International Journal of Computational Intelligence and Informatics, Vol. 2: No. 1, April - June 2012

Quick and Efficient Approach for Semantic Service Discovery using Ontology based Indexing

Chellammal Surianarayanan and Gopinath Ganapathy
School of Computer Science and Engineering
Bharathidasan University
Tiruchirappalli, TN, 620 024, India
chelsrsd@rediffmail.com, gganapathy@gmail.com

***Abstract*-A two-step indexing based method which indexes services by their ontologies is proposed for efficient and quick semantic service discovery. In first step, for a given service request, a set of candidate services are chosen from the index by matching the ontologies of the request with the keys of index. In the second step, the request is semantically matched with candidate services to find a list of matched services ranked by their similarity score. The indexing of services helps in eliminating the irrelevant services of a request. Semantic matching will be performed only to candidate services rather than all available services. From experimentation, it is found that the proposed method quickens service discovery by an average elimination of irrelevancy of 92%. The time characteristics are analyzed using measures such as 'service loading time' and 'service matching time'. The average service loading time and service matching time of the proposed method are significantly reduced to 18.91 seconds 38.7 milli seconds when compared to sequential method which has average service loading time and service matching time of 577.7 seconds and 353.4 milli seconds respectively. When compared to the method which indexes services by outputs, the proposed work exhibits excellent recall and precision.**

Keywords- indexing based service discovery, indexing by ontologies, service discovery, semantic similarity.

I. INTRODUCTION

In Service Oriented Computing (SOC), complex business transactions are implemented using service composition where atomic services are discovered and combined in a specific pattern to achieve the given need. In real time applications, manual discovery is complex due to the existence of huge number of services in the Web. So, services should be automatically discovered and combined. Service description languages such as Web Service Description Language (WSDL) cannot be used for automatic service discovery as it is syntactic [1-2]. To automate service discovery, services are described using semantic languages such as [3-4]. Semantic description and discovery bring in maximum automation into semantic discovery with sufficient accuracy [5]. Regardless of its accuracy, semantic discovery consumes huge time due to reasoning process[6-8] which is performed to find semantic relations such as equivalent, subsumes, plug-in and fail that exist among various concepts of services. Ontology reasoner such as Pellet is used to find the semantic relations between concepts. In any discovery framework, there are two major components namely, service matcher and ontology reasoner. The matcher is a matching algorithm discovers matched services of a query with the help of reasoner. Further, any business transaction involves multiple services to be composed in complex chain. So, 'performance' and 'scalability' have become critical needs in service composition based applications. To meet the critical needs, optimization is introduced either in semantic reasoner or in service matcher or in hybrid. As the first one is dependent on the architecture of a reasoner, the second method is extensively used. Predominantly used techniques for optimization at matcher include clustering [9-16] caching [17] and indexing.

In this work, a two-step inverted indexing based approach which indexes services by their ontologies is proposed for efficient and quick semantic service discovery. In first step, for a given service request, a set of candidate services which have ontologies same as that of request are chosen from a service index in which services are indexed by their ontologies . Here, services which have ontologies different from that of request are considered as irrelevant to the request and such services are eliminated. In the second step, the service request is semantically matched with candidate services to obtain matched services of the request. The matched services are ranked by their similarity score.

II. MOTIVATION

In the existing indexing based service discovery approaches [18-20], the output parameters of services are used as keys of index and each key is linked to a list of services that contain that key. This is shown in Fig 1. For

example, the key, 'http://127.0.0.1/ontology/concept.owl#_price' is linked to a list of three services namely, s_2, s_5 and s_8. These three services contain 'http://127.0.0.1/ontology/concept.owl#_price' as one of their outputs.

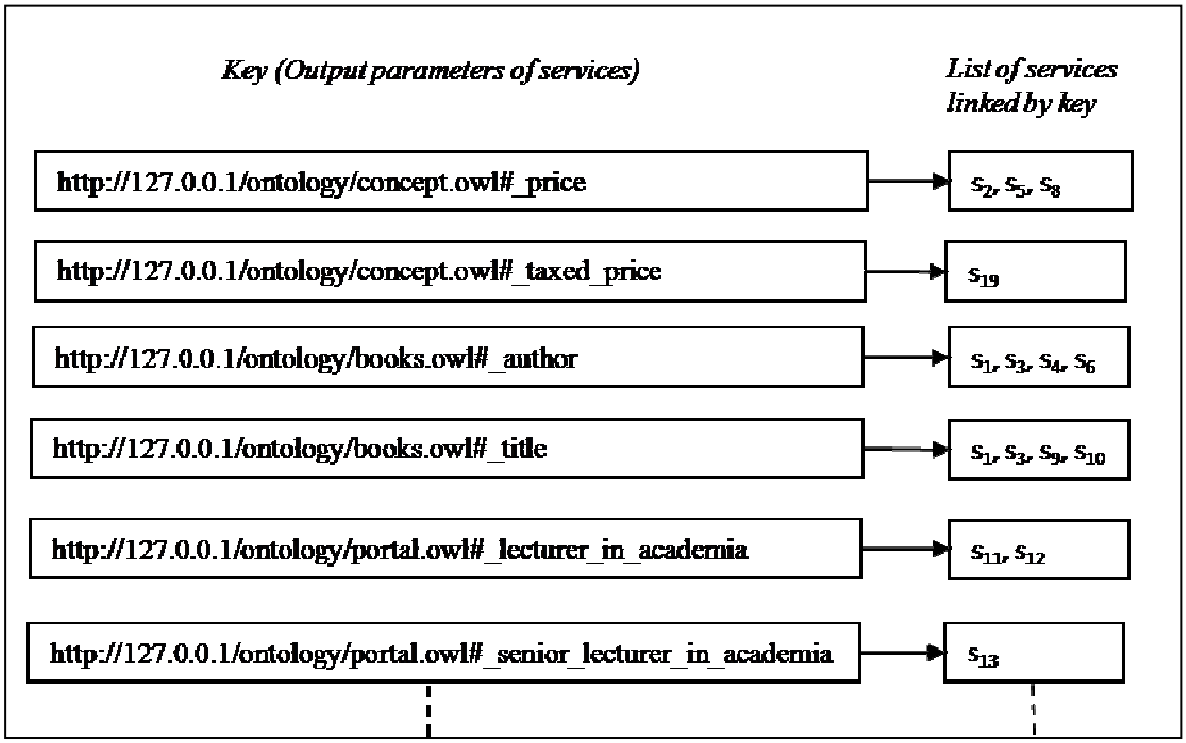

Figure 1. Indexing of services by their output parameters.

When a request (or query) is submitted, for every output parameter of the request if it exists as a key of index, the method retrieves the services linked by that key. Here, the method performs a key-word based matching to check any of output parameter of the request exists as key of the index. As the method uses key-word based (syntactic) method for matching, it cannot detect matched services for a request if the matched service contains subclass or super class of a key as its output parameter(rather than the output parameter itself). The following example illustrates how the method fails to detect potential matches when the request contains super class or sub class of an output parameter instead of the parameter itself. Consider an advertised service, *A* which has an output parameter, 'http://127.0.0.1/ontology/concept.owl#_price'. Let a service request, *R* search for a service which offers an output parameter, 'http://127.0.0.1/ontology/concept.owl#_amount', defined as the super class of 'http://127.0.0.1/ontology/concept.owl#_price'. Now, as the matching method performs a key-word based matching, it fails to detect *A* as a matched service for *R*. Though the index contains 'http://127.0.0.1/ontology/concept.owl#_price' which is a sub class of 'http://127.0.0.1/ontology/concept.owl #_amount', the matching method cannot detect the service linked by 'http://127.0.0.1/ontology/concept.owl#_price' as the method does not consider the semantic relations between the concepts of services. With same argument, it can be understood that when a service request contains super class of a key, the method will fail to detect matched services. Hence, in the proposed work, all the available advertised services are indexed by their ontologies. Ontologies of services are used as keys as in Fig. 2.

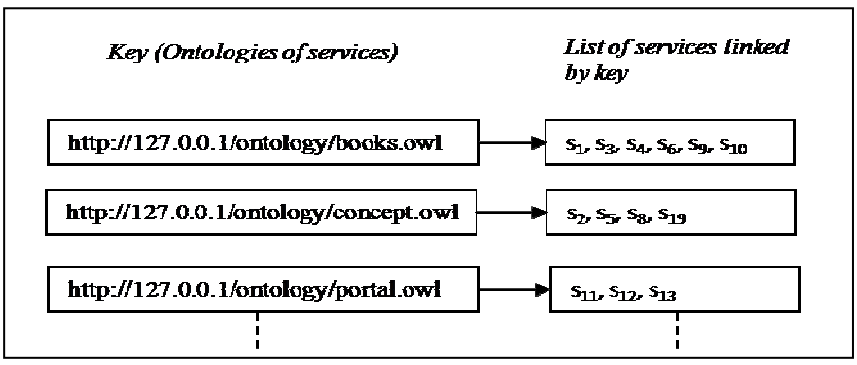

Figure 2. Indexing of services by their ontologies.

For any service request, the method compared the ontologies of service request with the keys of the index for any match. If the method finds a matched key, then it retrieves all the services linked by the key. Consider a request *R* which searches for a service offer with an output, 'http://127.0.0.1/ontology/concept.owl#_amount', defined as a super class of 'http://127.0.0.1/ontology/concept.owl#_price'. The matching method checks whether the ontology 'http://127.0.0.1/ontology/concept.owl' does exist as key of the index using key-word based matching. From Fig.2 it is found that for the request *R*, the proposed method of indexing retrieves s_2, s_5, s_8, and s_{19} as a list of services linked by the key. This list of services is referred to as candidate services and only these services are chosen for employing semantic matching of Input-Output (IO) capabilities with the request.

III. Related Work

Indexing based service discovery has been handled by approaches such as [18-20]. These methods use inverted indexing to rapidly remove irrelevant services and locate the suitable services quickly. But the above methods use output parameters of services as 'key' of the index. When a service request is submitted, the matching method can rapidly locate the services with required outputs from the index by comparing the outputs of request with key of index. This method retrieves the services having outputs which are exactly same as the outputs of the request. The existing methods fail to detect services when services contain 'sub-type' or 'super type' of the required outputs instead of the outputs itself.

An alternate indexing approach is proposed for quick and efficient service discovery which differs from existing approaches in three aspects. Firstly, it indexes the services by their ontologies and not by their output parameters. This indexing helps in retrieving a set of candidate services which have the same ontology as that of service request, ignoring all other advertised services as irrelevant. Secondly, the proposed method handles various patterns of service request which may be simply a set of inputs or a set of outputs or combinations of both. Thirdly, for a given service request, the proposed method produces a list of matched services ranked by their semantic similarity score.

IV. Proposed Approach

A two-step indexing based approach which indexes the services by their ontologies is proposed for quick and efficient service discovery. The method includes two steps, namely 'finding candidate services' and 'finding matched services'. In the first step, a set of candidate services having ontologies same as that of the given query is identified. In the second step, each candidate service is matched with the query semantically to find a list of matched services. The rest of this section describes the definitions used in the proposed approach and its two steps elaborately.

A. Definitions

It is assumed that web services are described using Web Ontology Languages for Services (OWL-S), published and available as *advertised services*. An advertised service is denoted by A. Let A contain k input parameters denoted by $(i_j^a \,|\, 1 \le j \le k)$, l output parameters denoted by $(o_j^a \,\|\, 1 \le j \le l)$. Let a_{ont} refer to all ontologies of A. A *service request* is denoted by *R*. Let *R* contain n input parameters denoted by $(i_i^r \,|\, 1 \le i \le n)$ and m output parameters denoted by $(o_i^r \,\|\, 1 \le i \le m)$. Let r_{ont} refer to all ontologies of *R*. A *candidate service, C* represents an advertised service which has same ontologies as that of a given request and it is a candidate chosen for matching service capabilities semantically with the request. Let *C* contain u input parameters denoted by $(i_i^c \,|\, 1 \le i \le u)$ and v output parameters denoted by $(o_i^c \,|\, 1 \le i \le v)$. Let c_{ont} refer to all ontologies of *C* and $c_{ont} = r_{out}$ for a given *R*.

B. Finding Candidate Services

In any semantic service discovery mechanism, one of the basic criteria to be fulfilled is that the ontologies used in a service request should match with the ontologies used in an advertised service. This means that if a service request expresses its inputs and outputs using ontology, say "http://www.amazon.com/ontology/books.owl", then semantic matching can be performed to such advertised services which also express their semantics using the same ontology, "http://www.amazon.com/ontology/books.owl". Hence, while matching, the first check to be done is to find whether the ontologies in a service request and an advertised service are same. If they are not same, then the advertised service becomes an irrelevant service to the service request. The services which satisfy the ontology check are candidate services.

C. Finding Matched Services

In this step, the given service request is matched with each candidate service to find whether a candidate service is similar to the request. The similarity between a request, *R* and a candidate service *C* denoted by $Sim(R,C)$ is computed as the sum of normalized input and output similarities between *R* and *C*. Let $InSim(R,C)$

denote the normalized input similarity between *R* and *C*. Let $OutSim(R,C)$ denote the normalized output similarity between *R* and *C*. Now, the value of $Sim(R,C)$ is computed using (1).

$$Sim(R,C) = InSim(R,C) + OutSim(R,C) \quad (1)$$

To illustrate the computation of $OutSim(R,C)$, consider *R* with two output parameters and *C* with three output parameters. The output parameters of *R* are denoted by o_1^r and o_2^r. The output parameters of *C* are denoted by o_1^c, o_2^c and o_3^c. While computing output similarity, each output parameter of *R* is matched with every output parameter of *C*. For example, the possible pairs of matches for o_1^r are (o_1^r, o_1^c), (o_1^r, o_2^c) and (o_1^r, o_3^c). Let $DoM(o_1^r, o_1^c)$, $DoM(o_1^r, o_2^c)$ and $DoM(o_1^r, o_3^c)$ denote the *DoM* of o_1^r with various parameters of o_1^c. The degree of match of o_1^r with all output parameters of *C* is given as $Max\{DoM(o_1^r, o_1^c), DoM(o_1^r, o_2^c), DoM(o_1^r, o_3^c)\}$. Similarly, the possible pairs of matches for o_2^r are (o_2^r, o_1^c), (o_2^r, o_2^c) and (o_2^r, o_3^c) and the degree of match of o_2^r with all the output parameters of *C* is given as $Max\{DoM(o_2^r, o_1^c), DoM(o_2^r, o_2^c), DoM(o_2^r, o_3^c)\}$. The normalized output similarity between *R* and *C* is obtained by adding the individual maximum values and dividing the sum by the number of output parameters of *R*. That is, the value of $OutSim(R,C)$ is computed using (2).

$$OutSim(R,C) = \frac{1}{2} \times (Max\{DoM(o_1^r, o_1^c), DoM(o_1^r, o_2^c), DoM(o_1^r, o_3^c)\} + Max\{DoM(o_2^r, o_1^c), DoM(o_2^r, o_2^c), DoM(o_2^r, o_3^c)\}) \quad (2)$$

In (2), $DoM(o_i^r, o_j^c)$ denotes the Degree of Match between i^{th} output parameter of *R* and j^{th} output parameter of *C*. The value of $DoM(o_i^r, o_j^c)$ is computed based on the semantic relations between the parameters. Various semantic relations and *DoM* s are defined below. Based on the *DoM*, a score value to represent the level of similarity is assigned.

Exact: If the type of o_i^r is equivalent to the type of o_j^c, then the match is exact and a score of 1 is assigned.

Plug-in: If the type of o_i^r is the subtype of o_j^c, then the match is plug-in and a score of 0.75 is assigned.

Subsumes: If the type of o_j^c is a subtype of o_i^r, then the match is subsumes and a score of 0.5 is assigned.

Fail: if the type of o_i^r is different from that of o_j^c, then the match is fail and a score of 0.0 is assigned.

Further, the computation of $InSim(R,C)$ is similar to the computation of $OutSim(R,C)$. After computing the similarity score between *R* and each *C*, a list of candidate services for which the similarity with *R* is greater than a minimum value (decided by application) are returned as matched services. Further, the matched services are returned as a sorted list ranked by their similarity score.

V. EXPERIMENTATION

The proposed approach is implemented with the experimental setup shown in Fig. 3.

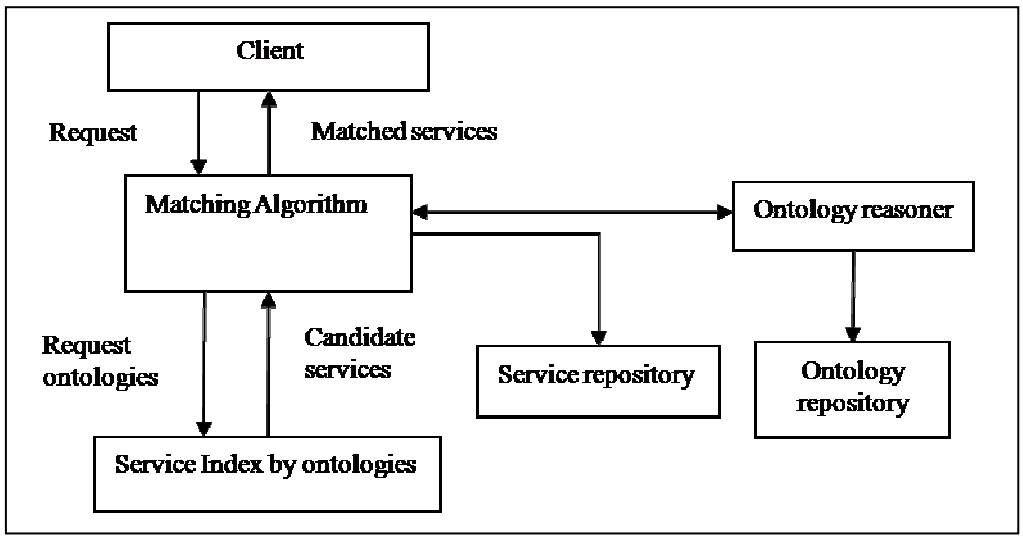

Figure 3. Experimental Setup.

There are two objectives of experimentation. The first objective is to find computation time of the proposed approach and to compare it with existing approaches. The second objective is to find the accuracy of the results obtained using the proposed approach and to compare it with that of existing approaches. To carry out the experiments, typical test data of 100 OWL-S web services has been constructed using services from publicly available OWL-TC version-3 test collection. The services and their ontologies are deployed in Xampp server. An index of services is created with ontologies of services as keys in Excel. Ten service requests covering the various patterns of service request are chosen as 'test queries'. The test queries and their ontologies are given in Table I.

TABLE I. TEST QURIES AND THEIR ONTOLOGIES

#	**Test Query**	**Ontologies used by the test queries**	**Description**
Q1	o_m^r =(#_Author)	http://127.0.0.1/ontology/books.owl	Query with one output
Q2	o_m^r =(#_Comedyfilm, #_Actionfilm)	http://127.0.0.1/ontology/my_ontology.owl	Query with two outputs
Q3	o_m^r =(#_Report, #_Coffee, #_Tea)	http://127.0.0.1/ontology/Mid-level-ontology.owl	Query with three outputs
Q4	i_n^r =(#_Title)	http://127.0.0.1/ontology/books.owl	Query with one input
Q5	i_n^r =(#_Geopolitical-entity, #_Geopolitical-entity1)	http://127.0.0.1/ontology/portal.owl	Query with two inputs
Q6	o_m^r =(#_Amount) i_n^r =(#_ThreeWheeledCar)	http://127.0.0.1/ontology/concept.owl http://127.0.0.1/ontology/my_ontology.owl	Query with one output and one input
Q7	o_m^r =#_Funding i_n^r =(#_Weapon, #_Missile, #_Government)	http://127.0.0.1/ontology/SUMO.owl http://127.0.0.1/ontology/Mid-level-ontology.owl	Query with three inputs and one output
Q8	o_m^r =(#_SkilledOccupation, #_TimeDuration) i_n^r =(#_Country)	http://127.0.0.1/ontology/Mid-level-ontology.owl http://127.0.0.1/ontology/SUMO.owl http://127.0.0.1/ontology/portal.owl	Query with two outputs and one input
Q9	o_m^r =(#_Price, #_TaxedPrice, #_StockMarket) i_n^r =(#_PreparedFood)	http://127.0.0.1/ontology/concept.owl http://127.0.0.1/ontology/Mid-level-ontology.owl	Query with three outputs and one input
Q10	o_m^r =(#_Sandwich) i_n^r =(#_GroceryStore, #_Quantity)	http://127.0.0.1/ontology/Mid-level-ontology.owl http://127.0.0.1/ontology/support.owl	Query with one output and two inputs

The computation time of the proposed approach is analyzed using three evaluation measures namely, number of services to be loaded, service loading time and service matching time. Experiments are performed on a Laptop with Intel Pentium(R) Dual-Core, 2.20GHz CPU, 3.0 GB memory and Windows 7 Ultimate Operating System. The implementation is performed using J2EE environment.

Three experiments are conducted using three different methods, namely sequential method (without indexing), proposed approach (which uses indexing of service by their ontologies) and method which uses indexing of services by their outputs. The first experiment is conducted using sequential approach. Let N_s denote the number of services to be loaded in sequential method, t_s^l denote time required to load the services and their ontologies into the reasoner (known as *service loading time*) in sequential mode, t_s^m denote the time required to match the service request with the available services (known as *service matching time*) and t_s^u denote time required to unload the services from reasoner(known as *service unloading time*). The second experiment is conducted using the proposed approach. Let N_p denote the number of services to be loaded in the proposed approach, t_p^l denote the service loading time of the proposed approach, t_p^m denote the service matching time of the proposed approach and t_p^u denote the service unloading time of the proposed approach. The third experiment is conducted using a standard indexing approach which indexes services by their output parameters. For simplicity, the third method is referred to as *output based indexing method*. Let N_o denote the number of services to be loaded in the output based indexing method. Let t_o^l denote the service loading time of the output based indexing method. Let t_o^m denote the service matching time of the output based indexing method and t_o^u denote

the service unloading time of the output based indexing method. The values of N_s, t_s^l, t_s^m, t_s^u, N_p, t_p^l, t_p^m, t_p^u, N_o, t_o^l, t_o^m and t_o^u are computed for different test queries and given in Table II.

TABLE II. TIME CHARACTERISTICS OF SEQUENTIAL, PROPOSED AND OUTPUT BASED INDEXING METHODS

#	N_s	N_p	N_o	t_s^l (sec)	t_p^l (sec)	t_o^l (sec)	t_s^m (milli sec)	t_p^m (milli sec)	t_o^m (milli sec)	t_s^u (milli sec)	t_p^u (milli sec)	t_o^u (milli sec)
Q1	100	8	2	547	5.3	1.4	223	32	31	16	0	0
Q2	100	5	2	583	5.0	2.9	312	47	30	0	0	0
Q3	100	7	5	589	27.5	22.5	390	48	32	16	0	0
Q4	100	8	0	529	6.9	0	265	32	0	0	0	0
Q5	100	10	0	541	15.9	0	285	46	0	0	0	0
Q6	100	12	0	622	20	0	390	32	0	15	0	0
Q7	100	1	1	599	4.1	4.1	484	16	16	16	0	0
Q8	100	1	1	590	6.4	6.4	374	16	16	16	0	0
Q9	100	24	18	593	86	70	468	84	79	16	0	0
Q10	100	4	0	584	12	0	343	32	0	0	0	0

From Table II, it is found that in sequential method, to find matched services of any service request, all the available advertised services are loaded and matching process has to be carried out for all the services. Hence in experiment-1, all the hundred services of the test collection are loaded. From experiment-2, it is found that ontology based indexing achieves a reduction in number of services to be loaded by 92% and from experiment-3, it is observed that the output based indexing method achieves a reduction in number of services to be loaded by 97.1%. In experiment-2, the services which use the ontologies of the given request will be loaded. The number of services loaded in experiment-2, i.e. N_p is always equal to or greater than N_o. Hence, the output based indexing method achieves greater reduction in number of services to be loaded for a request.

Regarding service loading time, sequential method needs an average service loading time of 577.7 seconds whereas the proposed method needs an average service loading time of 18.91 seconds and output based indexing method takes an average service loading time of 10.73 seconds. As all test services are loaded for matching, sequential method takes the highest time for loading when compared to the proposed method and output based indexing method. Further, as N_p is always equal to or greater than N_o, the service loading time of the proposed method is higher than that of output based indexing method.

It is found that the average matching time of sequential method, proposed method and output based indexing method is 353.4 milli seconds, 38.7 milli seconds and 20.4 milli seconds respectively. From the above experiments, it is very clear that loading services into reasoner consumes huge time. The time taken by the sequential method is more (few hundred of milli seconds) when compared to other two methods (few tens of milli seconds). Further, though the service loading time is slightly more for the proposed method when compared matching time, the service matching time of the proposed method and output based indexing method are of the order of few tens of milli seconds. It is also observed that the service unloading time is negligible.

Another analysis has been carried out using two evaluation measures recall and precision to find how accurate the proposed method is in finding the matched services for test queries. For analysis purpose, an advertised service is considered as relevant or similar to a service request if its normalized similarity score with the request is equal to or more than 0.5. This is chosen because any dynamic web service composition process involves a huge collection of services and it demands only highly compatible matches for a given service request. For each test query, the values of various parameters, namely, Actual Number of Relevant services present in the test data (*ANR*), number of services retrieved using sequential method (NR_s), number of relevant services retrieved using sequential method (NRR_s), number of services retrieved using proposed method (NR_p), number of relevant services retrieved using proposed method (NRR_p), number of services retrieved using output based indexing method (NR_o), number of relevant services retrieved using output based indexing method (NRR_o), recall of sequential method (R_s), recall of proposed method (R_p), recall of output based indexing method (R_o), precision of sequential method (P_s), precision of proposed method (P_p) and precision of output based indexing method (P_o) are calculated and given in Table III.

TABLE III. RECALL AND PRECISION VALUES OF SEQUENTIAL, PROPOSED AND OUTPUT BASED INDEXING METHODS

#	*ANR*	NR_s	NRR_s	NR_p	NRR_p	NR_o	NRR_o	R_s (%)	R_p (%)	R_o(%)	P_s (%)	P_p (%)	P_o (%)
Q1	2	2	2	2	2	2	2	100	100	100	100	100	100
Q2	4	4	4	4	4	4	4	100	100	100	100	100	100
Q3	1	4	1	4	1	1	1	100	100	100	25	25	25
Q4	3	3	3	3	3	0	0	100	100	0	100	100	0
Q5	1	1	1	1	1	0	0	100	100	0	100	100	0
Q6	4	8	4	8	4	0	0	100	100	0	50	50	0
Q7	1	1	1	1	1	1	1	100	100	100	100	100	100
Q8	1	1	1	1	1	1	1	100	100	100	100	100	100
Q9	2	6	2	6	2	2	2	100	100	100	33	33	33
Q10	2	2	2	2	2	0	0	100	100	0	100	100	0

From Table III, it is found that the average recall of sequential method, proposed method and output based indexing method are 100%, 100% and 60 % respectively. The average precision of sequential method, proposed method and output based indexing method are found to be 80.8%, 80.8% and 45.8 % respectively.

The recall and precision graphs of the three methods are given in Fig. 4 and Fig. 5 respectively.

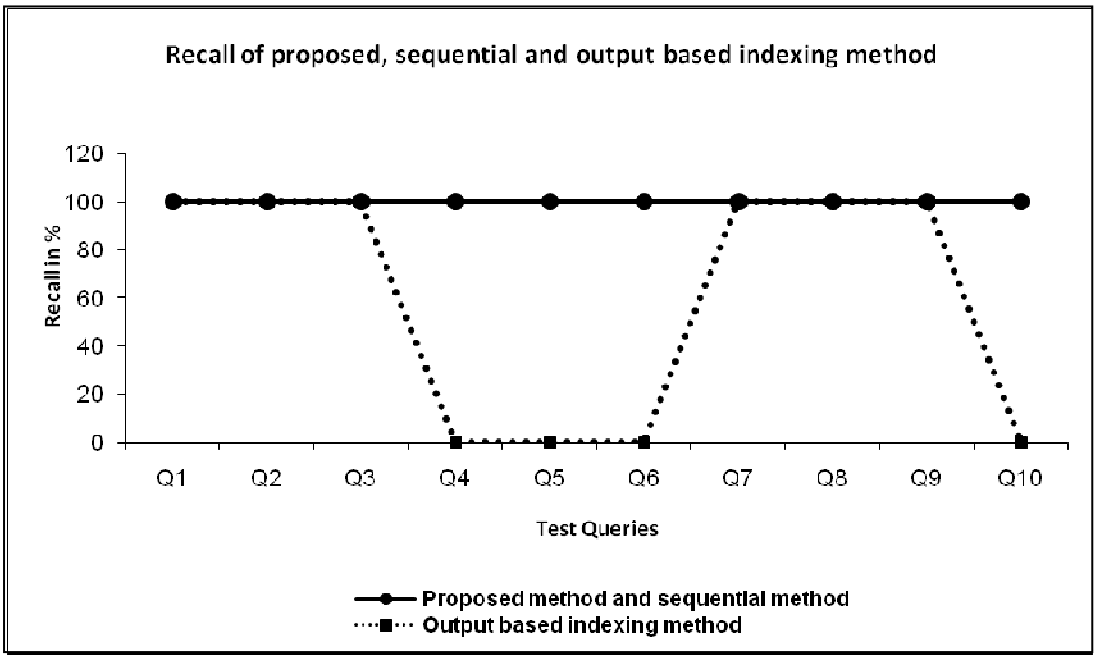

Figure 4. Recall of proposed, sequential and output based indexing methods.

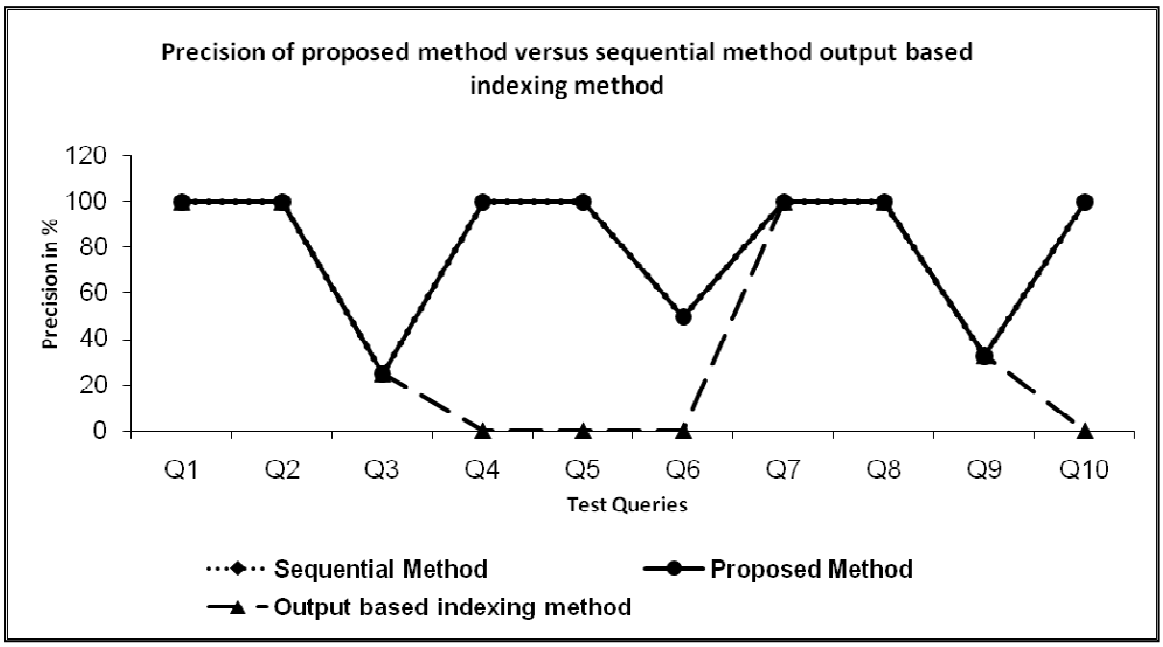

Figure 5. Precision of proposed, sequential and output based indexing methods.

From Table III, it is found that the output based indexing method has precision and recall as 0 for queries Q4, Q5, Q6 and Q10. When it is analyzed with service description files, the queries Q4 and Q5 did not contain any output parameters and the output based indexing method is failed to detect Q4 and Q5. Further, the method failed to detect Q6 and Q10 because those queries did not contain the outputs but the subtypes of outputs. But all the above queries are handled efficiently by the proposed method. The average precision of sequential method and the proposed method is found to be the same (80.8%). The reduction is precision is due to the reason that the proposed similarity computation considers plug-in matches also into account.

VI. CONCLUSION

An inverted indexing based approach is proposed as an optimization solution to semantic service discovery. The method is unique in using ontologies of services as keys of index. The method has been implemented and evaluated for its performance using different measures. By experimenting with a typical test data, it is found that the method outperforms both sequential matching method and output based indexing method. It achieves a significant reduction in number of services to be loaded which leads to an appreciable reduction in service loading time and service matching time. Though the proposed approach achieves greater time reduction using indexing when compared with sequential method, still it performs semantic matching of functional capabilities to candidate services during querying. It is planned to avoid the task of performing semantic matching to candidate services (during querying) by using an enhanced method of indexing. In the enhanced indexing, two indices, one with output parameters of services as keys and other with input parameters of services as keys are created and maintained. Also, key should be linked to all services having parameters which are semantically related to the key through equivalent, subsumes, plug-in matches.

REFERENCES

[1] Kaarthik Sivashanmugam, John A Miller, Amit P. Sheth, Kunnal Verma, "Framework for Semantic Web Process Composition", International Journal of Electronic Commerce, M.E. Sharpe Inc publishers, 2005, Vol 9. Issue 2 (Number 2/Winter 2004-5), pp. 71-106.

[2] Tanveer Syeda-Mahmood, Gauri Shah, Rama Akkiraju, Anca-Andrea Ivan, Richard Goodwin, "Searching Service Repositories by Combining Semantic and Ontological Matching", IEEE International Conference on Web Services (ICWS'05), Orlando, Florida, July 2005, pp. 13-20.

[3] Jacek Kopecky, Tomas Vitvar, Carine Bournez, Joel Farrell, "SAWSDL: Semantic Annotations for WSDL and XML Schema", IEEE Internet Computing, IEEE Computer Society, 2007, Vol. 11, No. 6, pp. 60-67.

[4] Ruiqiang Guo, Jiajin Le, XiaoLing Xia, "Capability matching of Web services based on OWL-S", Proceedings of 16th International Workshop on Database and Expert Systems Applications, 22-26 August 2005, pp. 653-657.

[5] Katia Sycara, Massimo Paolucci, Anupriya Ankolekar, Naveen Srinivasan, "Automated discovery, interaction and composition of Semantic Web Services", Journal of Web Semantics, Elsevier, December 2003, Vol. 1, No.1, pp. 27-46.

[6] Sonia Ben Mokhtar, Anupam Kaul Nikolaos Georgantas and Valerie Issarny, "Towards Efficient Matching of Semantic Web Service Capabilities", International Workshop on Web Services Modeling and Testing (WS-MaTe 2006).

[7] Sonia Ben Mokhtar, Anupam Kaul Nikolaos Georgantas and Valerie Issarny, "Efficient Semantic Service Discovery in Pervasive Computing Environments", M. Van Steen and M. Henning (Eds): Middleware 2006, LNCS Vol. 4290, pp. 240-259.

[8] Sonia Ben Mokhtar, Davy Preuveneers, Nikolaos Georgantas, Valerie Issarny, Yolande Berbers, "EASY", Efficient semantic Service discovery in pervasive computing environments with QoS and context support", The Journal of Systems and Software, 2008, Vol. 81, pp. 785-808.

[9] Dong Shou, Chi-Hung Chi, "A Clustering-based Approach for Assisting Semantic Web Service Retrieval", IEEE International Conference on Web Services, 2008, pp. 838-839.

[10] Dong Shou, Chi-Hung Chi, "Effective Web Service Retrieval Based on Clustering", Fourth International Conference on Semantics, Knowledge and Grid, IEEE Computer Society, 2008, pp. 469-472.

[11] Huiying Goa, Wolffried Stucky, Lei Liu, "Web Services Classification Based on Intelligent Clustering Techniques", International Forum on Information Technology and Applications, 2009, Vol. 3, pp. 242-245.

[12] Jingyu Zhang, Xueli Yu, Peng Liu and Zhen Wang, "Research on improving Performance of Semantic Search in UDDI", Proceedings of the WRI Global Congress on Intelligent Systems, IEEE Computer Society, 2009, Vol. 4, pp. 572-576.

[13] Li Ying, "Algorithm for Semantic Web Services Clustering and Discovery", International Conference on Communications and Mobile Computing, IEEE Computer Society pp. 532-536, 2010.

[14] Peng Liu, Jingyu Zhang, Xueli Yu, "Clustering-Based Web Service Matchmaking with Automated Knowledge Acquisition", W. Liu et. al (Eds), Springer-Verlag Berlin Heidelberg, LNCS, Vol, 5854, pp. 261-270, 2009.

[15] Richi Nayak, Bryan Lee, "Web Service Discovery with additional Semantics and Clustering" Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, IEEE Computer Society, 2007 pp. 555-558.

[16] Shun Hon, Haiyang Wang, Lizhen Cui, "A User Experience-Oriented Service Discovery Method with Clustering Technology", Second International Symposium on Computational Intelligence and Design, IEEE Computer Society, 2009, Vol. 2, pp. 64-67.

[17] Michael Stollberg, Martin Hepp and Jorg Hoffmann, "A Caching Mechanism for Semantic Web Service Discovery", Springer-Verlag Berlin Heidelberg, LNCS, 2007, Vol. 4825, pp. 480-493.

[18] Gao Ting, Wang Hiayang, Zheng Naihui, Li Fei, "An Improved Way to Facilitate Composition Oriented Semantic Service Discovery", International Conference On Computer Engineering and Technology, 2009, pp. 156-160.

[19] Bo Zhou, Tinglei Huang, Jie Liu, MeizhouShen, "Using Inverted Indexing to Semantic WEB Service Discovery Search Model, Proceedings of 5th International Conference on Wireless communications, networking and mobile computing, IEEE Press, 2009, pp. 4872-4875.

[20] Li Kuang, Ying Li, Jian Wu, Shuiguang Deng, Zhaohui Wu, "Inverted Indexing for Composition-Oriented Service Discovery", IEEE International Conference on Web Services, 2007, pp. 257-264.

Chellammal Surianarayanan obtained her M.Tech in 2008 and presently working as Assistant Professor with Department of Computer Science, Bharathidasan University Constituent College, Orathanadu. She has 10 years of Research & Development experience in Indira Gandhi Centre for Atomic Research (IGCAR), Kalpakkam. Also, she is pursuing Ph.D in computer science in Bharathidasan University, Tiruchrappalli, India. Her research interests include semantic web services and data mining.

Prof. Gopinath Ganapathy obtained his Masters in 1988 and Ph.D in 1996. Presently he is the Director of Technology Park and Head of School of Computer Science and Engineering, Bharathidasan University, Tiruchirappalli, India. He published around 30 research papers in International Journals and Conferences. He has 23 years of academic, industry, research and consultancy services. He has around 8.5 years of International experience in U.S and U.K. He is a profession member of IEEE and ACM. His research interests include semantic web, auto programming, Natural Language Processing and text mining.