*car rental service that has input name, bus rental service that has input name, age confirmation service that has input name,* …}. All of the services in this set guarantee that they can take in the input *name*. However, as with Type1 query, we can also create a separate WSS set for Type 2 query where the input processing is not guaranteed. As an example the WSS set can be something like {*jeep rental service that takes input first name, car rental service that has input last name,* …}. Over here if the query gives the full name it is not guaranteed that the service can accept the information *Joe Smith* in its entirety.

It is interesting to note that the SSS set for Type 2 can actually contain services that may not be related to the desire in a direct or indirect way. In the above example we can see that the Type 2 SSS set contains a member service *age confirmation*. Now even though this service is not a car rental service still it can be very significant for the discovery process. Imagine that the *age confirmation* service outputs *age verification certificate* and that is taken as an input by another car rental service within the Type 1 SSS/WSS set. In our query processing algorithm the third step, thus, involves checking composibility (direct or indirect) the SSS/WSS set of Type 2 query to the SSS/WSS set of Type 1 query (Fig. 3). The identification and selection of services that are not a one-to-one mapping with the query as such (like the example in this case) can be done because of: (a) the splitting of query into its desire (Type 1) and input (Type 2) and (b) the independent processing of Type 1 and 2 over the stratified O and I cluster spaces respectively. Comparing this technique to integrated approaches we can easily understand that such indirect matching between query and services is not at all possible.

Once the mapping between the solution set of Type 1 query and the solution sets of Type 1 query is done, it then filter a subset of the Type 1 query results such that all member services of the filtered subset are mapped, directly or indirectly, by at least one of the member services of Type 2 query solution sets. By indirect mapping we mean that the output of a member service in Type 2 result sets may not be directly the input of a member in the Type 1 result sets but instead can be mapped to some intermediate services within the I cluster space which in turn can be mapped to the Type 1 result sets. For an example, the *age confirmation* service in Type 2 SSS set can be mapped to an intermediate service *rental history lookup* service in I cluster space. This service may in turn output *rental history validation certificate* that may be required by yet another car rental service within the Type 1 SSS set as an input. So we can observe that as the service composition problem is totally dependent on the service discovery problem it may also be true that we may require some composition process even to discover the complete set of matching services.

5. EVALUATION

The proposed taxonomy based clustering algorithm has been shown to be free from the effect of sample selection order. This provides a significant edge over other known approaches with respect to accuracy. Moreover, the hit counts for clustering a particular sample are empirically seen to be extremely small as compared to the cluster space size. This provides significant improvement on the runtime performance as well. As query matching is a sub-problem of sample clustering we can also conclude that the average query response time should also be significantly low.

The evaluation methodology is three-folded:(A) runtime clustering performance evaluation, (B) runtime average query response time performance evaluation, and (C) accuracy (in terms of precision vs. recall and entropy) evaluation. The performances are measured based on: (a) randomly generated samples via simulation and (b) OWL-S TC v.2 test sets of 871 web services. Figure.4 shows the average query response time for these services.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

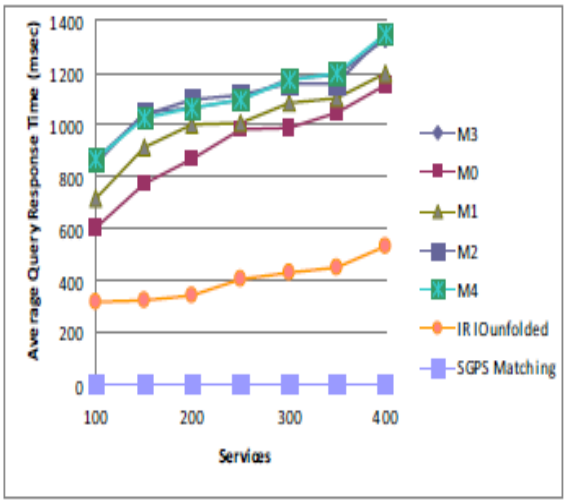

Figure 4. Average Query Response Time

6. CONCLUSIONS AND FUTURE WORK

The proposed taxonomic clustering algorithm has been shown to be free from the effect of sample selection order. This provides a significant edge over other known approaches with respect to accuracy. Moreover, the hit counts for clustering a particular sample are empirically seen to be extremely small as compared to the cluster space size. This provides significant improvement on the runtime performance as well. As query matching is a sub-problem of sample clustering, we can also conclude that the average query response time should also be significantly low. The query response analysis will be taken as a future work.

REFRENCES:

[1] Stoicay, I., Morrisz, R., Liben-Nowellz, et al. Chord: A Scalable peerto- peer lookup protocol for internet applications. IEEE/ACM Transactions on Networking (TON), Vol 11(1), 2003, pp. 17 – 32.

[2] Martin, D., et al. OWL-S: Semantic Markup for Web Services, W3C Member Submission, 22 Nov. 2004

[3] Zhao, W., Schulzrinne, H., Guttman, E. MSLP-Mesh-enhances Service Location Protocol, 9th ICCCN, Las Vegas, USA, 2000.

[4] Mokhtar, S., Preuveneers, D., Georgantas, N., Issarny, V., Berbers, Y. EASY: Efficient semantic Service discoverY in pervasive computing environments with QoS and context support, J. of Systems and Software, 81(5):785-808 2008.

[5] Tosic, V., Patel, K., Pagurek, B. WSOL - Web Service Offerings Language, International Workshop on Web Services, E-Business and the Semantic Web (WES 2002), CAiSE, Toronto, Canada, 2002.

[6] Dasgupta, S. Bhat, S., Lee, Y. An Abstraction Framework for Service Composition in Event-driven SOA systems, IEEE ICWS, Los Angeles, CA, USA, 2009

[7] Corella, M. A., Castells, P. A Heuristic Approach to Semantic Web Services Classification, 10th International Conference on Knowledge- Based & Intelligent Information & Engineering Systems (KES).2006.

[8] Bianchini, D., Antonellis, V., Pernici, B., Plebani P. Ontology-based Methodology for e-Service Discovery, ACM J.l of Information Systems, Vol 31(4): 361 – 380, June 2006.

[9] Wang, G., Xu, D., Qi, Y., Hou, D. A Semantic Match Algorithm for Web Services Based on Improved Semantic Distance, 4th International Conference on Next Generation Web Services Practices, Seoul, S. Korea, 2008.

[10] Paolucci, M., Kawamura, T., Payne, T., Sycara, K. Semantic Matching of Web Services Capabilities, International Semantic Web Conference, Italy, 2002

[11] Rada, R., Mili, H., Bicknell, E., Blettner, M. Development and application of a metric on semantic nets, IEEE Trans. on Systems, Man, and Cybernetics, 19(1):17-30 1989.

[12] Hirst, G., St-Onge, D. Lexical chains as representations of context for the detection. Fellbaum, 1998, pp. 305–332.

[13] Resnik, P. Using information content to evaluate semantic similarity, 14th International Joint Conference on Artificial Intelligence, Montreal, 1995, pp. 448–453.

[14] Keßler, C., Raubal, M., Janowicz, K. The Effect of Context on Semantic Similarity Measurement, Lecture Notes in Computer Science 4806. Springer-Verlag Berlin Heidelberg, 2007, 1274-1284.

[15] Qiu, T., Li, L., Lin, P. Web Service Discovery with UDDI Based on Semantic Similarity of Service Properties. 3rd International Conference on Semantics, Knowledge and Grid, Xi'an, China, 2007.

[16] Sourish Dasgupta, Satish Bhat, Yugyung Lee, SGPS: a semantic scheme for web service similarity. WWW 2009: 1125-1126.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[17] J. Sousa, V. Poladian, D. Garlan, B. Schmerl, M. Shaw. "Task-based Adaptation for Ubiquitous Computing", IEEE Transactions on Systems, Man, and Cybernetics, Vol. 36(3), 2006. pp. 328-340.

[18] Klusch, M., Fries, B., Sycara, K. OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services, Web Semantics: Science, Services and Agents on the World Wide Web, 7(2):121-133, 2009.