Repository Filter with Semantic Heterogeneity Reconciliation Using WordNet for Efficient Semantic Web Service Discovery

Mrs. Tanuja P. Lonhari
Department of Computer Engineering
D. Y. Patil College of Engineering
Pune, India

Mrs. D. A. Phalke
Department of Computer Engineering
D. Y. Patil College of Engineering
Pune, India

***Abstract*—Popularity of Semantic Web is increasing by the day. Ontology is a key building block of Semantic Web Applications. It represents information in a particular domain in a well-structured manner. It is used to share common information about the domain amongst entities working in that domain so that when multiple entities interact with each other, they can do so in an unambiguous manner. Even though Semantic Web has been around for a few years, there are very few standard ontologies available for use. This gives rise to semantic heterogeneity. The proposed system tries to reduce the semantic heterogeneity using WordNet from a repository filter mechanism used to improve performance of Sematic Web Service (SWS) discovery mechanism.**

***Index Terms*—Semantic heterogeneity reconciliation, WordNet, Semantic Web Service discovery, repository filtering, SPARQL.**

I. INTRODUCTION

As Semantic Web is gaining popularity, number of available Semantic Web Services (SWS) is also increasing rapidly. Automatic discovery of relevant services is the key feature to improve usability of these services. As the users do not have any prior knowledge of published services, an efficient search mechanism is crucial to improve usability of services. Currently, there are several different approaches being followed for discovery of SWS. Logical reasoning-based matching, graph-based matching, hybrid matching are some of them [2]. All the existing discovery mechanisms for SWS are fairly computation intensive and do not scale well when the number of published services grows or complexity of the domain ontology increases.

Several mechanisms have been suggested to overcome the scalability issue of the discovery mechanisms. Amongst the suggested approaches are indexing and caching techniques, clustering mechanisms, and other preprocessing steps. An effective preprocessing mechanism is the repository filtering mechanism based on SPARQL queries [5]. This mechanism analyses user query and extracts concepts defining various terms in the service annotation ontology. A SPARQL query is formed based on these concepts. This query finds out those service descriptions present in the repository that contain some or all the concepts mentioned in the user query filtering out all the rest of descriptions. The service descriptions selected by the filter are passed as input to the subsequent discovery mechanism. This filtering mechanism considerably reduces the search space and hence improves performance of the discovery mechanism.

In Semantic Web, ontologies present information in a particular domain in a well-structured manner. It is used to share common information about the domain amongst entities working in that domain so that when multiple entities interact with each other, they can do so in an unambiguous manner [12]. For example, if the service requester and service provider refer to same ontology, the provider will exactly know what the requester is looking for. Most of the matchmaking services and also the repository filtering approach mentioned earlier assume that concepts from service description and service request refer to the same ontology but this rarely happens in an open environment such as Internet where the Web Services reside. In such a situation service providers providing similar services might refer to different ontologies. This results into semantic heterogeneity. If the service request and service advertisements are not referring to same ontology, the discovery or filtering mechanism fails to retrieve a service even if it is relevant for the request. This reduces recall of the mechanism considerably. The proposed system uses a technique based on WordNet to improve recall of the repository filtering mechanism.

II. RELATED WORK

Web Services essentially follow Service Oriented Architecture (SOA). In this publish-find-bind architecture the client of the service invocation has no prior knowledge of the service description and hence cannot link in pre-compiled stubs. Service discovery plays the most important role in this architecture and has been the subject of major research work going on in the area. Inclusion of semantic annotations to the service description in Semantic Web Services greatly improves efficiency of discovery mechanisms and enables automatic discovery of services.

There are several different approaches being used for discovery of SWS. Service annotation ontology allows semantic services to be described in terms of their capabilities and the functionality they provide. In general, service ontologies describe inputs, outputs, preconditions, and effects (IOPE) for a service. In one of the very first algorithms proposed for SWS discovery, Paolucci, Kawamura, Payne, and Sycara proposed a logical reasoning-based matching of service request with service advertisement [10]. The proposed system extends the UDDI service advertisement to incorporate semantic information about inputs and outputs of the service. It then matches all the request outputs with the outputs of advertisements and all request inputs with inputs of the advertisements using logic-based matching and assigns a degree of match as one of the following: Exact, Plugin, Subsumes, or Fail. Later on Srinivasan, Paolucci, and Sycara implemented the same algorithm for the services which were annotated using OWL-S [11]. Both these approaches use UDDI as the repository of services. The advantage of these approaches is the fact that UDDI is already well established service registry and enjoys considerable industry support.

In another approach, the matching service is published as an external service in UDDI. This provides a more seamless integration of semantic matching mechanism with UDDI registry. In this approach, the matching service providers publish their services in UDDI as normal web services [3]. This architecture eliminates the need of installing matchmaking infrastructure either on the registry side or on the user side. This makes the matchmaking process more flexible where user can choose from multiple matchmaking services developed by independent vendors instead of just the one hardwired in UDDI. This approach also allows user and service provider to use different semantic markup languages, namely OWL-S, UML, XML, or any other custom format. Meditskos and Bassiliades have proposed a matchmaker that matches user request based on similarity between keywords in the request with keywords in service descriptions [7].

The authors Klusch, Fries, and Sycara have implemented a hybrid algorithm based on similarity measures in information retrieval techniques. OWLS-MX is a hybrid matchmaker that complements the logic-based reasoning with syntactic IR based similarity comparisons [6]. Authors have shown that under certain constraints, the logic-only based algorithms for service I/O matchmaking are outperformed by this hybrid algorithm. The authors Bellur and Kulkarni have proposed a novel method of SWS discovery. The method considers matching of output and input concepts of the request and service advertisement. It constructs a bipartite graph using the concepts of output concepts from request and advertisement respectively and tries to find the matching bipartite graph [1] using standard graph matching algorithms.

There are several preprocessing methods that employ clustering based approaches to improve efficiency of discovery mechanisms. Conventional web services published in UDDI are categorized depending on some standard classification such as North American Industry Classification System (NAICS), but service provides may not always publish their services under the relevant category. As a result, a service published in an unrelated category might not get selected even though it provides relevant outputs. To overcome this, Paliwal et al. have proposed a clustering on top of the existing categorization. Their solution includes a service categorization module that first matches the service description to a standard ontology with the help of WordNet and then clusters web services into functional categories depending on ontology concepts [9]. In another clustering-based approach proposed by Nayak and Lee [8], Jaccard Coefficient method is used to calculate semantic similarity between service descriptions and clustering is performed based on this similarity value. The system proposed by Elgazzar, Hassan, and Martin [4] mines WSDL documents to extract semantic concepts and then clusters the services in functionally similar groups.

There are thousands of services already available in the public domain and with the interest generated by Semantic Web Services; the number is expected to explode in future. This will necessitate large and complex repositories. The existing matchmaking algorithms are considerably heavyweight mechanisms since the ontologies expressed using existing SWS frameworks such as OWL-S or WSMO present a high complexity in defining and processing them. Due to this reason and with exponentially growing numbers of available services, the discovery mechanisms will face scalability issues. The main bottleneck is caused by the reasoning facilities which need to match hundreds of ontologies in order to fetch relevant services. Garcia, Ruiz, and Ruiz-Cortes [5] have proposed a novel solution that aims at alleviating the scalability issue of discovery mechanisms. It introduces a preprocessing step that applies a SPARQL-based filtering mechanism to reduce the search space of the discovery process. Since the input size of the reasoning facility is reduces, it brings marked improvement in the performance of the discovery mechanism.

III. Implementation Details

To date, there are very few standard domain ontologies available covering major domains. Due to this reason, different service providers developing services for the same operative domain may refer to different ontologies. This will result into web services providing similar services but referring to concepts from different ontologies and introduces semantic heterogeneity. As a result, the matchmaking services or repository filter will fail to retrieve some services even if they are relevant to the request if request and service description refer to different ontologies. To overcome this problem, the proposed system introduces an **ontology scan** module to the filtering mechanism. It uses a technique based on WordNet to reduce the semantic heterogeneity and improve recall of the repository filtering mechanism. WordNet is a publicly available lexical database in which nouns, verbs, adjectives, and adverbs are arranged into sets of synonyms.

For example, Listing 1 and 2 show service advertisements for CarPriceService and AutoCostService respectively. Both of these services provide price of a car but they refer to different ontologies. If a user gives a request with concepts as

Car and Price, then repository filter will not select AutoCostService as a probable match because of the semantic heterogeneity introduced by different ontologies. As a result AutoCostService will not be selected even if it is a relevant service. The ontology scan module in the proposed system helps reconcile this heterogeneity by including the concepts in the user request as well as all the synonyms of all the concepts in the generated filter. Listing 3 shows the filter generated by the proposed system. This filter will be able to select both CarPriceService and AutoCostService.

```
@prefix profile: <http://www.daml.org/services/owl-s
/1.1/Profile.owl#>.
@prefix process: <http://www.daml.org/services /owl-s
/1.1/Process.owl#>.
@prefix portal: <http://purl.org/iserve /ontology /owlstc
/portal.owl#>.
@prefix travel: <http://purl .org/iserve /ontology
/owlstc/travel.owl#>.

:CarPriceServiceProfile a profile :Profile ;
profile:hasInput :Car;
profile:hasOutput :Price.

:Car a process :Input ;
process:parameterType travel:Car.
:Price a process :Output ;
process:parameterType travel:Price.
```

Listing 1: Service Advertisement for CarPriceService

```
@prefix profile: <http://www.daml.org/services/owl-s
/1.1/Profile.owl#>.
@prefix process: <http://www.daml.org/services /owl-s
/1.1/Process.owl#>.
@prefix myOntology: <http://des.org/data /ontology /owlstc
/myOntology.owl#>.

:AutoCostServiceProfile a profile :Profile ;
profile:hasInput :Automobile;
profile:hasInput :Make;
profile:hasInput :Model;
profile:hasOutput :Cost.

:Automobile a process :Input ;
process:parameterType myOntology:Automobile.
:Make a process :Input ;
process:parameterType myOntology:Make.
:Model a process :Input ;
process:parameterType myOntology:Model.
:Cost a process :Output ;
process:parameterType myOntology:Cost.
```

Listing 2: Service Advertisement for AutoCodstServiceService

```
SELECT DISTINCT ? service
WHERE {
?service a service:Service ;
service:presents ?profile .
# match all inputs and outputs of the profile...
?profile profile:hasInput ?inputTerms.
?profile profile:hasOutput ?outputTerms.

{?inputTerms process:parameterType travel:Car}
UNION
{?inputTerms process:parameterType myOntology:Automobile
}
UNION
{?inputTerms process:parameterType myOntology:Make}
UNION
{?inputTerms process:parameterType myOntology:Model }
UNION
{?outputTerms process:parameterType travel:Hotel }
UNION
{?outputTerms process:parameterType travel:Hotel }
}
```

Listing 3: Generated SPARQL filter

A. System Overview

Architecture of the proposed system is shown in figure 1.

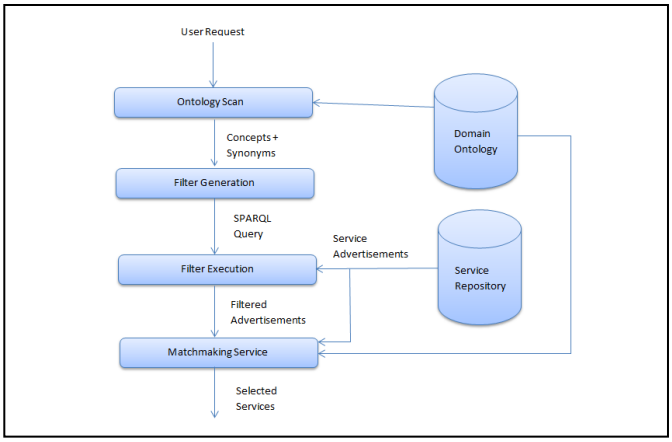

Fig. 1. Architecture of Proposed System

The system contains following modules.

1. Ontology Scan: This module finds all synonyms for each concept extracted from user request. After concepts are extracted from user request, this module iterates over all the concepts and finds synonyms for each concept with the help of WordNet database. All the available ontologies are scanned to find if any of the synonyms of extracted concepts exist in any of these ontologies. If synonyms of any of the concepts are present in available ontologies, these are added to the array of concepts.

2. Filter Generation: All the concepts from user request as well as any synonyms present in the existing ontologies are

used to the generate SPARQL query which is used as filter. This enables selection of services which refer to same concepts but from different ontologies. This improves recall of the filtering mechanism.

3. Filter Execution: The generated SPARQL query is executed against all the registered service advertisements present in service repository. This filter selects only those advertisements that contain one or more concepts from user request or their synonyms rejecting rest of the advertisements. Only the service advertisements selected by the filter are sent to subsequent matchmaker service.

B. Mathematical Model

Let S = {O, A, R, Q} be the system where

$O = O_1 \cup O_2 \cup \ldots\ldots\ldots \cup O_n$ is the set of domain ontologies.

$A = A_1 \cup A_2 \cup \ldots\ldots\ldots \cup A_m$ is the service advertisements registered in service repository.

$R = R_1 \cup R_2 \cup \ldots\ldots\ldots \cup R_x$ is the set of user requests.

Each services advertisement A_i contains several input and output terms defining capabilities of the service. Let AC_i represent collection of input and output terms of advertisement A_i.

$AC_i = \{C_{Ai1}, C_{Ai2}, \ldots\ldots, C_{Ain}\}$

Each user request R_i also contain several input and output concepts. Let RC_i represent collection of concepts in request R_i.

$RC_i = \{C_{Ri1}, C_{Ri2}, \ldots\ldots, C_{Rim}\}$

Ontology scan mechanism tries to find all available synonyms for each concept in the request such that

$RC'_i = \{C_{Ri1}, S_{C11}, S_{C12}, \ldots, S_{C1m}, C_{Ri2}, S_{C21}, S_{C22}, \ldots, S_{C2m}, \ldots\ldots, C_{Rim}, S_{Cn1}, S_{Cn2}, \ldots, S_{Cnm}, \}$ where $S_{Ci1}, S_{Ci2}, \ldots, S_{Cim}$ are synonyms of concept C_{i1}.

Filter generator module generates a SPARQL query Q based on all concepts from RC'_i that includes original concepts from user request R_i as well as all the synonyms.

Filter execution module executes the generated query Q against the service advertisements A present in the repository. It selects only those advertisements that contain one or more concepts from RC'_i and filter out rest of the advertisements. It can be represented as

$F = \{A_i | AC_i \cap RC'_i \neq \varnothing \}$

C. Algorithms

Algorithm 1 is the algorithm of the proposed system.

1) Accept user request.
2) Concepts = Extract input and output concepts from user request.
3) Find synonyms of each concept in Concepts.

For each c of Concepts **do**
synonyms = findSynonyms(c)
Concepts = Concepts + synonyms
End for

4) Filter = Generate SPARQL query based on all the concepts in Concepts.
5) Execute Filter against all advertisements in service repository.
6) Pass service advertisements selected by the filter to matchmaking engine.

Algorithm 1: Algorithm of the proposed system

IV. RESULTS

A. Data Sets

The experimental evaluation of the proposed system uses a dataset called as OWLS-TC V3. It is a widely used test collection of service advertisements and service requests. This collection contains 1007 service descriptions from seven different domains such as education, medical care, food, travel, communication, economy, weapons. The advertisements are service profiles created using OWL-S framework. It also contains 29 user requests that can be matched against the service description. The collection also lists relevant services against each request so that the performance of the matchmaker can be evaluated against it. It is available at SemWebCentral.org. Evaluation is performed on a system having 1.71 GHz CPU, 4GB RAM, Windows 7 SP1 Operating system and with Java 7.

B. Result Set

Experiments have shown that with the introduction of **ontology scan** module, recall of the repository filter is improved by 7% to 35%. Experiments were conducted for all the requested included in the test collection. Results for few of the requests have shown below.

	Precision	Recall	Precision with Ontology Scan	Recall with Ontology Scan	Improvement in recall
book_price_service	28%	62%	34%	66%	7%
car_price_service	39%	71%	45%	74%	4%
hospital_investigating_service	100%	48%	100%	58%	22%
novel_author_service	95%	81%	96%	84%	3%
shoppingmall_cameraprice_service	12%	80%	19%	86%	7%
university_lecturer-in-academia_service	100%	62%	100%	74%	20%
geopolitical-entity_weatherprocess_service	29%	38%	49%	51%	35%
surfing_destination_service	92%	70%	94%	75%	7%
grocerystore_food_service	100%	57%	100%	68%	19%

V. CONCLUSION

Web Services are self-contained, self-describing components that publish a well-defined interface. It allows programmers to build complex applications by using Web Services as building blocks. Conventional keyword-based search mechanism fails to discover most relevant services. Thus, they fail in achieving their main design goal, namely automatic discovery and composition of services. Semantic annotations provide a way to express capabilities of Web Services in a machine readable format, thus allowing the services to be discovered automatically. There are various approaches being used for semantic matchmaking. These existing mechanisms are computation intensive and face scalability issues when number of published services or

complexity of ontology increases. Repository filtering approach aims at overcoming this scalability problem by reducing the size of the search space for the matchmaking algorithm and considerably improves response time. The proposed system improves performance of the repository filtering mechanism by introducing an ontology scan module that includes request concepts as well as all its synonyms to the generated filter so that it will be able to select services even if they are not referring to the common ontology.

ACKNOWLEDGMENT

It gives me immense pleasure to present this paper. I wish to thank all the people who gave me an unending support directly or indirectly. I express my sincere and profound thanks to our teachers \textbf{Mrs. M. A. Potey} (Head of Department), \textbf{Ms. S. S. Pawar} (PG Co-ordinator). It would not have been possible to complete the work without their kind support and help. I want to extend my special thanks to \textbf{Mrs. D. A. Phalke} (Project Guide) for her guidance and constant supervision. I am also thankful to all my classmates who have helped me in the preparation of this seminar and I would also like to thank our college, D. Y. Patil College of Engineering, Akurdi and the Computer Department for providing all the necessary resources.

REFERENCES

[1] Bellur, U. a. (2007). Improved matchmaking algorithm for semantic web services based on bipartite graph matching. *IEEE International Conference on Web Services, ICWS.* (pp. 86--93). IEEE.

[2] Cardoso, J. (2007). *Semantic Web Services: Theory, Tools and Applications.* IGI Global.

[3] Colgrave, J. a. (2004). External matching in UDDI. *IEEE International Conference on Web Services. Proceedings.* (pp. 226--233). IEEE.

[4] Elgazzar, K. a. (2010). Clustering wsdl documents to bootstrap the discovery of web services. *IEEE International Conference on Web Services (ICWS)* (pp. 147--154). IEEE.

[5] Garcia, J. M.-C. (2012). Improving semantic web services discovery using SPARQL-based repository filtering. *Web Semantics: Science, Services and Agents on the World Wide Web*, Elsevier.

[6] Klusch, M. a. (2006). Automated semantic web service discovery with OWLS-MX. *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, 915--922.

[7] Meditskos, G. a. (2010). Structural and role-oriented web service discovery with taxonomies in OWL-S. *IEEE Transactions on Knowledge and Data Engineering*, 278--290.

[8] Nayak, R. a. (2007). Web service discovery with additional semantics and clustering. *IEEE/WIC/ACM International Conference on Web Intelligence* (pp. 555--558). IEEE.

[9] Paliwal, A. V. (2012). Semantics-based automated service discovery. *IEEE Transactions on Services Computing*, 260--275.

[10] Paolucci, M. a. (2002). Importing the Semantic Web in UDDI. *Springer*, 225--236.

[11] Srinivasan, N. a. (2005). An efficient algorithm for OWL-S based semantic search in UDDI. *Springer*, 96--110.

[12] Staab, S. a. (2009). *Handbook on ontologies.* Springer.