Semantic Web Service Discovery

Ezdehar Jawabreh

Abstract

Nowadays, Web services play a vital role in software development. This is due to their distinguished characteristics such as being self-contained, reusable, composable, and highly interoperable. However, the number of published Web services is increasing every day. Currently, researchers face many challenges due to this large number. The discovery, selection, and composition of Web services are becoming harder to accomplish perfectly. Several solutions are needed to filter services to obtain the best-matched according to user functional and non-functional requirements. Among these solutions is the use of semantic Web services which facilitate the discovery of suitable Web services. In this paper, we reviewed the literature in order to understand state-of-art achievements in this field. In addition, we implemented a semantic Web service discovery algorithm that automated the discovery of services. The algorithm utilized two types of ontologies which were the OWL-S ontology of Web services and the WordNeT ontology. We evaluated it using the SWS Test Collection dataset [1], the results showed that the algorithm has a good precision rate and thus can be effectively used in the discovery of services.

1 Introduction

Web Service is defined as "A software system designed to support interoperable machine-to-machine interaction over a network" [2]. Recently Service Oriented Architecture (SOA) has become a major paradigm in system engineering. Many systems are constructed by integrating services as the basic unit of building. In the service selection problem, it is crucial to select the right service in order to achieve high system stability and user satisfaction. In order to achieve an appropriate selection, a discovery algorithm is needed to obtain the candidate services that match both user/system functional and non-functional requirements. Usually, Web services utilize a set of XML-based standard specifications for describing, registering, and binding them, examples of these standards are the Web Service Description Langauge (WSDL) for describing the interface of service, and the Universal Description Discovery and Integration (UDDI) as a registry for publishing services, in addition these standards are based on the SOAP protocol to exchange messages between different parties.

In reality, the number of web services is increasing tremendously, and this increase hinders the tasks related to their discovery, selection, and composition. However, the previously mentioned standards describe services syntactically and lack the expressiveness of semantics which again will harden the discovery and selection of Web services. To mitigate the problem, the suggested solutions in literature are utilizing the semantic Web to describe services as Web resources, resulting in a so-called semantic Web services.

Semantic Web services is depending on ontologies to annotate Web services in a way that makes them more interpretable, machine-readable, and self-describing, in order to facilitate their discovery and other related tasks. There are several ontologies existing in the literature to describe the semantics of Web Services such as OWL-S, WSMO, WSDL-S, and SAWDSL [3].

In this paper, we reviewed the state-of-art studies that used semantic Web services in discovery and composition tasks. We synthesized them in order to understand the challenges they addressed. In addition, we provided an implementation for an algorithm that automates the discovery of web services. The algorithm utilizes OWL-S services ontology and WordNet [4] as another intermediate domain ontology. We conducted several experiments to demonstrate the validity of the algorithm. The experiments utilized SWS Test Collection [1] and the results showed that the algorithm achieved a good precision rate against a predefined set of user queries.

2 Related Work

In a recent study [7], the authors proposed an approach for the discovery, selection, and composition of semantic Web services, it used the semantic similarities between OWL-S files of Web services and user queries on both user functional and non-functional requirements, they enriched the nonfunctional requirements with three types which are: Quality of service (QoS), Quality of Experience(QoE) and Quality of Business(QoBiz), their approach achieved high degrees of accuracy and efficiency.

Another work [8] created an ontology for pharmaceutical Web services using OWL-S language, in addition, they created a pharmaceutical domain ontology to add more semantics when describing Web services in their field. In general, their approach improved the service quality in the healthcare field. In [11] the authors proposed an algorithm for semantic service discovery that used input and output similarity, it enhanced the discovery process by matching the precondition and effect of user query along with the candidate services, the approach achieved high precision compared with other algorithms.

In another study [6] the authors extended the OWL-S ontology and provided a new one called Comp-O that contained the semantics needed to build component-based applications automatically. In [9] the authors proposed a deep learning approach (encoder-decoder) to cluster services according to functional and non-functional requirements using their OWL-S files. The authors also extended the OWL-S file by providing a description of QoS (i.e the nonfunctional requirements of the services), at the end, they provided an adaptive service matching method with QoS threshold to discover and rank services.

The work in [10] reviewed the state-of-the-art work in semantic web service selection, it also proposed two approaches based on network flow-based algorithms to enhance the service selection and to maximize the matching with user queries. In a recent work, [12] the authors proposed a method for semantically discovering microservices published in Swagger API, their approach used domain ontologies to compute the input and output similarities of Web services with respect to a given user query.

3 Semantic Service Discovery

Semantic service discovery is the process of finding services that match a given user query. Usually, the user query describes the needed services in an ontological manner by giving information about its interface such as the number and names of input/output parameters. However, a service discovery algorithm implements both syntax and semantic procedures to find the maximum matched services. Utilizing semantic discovery will improve the accuracy of such an algorithm and maximize the number of candidate services that achieve user goals. For example, consider user searches for services that return the prices of cars, if semantic search is used, then he will get (car, price), (automobile, price), and (vehicle, price) as outputs since both car, automobile, and vehicle are close in semantics.

In the following subsection, we will provide a detailed description of the implemented semantic discovery algorithm.

3.1 The Semantic Discovery Algorithm

The implemented algorithm consists of three main filtration steps summarized in algorithm 1. The following points provide more clarification of these steps.

3.1.1 First Filtration step

In the first step, the algorithm finds the syntactically matched services, this is done by comparing the number of input and output parameters of each service with a given user query. A service is considered syntactically equal to a user query if the number of its input and output parameters is equal to the number of input and output parameters of the user query.

3.1.2 Second Filtration step

In the second filtration step, the set of syntactically equal services is filtered more to get the set of Semantically Matched Services (SMS). This is performed using the WordNet Similarity for Java (WS4J) library [5], which provides a pure Java API that supports a set of algorithms measuring

semantic similarity, from the supported algorithms we utilized the WuPalmer algorithm [13] that computes similarity by considering the depths of the two synsets in the WordNet ontology and returns a value between 0 and 1.

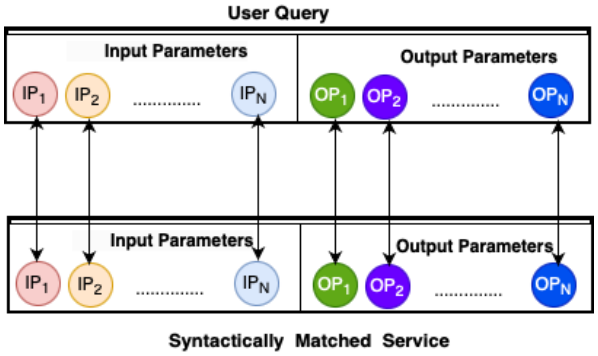

Figure 1: Similarity Match for Input and Output parameters

The algorithm computes the similarity degree at the level of input parameters concepts which we called the Final Input Similarity (FInputSim), also it computes the Final Output Similiarty (FOutputSim) at the level of the output parameters concepts.

Equation 1 [12] shows how the FInputSim is calculated as the average of similarity degree between each input parameter in the user query with its corresponding input parameter in a syntactically matched service as shown in Figure 1.

$$FInputSim = \frac{1}{count(IP)} \sum_{IP \in a,q} Similarity\_Degree(IP_a, IP_q) \tag{1}$$

Where count(IP) is the number of Input Parameters, (a) is the syntactically matched service, and (q) is the user query. Similarly, The FOutputSim is calculated as the average of similarity degree between each output parameter (OP) in the user query (q) along with its corresponding output parameter in the service (a), as shown in equation 2 [12].

$$FOutputSim = \frac{1}{count(OP)} \sum_{OP \in a,q} Similarity\_Degree(OP_a, OP_q) \tag{2}$$

The final similarity for each service is computed from the average of both Input and output similarities. see equation 3.

$$FinalSim = \frac{1}{2}(FInputSim + FOutputSim) \tag{3}$$

3.1.3 Third Filtration step

In the third filtration step, a user-predetermined threshold is used to filter the syntactically matched service based on their final similarity. Eventually, a set of Semantically Matched Services (SMS) is returned to the user as a final result as explained in equation 4.

$$SMS = \{S | FinalSim(S) \geq Sim\_Threshold\} \tag{4}$$

Algorithm 1 Web Service Semantic Discovery Algorithm

INPUT: ServRegistery(ServName,IP,OP),UserQuery(UIP,UOP), Sim_Threshold, WordNet ontology.
OUTPUT: Semantically Matched Services (SMS)

// Initialize the Count of input and output param.
for $S \in ServRegistery$ **do**
    Count number of IP and OP.
end for
// Find Functionally(Syntax) Matched Services.
for $S \in ServRegistery$ **do**
    if $count(S.IP) = count(UIP)$ *and* $count(S.OP) = count(UOP)$ **then**
        add S *to* $Match\_Syntax\_Services(MSS)$
    end if
end for
if count (MSS)=0 **then** return No Match service
end if
//Find Semantic Matched Service (SMS)
for $S \in MSS$ **do**
    // Calculate Input Similarity
    FInputSim= Equation 1
    // Calculate Output Similarity
    FOutputSim= Equation 2
    // Find Final Total Sim
    $FinalSim = 0.5 \times (FInputSim + FOutputSim)$
end for
//Filter Matched services according to Threshold Value
for $S \in MSS$ **do**
    if $S.FinalSim \geq Sim\_Threshold$ **then** *add* S *to* SMS
    end if
end for
return SMS

4 Experiments

In order to validate the algorithm, we conducted a set of experiments that cover a variety of queries performed by the user. Next, we will explain the steps we follow to perform the experiments.

4.1 Test Suite Preparation

Our experiments used the test suite that is publicly published on $SWS\_Test\_Collections$. We utilized the OWLS-1.1 (latest version) folder which contains (1083) services from different domains including communication, weapon, education, travel, medical, food, geography, and economy. However of these services, we excluded the ones that miss essential parts such as service description and presentation, in addition, we excluded services that have syntax errors. Eventually, we have a final number of (1006) services that match our final goals and that were used as the basis for the incoming experiments. The final set of services along with the algorithm implementation are provided on a public Github link [1].

To simulate user queries we prepared a set of 10 queries as shown in Table 1. A query is described by the user by giving the names of input and output parameters along with the similarity threshold he wants to use for semantically matching the available services. In our experiments and as shown in the mentioned table we fixed the threshold to a value of 0.8 which is selected experimentally in order to receive reasonably matched services, however, if an exact match is required by the user, then the highest degree of 1.0 can be provided. The last column in the table represents the ground truth for the number of services available in the service registry for each query. This number is calculated using human experts after investigating the registry for each given query.

[1] https://github.com/ezdehar1/Data_Engineering-Course

Table 1: User Queries Set

Q.No	Query	Input Param.	Output Param.	Sim_Threshold	Available No.
1.	Title_film	Title	Film	0.8	9
2.	Retialstore_food	Retial_store	Food, Quality	0.8	1
3.	Sport_destination	Sport	Destination	0.8	15
4.	Car_price	Car	Price	0.8	12
5.	ISBN_book_author	ISBN	Book, Author	0.8	2
6.	Researcher_address	Researcher	Address	0.8	13
7.	Missile_financing	Missile	Financing	0.8	3
8.	Care_diagnostic	Care	Diagnostic, Time	0.8	4
9.	Award_scholarship	Award	Scholarship	0.8	2
10.	Country_weather	Country	Weather	0.8	9

4.2 Service Registry Creation

The service registry is created by uploading the OWL-S files of the services to the GraphDB [2] store. A set of Sparkle commands shown in Figures 2 and 3 are used to extract the required components of the registry including service name, input parameters, and output parameters.

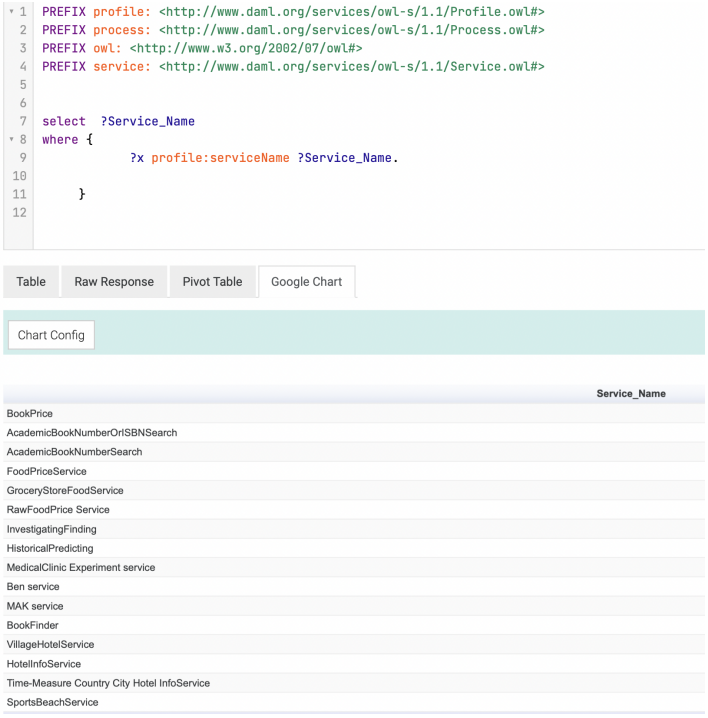

Figure 2: Service Name Extraction

4.3 Evaluation Metrics

Evaluation metrics are used to measure the deviation of the results of the algorithm from the ground truth, we used three metrics: precision equation 5 which measure how many true services were returned from the total number of returned services. Recall equation 6 which measures how many true services returned over all true services that should be returned. and the F1 score equation 7 which is a harmonic

[2] https://graphdb.ontotext.com

```
select  ?Input_Parameter
where {
            ?x profile:hasInput ?Input_Parameter.

    }
```

(a) Output Sparkle Query

```
select  ?Output_Parameter
where {
            ?x profile:hasOutput ?Output_Parameter.

    }
```

(b) Input Sparkle Query

Figure 3: Sparkle Query

mean of both precision and recall.

$$Precision = \frac{TPS}{TPS + FPS} \tag{5}$$

$$Recall = \frac{TPS}{TPS + FNS} \tag{6}$$

Where TPS is the True Positive Service, FPS is the False Positive Service, FNS is the False Negative Service.

$$F1\_Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{7}$$

4.4 Results and Discussions

In this subsection, we show the results obtained from running the prepared user queries, we elaborate on the results, discuss them and provide the reader with possible enhancements and threats to validity.

4.4.1 Results

Table 2 shows the obtained evaluation metrics for each user query. We can notice that for most queries, the precision was higher than the recall, indicating that the algorithm rarely returned unmatched services(i.e., the false-positive ratio is low). However, the majority of queries have lower recall values which means that the algorithm may miss some of the true services (i.e. false-negative ratio is high). On average, the algorithm produced a precision score of 81%, recall score of 67%, and F1_score of 66%.

Table 2: Results of semantic service discovery algorithm

Q.No.	TPS	FPS	FNS	Precision	Recall	F1 Score
1	3	3	6	5%	33%	40%
2	1	3	0	25%	100%	40%
3	9	0	6	100%	60%	75%
4	5	0	7	100%	42%	58%
5	2	0	0	100%	100%	100%
6	8	0	5	100%	62%	76%
7	3	3	0	50%	100%	66%
8	3	0	1	100%	75%	86%
9	1	0	1	100%	50%	67%
10	4	1	5	80%	44%	57%
Average				81%	67%	66%

Obtaining a high precision score (81%) indicates that users can be confident enough to deal with the algorithm, since the majority of returned services are suitable ones, and it is safe to be used for service-related tasks such as the composition of services. The algorithm is beneficial in reducing the costs and any negative impact that may happen if the irrelevant services were returned, and this surely demonstrates its effectiveness and efficiency. However in order not to miss any other available suitable services we have to improve the recall of our algorithm, in the incoming section we analyzed the reasons for that and we suggest enhancements to manage it.

4.4.2 Discussion

A semantic service discovery algorithm should have a high score of accuracy, such that it can retrieve as many semantically matched services as possible. However, we can summarize the reasons that lower the obtained accuracy in the algorithm in the following points:

- The shortcoming in the utilized WordNet semantic similarity tool. In some cases WordNet Ontology misses some concepts, especially compound phrases, for example in query number 1, four services were negatively missed because the concept in their output parameters was like comdeyfilm which does not exist in WordNet ontology. Actually, this makes the number of matched services by subsume or pluging relationships are little [12]. As a result, this lowers the overall accuracy of the algorithm. According to our experiments, some of these missed services may return to users if a lower degree of similarity threshold is used (i.e by utilizing similarity values obtained from the remaining input and output parameters)
- Another shortcoming is that some concepts may have semantic relationships, but this relationship will cause confusion in service discovery. For example, according to WordNet ontology, the concepts of car and missile have a high degree of similarity, which causes some car services to be returned to users when they were searching for missiles.

Several possible enhancements can be done in the future to overcome the mentioned points, such as using a hybrid of WordNet ontology and other ontologies that are specific to the domain of services, this could mitigate the effect of missing concepts in WordNet. Another issue is augmenting user queries with more detail about the required services such description part to prevent retrieving irrelevant services.

4.4.3 Threats to Validity

Several threats faced us during the implementation of the algorithm. These threats were treated carefully to preserve our work's validity. Among these, is the internal threat of selecting user queries for the evaluation process, to avoid the researcher bias in selecting them and to guarantee that are representative evaluation set, we selected them randomly from the set of services in all available domains. Another threat is finding the golden standard for the true number of available services, since it is calculated manually using human experts, we mitigate the effect of this threat by first: developing a tool that automatically extracts all possible services in order not to miss anyone by mistake, second, we (the authors) read the given description section of each service carefully in order to determine if it should be considered in the golden standard or not. However, regarding this point, we recommend enriching the user queries with more detail in order to prevent misunderstanding that could happen in selecting services.

5 Conclusion and Future Work

In this paper, we reviewed the work performed in the semantic Web service discovery field. Also, we implemented an algorithm that automated the discovery of semantic Web services. The algorithm considered both the syntactical and the semantic match with the user query. It computed the semantic similarity with user queries at the level of input and output parameters. In order to accomplish its task, it utilized two types of ontologies including the OWL-S ontology of the Web services and the WordNet ontology. We validated the algorithm by performing experiments on a service registry created from publicly published services and we gained good results at the precision level, we suggested several enhancements, such as combining different types of domain ontologies to improve its overall accuracy.

In future, we aim to extend the OWL-S ontology by augmenting the nonfunctional attributes such as response time and availability of Web services as this may achieve a better match with both system and user needs.

References

[1] Semantic web services test collection. URL https://github.com/kmi/sws-test-collections. accessed: 29.06.2023.

[2] Web services architecture. https://www.w3.org/TR/ws-arch/, . Accessed: June 28, 2023.

[3] Semantic markup for web services. https://www.w3.org/Submission/OWL-S/, . Accessed: June 29, 2023.

[4] Wordnet. https://wordnet.princeton.edu, .

[5] Ws4j. https://code.google.com/archive/p/ws4j/ws, .

[6] G. Alary, N. Hernandez, J.-P. Arcangeli, S. Trouilhet, and J.-M. Bruel. Comp-o: an owl-s extension for composite service description. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 171–182. Springer, 2020.

[7] M. Driss, S. Ben Atitallah, A. Albalawi, and W. Boulila. Req-wscomposer: a novel platform for requirements-driven composition of semantic web services. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–17, 2022.

[8] Z. Fayçal and T. Abdelkamel. Building a semantic web services ontology in the pharmaceutical field using the owl-s language. In *2021 International Conference on Information Systems and Advanced Technologies (ICISAT)*, pages 1–8. IEEE, 2021.

[9] J. Lu, J. Zheng, Z. Chen, Q. Wang, D. Li, and G. Xiao. A service composition evolution method that combines deep clustering and a service requirement context model. *Expert Systems with Applications*, 224:119920, 2023.

[10] R. Pahariya and L. Purohit. Recent advancements in semantic web service selection. *IETE Journal of Research*, pages 1–10, 2022.

[11] M. Rostami, M. R. Forghani, and R. Soorani. Effective approach based on concepts and concepts features parameters, for detecting semantic web services. *Journal of Advances in Computer Research*, 10(3):41–63, 2019.

[12] C. Surianarayanan, G. Ganapathy, and P. R. Chelliah. A novel approach for semantic microservices description and discovery toward smarter applications. In *Machine Intelligence and Smart Systems: Proceedings of MISS 2021*, pages 89–101. Springer, 2022.

[13] Z. Wu and M. Palmer. Verb semantics and lexical selection. *arXiv preprint cmp-lg/9406033*, 1994.