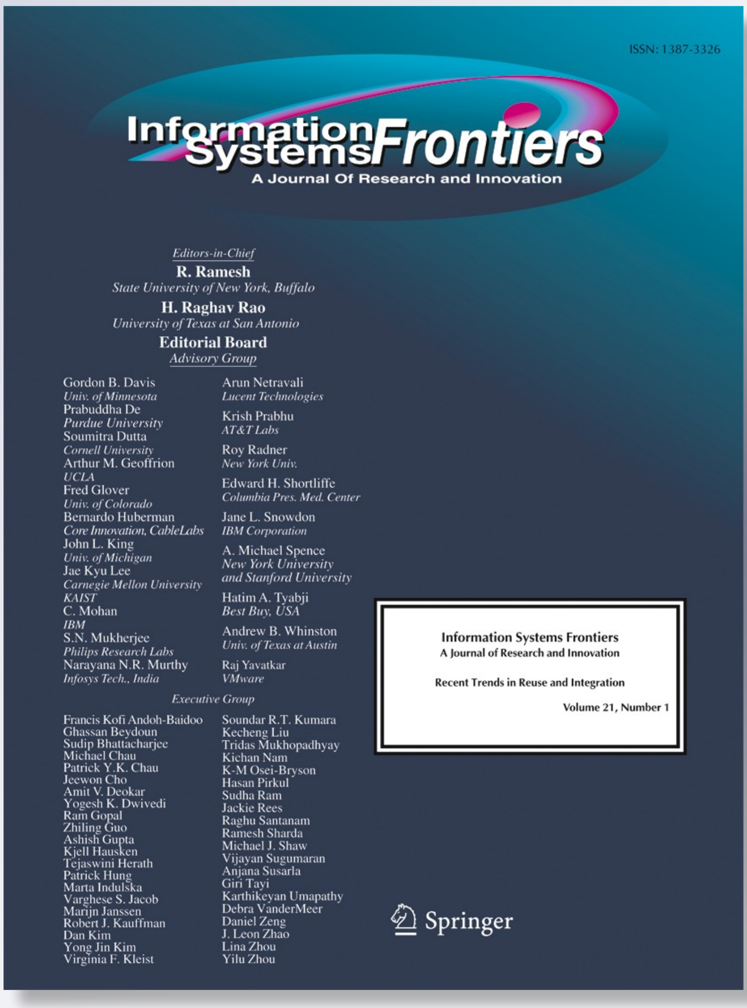
Discovering composable web services using functional semantics and service dependencies based on natural language requests

Sowmya Kamath S & Ananthanarayana V. S.

Information Systems Frontiers
A Journal of Research and Innovation

ISSN 1387-3326
Volume 21
Number 1

Inf Syst Front (2019) 21:175-189
DOI 10.1007/s10796-017-9738-2



ISSN: 1387-3326

Information Systems Frontiers
A Journal Of Research and Innovation

Editors-in-Chief
R. Ramesh
State University of New York, Buffalo
H. Raghav Rao
University of Texas at San Antonio
Editorial Board
Advisory Group

Gordon B. Davis *Univ. of Minnesota*	Arun Netravali *Lucent Technologies*
Prabuddha De *Purdue University*	Krish Prabhu *AT&T Labs*
Soumitra Dutta *Cornell University*	Roy Radner *New York Univ.*
Arthur M. Geoffrion *UCLA*	Edward H. Shortliffe *Columbia Pres. Med. Center*
Fred Glover *Univ. of Colorado*	Jane L. Snowdon *IBM Corporation*
Bernardo Huberman *Core Innovation, CableLabs*	A. Michael Spence *New York University and Stanford University*
John L. King *Univ. of Michigan*	Hatim A. Tyabji *Best Buy, USA*
Jae Kyu Lee *Carnegie Mellon University KAIST*	Andrew B. Whinston *Univ. of Texas at Austin*
C. Mohan *IBM*	Raj Yavatkar *VMware*
S.N. Mukherjee *Philips Research Labs*	
Narayana N.R. Murthy *Infosys Tech., India*	

Executive Group

Francis Kofi Andoh-Baidoo	Soundar R.T. Kumara
Ghassan Beydoun	Kecheng Liu
Sudip Bhattacharjee	Tridas Mukhopadhyay
Michael Chau	Kichan Nam
Patrick Y.K. Chau	K-M Osei-Bryson
Jeewon Cho	Hasan Pirkul
Amit V. Deokar	Sudha Ram
Yogesh K. Dwivedi	Jackie Rees
Ram Gopal	Raghu Santanam
Zhiling Guo	Ramesh Sharda
Ashish Gupta	Michael J. Shaw
Kjell Hausken	Vijayan Sugumaran
Tejaswini Herath	Anjana Susarla
Patrick Hung	Giri Tayi
Marta Indulska	Karthikeyan Umapathy
Varghese S. Jacob	Debra VanderMeer
Marijn Janssen	Daniel Zeng
Robert J. Kauffman	J. Leon Zhao
Dan Kim	Lina Zhou
Yong Jin Kim	Yilu Zhou
Virginia F. Kleist	

Information Systems Frontiers
A Journal of Research and Innovation

Recent Trends in Reuse and Integration

Volume 21, Number 1

Springer

Springer

Your article is protected by copyright and all rights are held exclusively by Springer Science +Business Media New York. This e-offprint is for personal use only and shall not be self-archived in electronic repositories. If you wish to self-archive your article, please use the accepted manuscript version for posting on your own website. You may further deposit the accepted manuscript version in any repository, provided it is only made publicly available 12 months after official publication or later and provided acknowledgement is given to the original source of publication and a link is inserted to the published article on Springer's website. The link must be accompanied by the following text: "The final publication is available at link.springer.com".

Springer

Author's personal copy

Inf Syst Front (2019) 21:175–189
DOI 10.1007/s10796-017-9738-2

Discovering composable web services using functional semantics and service dependencies based on natural language requests

Sowmya Kamath S[1] · **Ananthanarayana V. S.**[1]

Published online: 10 February 2017
© Springer Science+Business Media New York 2017

Abstract The processes of service discovery, selection and composition are crucial tasks in web service based application development. Most web service-driven applications are complex and are composed of more than one service, so, it becomes important for application designers to identify the best service to perform the next task in the intended application's workflow. In this paper, a framework for discovering composable service sets as per user's complex requirements is proposed. The proposed approach uses natural language processing and semantics based techniques to extract the functional semantics of the service dataset and also to understand user context. In case of simple queries, basic services may be enough to satisfy the user request, however, in case of complex queries, several basic services may have to be identified to serve all the requirements, in the correct sequence. For this, the service dependencies of all the services are used for constructing a service interface graph for finding suitable composable services. Experiments showed that the proposed approach was effective towards finding relevant services for simple & complex queries and achieved an average accuracy rate of 75.09 % in finding correct composable service templates.

Keywords Web service discovery · Service composition · Natural language processing · Semantics

✉ Sowmya Kamath S
sowmyakamath@nitk.ac.in

Ananthanarayana V. S.
anvs@nitk.ac.in

1 Department of Information Technology, National Institute of Technology Karnataka, Surathkal, Mangalore, 575025, India

1 Introduction

Service orientation has played a very important role in the development and deployment of distributed systems over the Web. Web service technologies have fueled the development of efficient web-based applications by facilitating the integration and interoperability of heterogeneous systems owned by different organizations. As a result, monolithic business application development has now yielded way to loosely coupled, reusable large-scale enterprise system development methodologies, based on the web service paradigm (Truong and Dustdar 2009). This has further promoted novel application development and reuse through the use of existing services in various scenarios as per business needs, thus minimizing cost and effort.

Given the tremendous potential that full service orientation brings to businesses, there has been a proliferation of published services on the Web. Due to this, there is a need for automated approaches for improving tasks like service discovery and selection for supporting novel application development. Web Service discovery and selection often becomes a bottleneck and considerable research effort is currently concentrated on the problem of adding intelligence and machine understanding to web service capabilities to support automated on-the-fly discovery, matchmaking and composition (Klusch et al. 2006).

Most existing works in the area of web service discovery deal with the problem of identifying relevant basic services for a given user requirement. Researchers have used various approaches – keyword based search (Song et al. 2007; Wu and Chang 2008), service broker based approaches (Almasri and Mahmoud 2009; D'Mello et al. 2008), service tagging (Fang et al. 2012; Li et al. 2014; Lin and Cheung 2014), service classification (Yang and Zhou 2014; Varguez-Moo et al. 2013) , semantics based search (Chan et al. 2011; Wu

Author's personal copy

et al. 2008), domain ontology based matching (Benatallah et al. 2005; Nayak and Lee 2007) etc, and have reported good results. However, the fact remains that most web service based applications are actually workflows that depend on multiple, loosely coupled web services that are composed in a predetermined sequence to provide the intended functionality. For example, the *ordering task* on an e-commerce site is actually a complex business process that requires multiple tasks to be performed by different web services in a well designed workflow (*login* service to be invoked to ensure that user is logged in before ordering, *inventory* service to run in the background to check if item is actually available before it can be added to the shopping cart etc.), to ensure utmost consistency in automated order processing. Hence, any web service discovery mechanism intended for application developers will be of limited use, unless certain techniques to automatically or semi-automatically recommend services that are suitable for composition as per their complex requirements are incorporated. Due to large volume of published web services, the demand for automated techniques to identify suitable composable services for fast application development has also increased.

The proposed methodology aims to discover relevant web services for a given task, be it basic services or a set of basic services (called a composable service set or composite service template) as per the user requirement. It is based on effectively capturing the functional semantics and service interface information of each service. This is used to improve the accuracy in finding basic services if they satisfy the query or possible compositions of available services that may together meet the requirements if a basic service alone is not satisfactory. Semantics based user query processing is also applied to capture the service requirements correctly and recommend relevant services for both simple and complex queries. In summary, the main contributions are -

- Designing an efficient semantics based service dependency capturing mechanism, to identify the constituent services of a valid composite service template.
- Demonstrating that the proposed mechanism correctly determines the invocation sequence of constituent services to satisfy given complex task requirements.
- Demonstrating that the proposed approach achieved good accuracy while discovering composite service templates, and the result generation time was satisfactory.

The paper is organized as follows. In Section 2, we present a discussion on existing work in the domain of basic and composite web service discovery. The proposed methodology and processes are presented in detail in Section 3 and the process of finding a composable service set for serving complex user queries is described in Section 4. Experimental results validating the approach used and theoretical performance analysis of the proposed technique are presented in Section 5, followed by concluding remarks & potential directions for future work and references.

2 Related work

Different approaches have been proposed for dealing with the web service discovery problem to overcome the limitations of keyword based matching and inefficient classification in UDDI registries. The public UDDI (Universal Description, Discovery and Integration) Business Registries were a major source of service advertisements, but were shut down in 2006. Several approaches based on using conventional search engines for discovering services on the open Web were proposed by Song et al. (2007) and Wu and Chang (2008). Extracting the functional information available in a WSDL (Web Service Description Language) document for use in various tasks like matchmaking and compositions are also available in previously published literature (Steinmetz et al. 2009; Fang et al. 2012). Specialized web service search engines like *Woogle* (Dong et al. 2004), *Seekda* (Pedrinaci et al. 2010) and *Service-finder* (Della-Valle et al. 2008) were developed, but none of these are currently accessible.

Many researchers have focused on the problem of lack of semantics in service descriptions and using inherent meaning in the natural language terms in a service description by applying natural language processing (NLP) techniques. Some approaches used the concepts of vector space model based indexing for extracting feature vectors from each service (Chan et al. 2011; Hao et al. 2010; Elgazzar and et al. 2010). Semantic techniques like Latent Semantic Indexing (LSI) and Probabilistic LSI were applied on WSDL documents in order to extract the inherent semantics in a service description for clustering and fast retrieval (Wu et al. 2008; Ma et al. 2008). Sangers et al. (2013) applied NLP techniques like lemmatization and Word Sense Disambiguation (WSD) to enhance the syntactic service description for optimizing the search process.

Peng and Wu (2010) proposed an improved semantic web service discovery method, that treats the service discovery problem as an assignment problem with functional constraints. They used a three-step matchmaking process – service library matchmaking, service matchmaking and operation matchmaking (interface and concept) for this purpose. Platzer and Dustdar (2005) developed a Vector Space Search Engine to index descriptions of already composed services. Each document is represented as a vector within a

Author's personal copy

'term space' where each dimension is a keyword. The position of this vector relative to others within the same vector space gives their pair-wise similarity.

Very few researchers have addressed the problem of composite service discovery. Brogi et al. (2005) proposed an algorithm called Service Aggregation Matchmaking (SAM) that indexes OWL-S (Web Ontology Language for Services) process models of services in the repository as a tree structure. The authors claimed that, for a given query, it can perform a fine-grained matching at the level of simple processes, however, no experimental results were presented for verifying the validity of their approach. Kuang et al. (2007) proposed a method for indexing outputs of all UDDI registered services, based on which a composition-oriented service discovery algorithm helps in filtering out irrelevant atomic services for a particular composition. Cuzzocrea and Fisichella (2011) described a method for using graph-based representation of composite OWL-S processes by considering both internal structure and component services. They also proposed an algorithm that matches over such representations and computes their degree of matching based on the similarity of the control flow.

Fethallah and Chikh (2013a) proposed an approach that exploits the service interfaces and the domain ontology, in order to index web services conceptually to determine relevance to a particular query. Liang and Su (2005) proposed an Graph Search Algorithm for discovering web services for composition. They presented a formalization of web service composition as a search problem by representing service nodes as AND nodes and data nodes as OR nodes in a graph. A graph traversal algorithm was used for searching the graph to match most relevant service and data nodes. Shiaa et al. (2008) proposed an incremental graph-based approach to automatic service composition. Filtering and reasoning is accomplished by validating the composition candidates using goal-based expressions and the search is based on breadth-first search algorithm, due to which the algorithm is exponential in complexity.

Li et al. (2013) developed the MTNet Project, which can be employed to assist developers' analysis and for the construction of the semantic models for Web services as per previously described definitions. They first redefined the corresponding state transitions by decomposing them further into sub-transactions and semantic transactions and are used to determine possible re-implementation of specific Web services to develop new composite services. Bianchini et al. (2015) applied data mining techniques to service descriptions to infer patterns representing the service's functionalities. These are used for knowledge-based querying and recommendation via a system call W-DREAM (Web services DiscoveRy via intEnsionAl knowledge Mining).

Feng et al. (2013) proposed a multi-granularity service composition technique that considers QoS that can recommend alternate and better component Web services when a new composition plan is generated. They used a behavioral signature model to capture the interface details of services, and proposed that two service composition plans are choreography equivalent if they conform to the same behavioral signature model. Rong et al. (2015) considered the fact that a service composition pattern can be significant hint for service selection, and used a service profiling mechanism to improve ranking and recommendation. Earlier service composition patterns are inferred via collaborative filtering based on similar set of users, using which a service re-ranking mechanism is employed for personalized recommendation.

In our proposed approach, we focus on using the inherent functional semantics of real-world web services for finding relevant services for a given query during service retrieval. The extracted information is enhanced using semantics based approaches and NLP techniques to generate additional metadata for each service in the form of weighted feature vectors. User requirements are obtained through a natural language querying interface and query analysis techniques are used to detect any subqueries in user request, that cannot be served by basic services only. During search, basic services are recommended if they satisfy the query, if not, possible compositions of available services that may together meet the requirements are retrieved, if found.

3 Proposed system

In this section, the proposed approach for automatic service discovery and the major processes in the proposed methodology are is discussed in detail. Figure 1 depicts the phases in the proposed framework. The major components are - *Service processing module*, *Query processing module* and the *Service discovey engine*.

3.1 Service processing module

To capture the service's functional semantics and also the input/output service dependencies correctly, the OWL-S description of a service is used in the proposed framework. OWL-S semantically describes the syntactic descriptions of the services using ontologies. Each OWL-S file consists of three sub-ontologies, *profile, process* and *grounding*, of which the *profile model* provides information on the capabilities of a service (service name, input/output names, natural language description, service provider details etc). Hence, the OWL-S profile model is helpful in capturing essential data about a web service while recommending

Author's personal copy

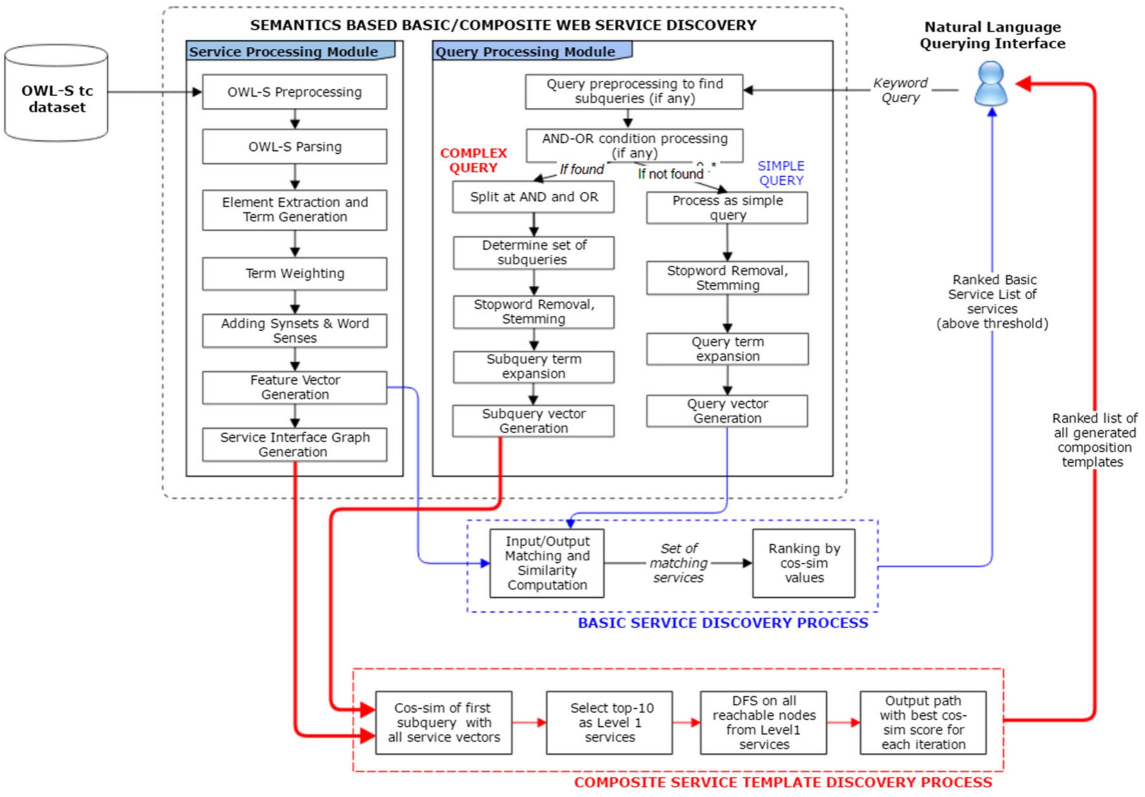

SEMANTICS BASED BASIC/COMPOSITE WEB SERVICE DISCOVERY
OWL-S tc dataset
Service Processing Module
OWL-S Preprocessing
OWL-S Parsing
Element Extraction and Term Generation
Term Weighting
Adding Synsets & Word Senses
Feature Vector Generation
Service Interface Graph Generation
Query Processing Module
Query preprocessing to find subqueries (if any)
AND-OR condition processing (if any)
COMPLEX QUERY
If found
If not found
SIMPLE QUERY
Split at AND and OR
Determine set of subqueries
Stopword Removal, Stemming
Subquery term expansion
Subquery vector Generation
Process as simple query
Stopword Removal, Stemming
Query term expansion
Query vector Generation
Natural Language Querying Interface
Keyword Query
Ranked Basic Service List of services (above threshold)
Ranked list of all generated composition templates
Input/Output Matching and Similarity Computation
Set of matching services
Ranking by cos-sim values
BASIC SERVICE DISCOVERY PROCESS
Cos-sim of first subquery with all service vectors
Select top-10 as Level 1 services
DFS on all reachable nodes from Level1 services
Output path with best cos-sim score for each iteration
COMPOSITE SERVICE TEMPLATE DISCOVERY PROCESS

Fig. 1 Proposed methodology

relevant ones to satisfy user queries and so, such details are extracted from the OWL-S profile and used for the conceptual indexation of dataset services.

The dataset used for the experiments is the OWL-S TC 4 collection.[1] It contains 1076 real world web service descriptions (WSDLs) from different domains like communication, education, economy, medical, travel etc., and their corresponding OWL-S files. The OWL-S profile model has a set of elements named <profile:hasInput>, <profile:hasOutput> and <profile:textDescription>. A sample OWL-S profile model defining the interfaces of its corresponding web services has been shown in Fig. 2.

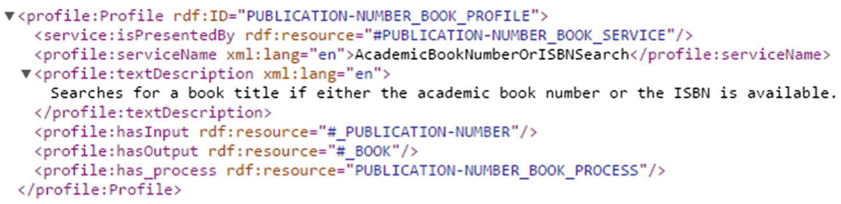
OWL-S parsing The profile model of each service in the dataset is first captured while parsing the OWL-S file. These elements are extracted and are processed further to extract information that will be used to index the web services conceptually. To extract the information, we developed an OWL-S Document Parser that parses each file and extracts the required elements.

Element extraction and term generation The elements <profile:hasInput>, <profile:hasOutput> and <profile:textDescription> extracted from the profile contain natural language phrases (as seen in Fig. 2). These phrases are extracted and further processed for obtaining the functional semantics and context of the service. For each of the phrases extracted, the natural language names are first split into tokens. These element names are mostly a combination of natural language words, as most service designers follow standard programming conventions and good naming practices like camel-casing, pascal-casing & underscores to ensure maintainability while naming such service elements. For example, the <profile:hasInput> element of the sample OWL-S shown in Fig. 2 is '*#_PUBLICATION-NUMBER*', which after splitting (using underscore and dash) and special character removal (#, , % signs), results in a term list of *'publication'*, *'number'*. Similarly, terms are extracted from the <profile:hasOutput> & <profile:textDescription> and are added to the term list. Table 1 summarizes the rules followed for obtaining terms from natural language element names and phrases for generating the feature vector.

After all the term tokens are extracted, any stop words (like *'get'*, *'the'*, *'or'*, *'is'* etc., that do not contribute to the semantics of a service) are filtered out. In addition to this, words used commonly in web services domain, referred to as *function words* (for e.g. *'service'*, *'input'*, *'output'*, *'request'*, *'response'*, *'process'* etc) are also filtered as they contribute very little to the context of a service with respect

[1]OWL-S Service Retrieval Test Collection version 4, Available online http://projects.semwebcentral.org/projects/owls-tc/

Author's personal copy



```
▼<profile:Profile rdf:ID="PUBLICATION-NUMBER_BOOK_PROFILE">
   <service:isPresentedBy rdf:resource="#PUBLICATION-NUMBER_BOOK_SERVICE"/>
   <profile:serviceName xml:lang="en">AcademicBookNumberOrISBNSearch</profile:serviceName>
  ▼<profile:textDescription xml:lang="en">
     Searches for a book title if either the academic book number or the ISBN is available.
   </profile:textDescription>
   <profile:hasInput rdf:resource="#_PUBLICATION-NUMBER"/>
   <profile:hasOutput rdf:resource="#_BOOK"/>
   <profile:has_process rdf:resource="PUBLICATION-NUMBER_BOOK_PROCESS"/>
 </profile:Profile>
```

Fig. 2 Profile model of a sample OWL-S document

to a query. Finally, the remaining terms are stemmed to their root form and are added to the OWL-S document's feature list. The process is repeated for all OWL-S documents in the dataset.

Term weighting The terms obtained for each OWL-S file may range from 30 - 50 terms, all of which cannot be part of that service's feature vector. Since all the features in the generated vector cannot have equal importance for a particular service, a term weighting scheme or a suitable feature selection technique should be applied. To rank the term candidates and choose the top few as the feature vector for a service, an integrated approach was used to generate a ranking for the terms, using a term weighting method called Tf-idf (term frequency-inverse document frequency), which is used to compute the importance of a particular term in a given service when compared to other services in a corpus. Tf-idf is given by -

$$Tf-idf = (0.5 + \frac{tf}{tf_{max}}) \cdot log(0.5 + \frac{D}{df}) \quad (1)$$

where tf is the frequency of term t in the given OWL-S document d; tf_{max} is the highest frequency of any term in the OWL-S document d, df is the frequency of occurrence of the term t in other OWL-S documents and D is the total number of OWL-S documents in the dataset. Using Eq. 1, a sorted list of Tf-idf values for each OWL-S term is built and the top 5 terms are added to its feature vector. This feature vector effectively represents the relative importance of each term word in the service document, with respect to all the other documents in the dataset.

Adding synsets and word senses Next, each of the terms is searched in WordNet using Python NLTK packages to extract the related upper concepts like hyponym/hypernyms. These concepts are utilized to determine the category of the web services to be searched. NLTK provides several path similarity computation algorithms like Leacock-Chodorow Similarity, Wu-Palmer Similarity etc, that use WordNet path hierarchy concepts to extract the root concepts of the concept hierarchy that have the terms as its leaf nodes. Once obtained, the WordNet synsets (synonyms) of each of the terms are extracted.

Now, the problem is that each of these terms may have multiple word senses, based on possible English language usage. Also, the overall functionality of a web service is typically described in natural language within the <profile:textDescription> element in the profile model. This is intended for the use of human readers to understand the capabilities of the service. However, the terms used by the service designer in the service's natural language description may sometimes be ambiguous, for example, a term *'book'* may be meant as a verb or a noun, thus affecting the accuracy if the user expectation was different.

To capture context correctly, a simple word sense disambiguation algorithm called JCN Similarity algorithm (Jiang and Conrath 1997) was applied to choose the right meanings of each of the words obtained from the profile model. JCN Algorithm captures the semantic relatedness between word senses in a phrase as per the Jiang-Conrath Similarity metric. It returns a score denoting how similar two word senses are, based on the Information Content (IC) of the Least Common Subsumer (most specific ancestor node) and that of the two input Synsets. The score is used to choose the most relevant word sense for each subquery term, in context with the rest of the POS tagged query, and is calculated as per below.

$$Score = \frac{1}{(IC(s_1) + IC(s_2) - 2 * IC(lcs))} \quad (2)$$

Table 1 Term vector generation process - Examples

Element name	Splitting Rule(s)	Initial terms	Final terms
filmAction	pascal-casing	*film, action*	*film, action*
BookAuthor	camel-casing	*book, author*	*book, author*
latitude1	suffix number elimination	*latitude, 1*	*latitude*
ISBN_number	underscore operator	*isbn, number*	*isbn, number*
SMStoIndiaRequest	Contiguous capital letters; Pascal casing	*sms, to, india, request*	*sms, india*
getFamily-roomPrice	Pascal-casing; Dash operator	*get, family, room, price*	*family, room, price*

Author's personal copy

where, $IC(s_1)$ and $IC(s_2)$ are the information content of the two word senses, and $IC(lcs)$ is the information content of their Least Common Subsumer. This score is recursively calculated for all the word senses obtained from WordNet for each SR term. The sense with the highest score is chosen as the most relevant word sense for each subquery term, in context with the rest of the subquery. This is the query expansion phase, giving the final SR, as below.

Feature vector generation Finally, a service feature vector $\mathbf{s_k}$ is created to represent each OWL-S in vector space. Vector $\mathbf{s_k}$ contains the final terms obtained from the profile model elements considered, after all semantic processed are complete. This procedure is repeated for all OWL-S files in the dataset, at the end of which, an indexed dataset that contains the vector representation of every OWL-S document is obtained. Each OWL-S document is indexed by its corresponding set of top 5 terms and identified senses of each.

3.2 Query processing module

To understand user's context and display relevant results, it is necessary to analyze the meaning and structure of the query text entered by the user. It is also necessary to determine if any subqueries are present that may require a composite service set to serve the query. The query analysis process is composed of several preprocessing tasks as below

Subquery identification Once the user query is submitted, it has to be analyzed to 'understand' the desired output and to determine if any subqueries are present. Firstly, the query is part-of-speech (POS) tagged using the Stanford POS tagger. In the POS tagged query, terms tagged as */CC* are basically the *Coordinating conjunctions* in the text. Coordinating conjunctions connect words, phrases & clauses and are made up of seven different words – *'and', 'or', 'but', 'for', 'nor', 'yet'* and *'so'*. At present, only two coordinating conjunctions, *'and'* & *'or'*, have been considered in the input query to determine if any subqueries exist.

AND/OR condition processing In the proposed approach, if the *'and'/'or'* coordinating conjunctions are found, then the query is treated as a complex query. Whenever an *'and'/'or'* is found, each part is processed as an independent query, which helps in determining each individual set of services that form a part of the composite candidate set that can serve the complex user request. For example, if the submitted query is *"book hotel or resort and hire taxi"*, then, the tagged query is as shown below:

book/NN hotel/NN or/CC
resort/VB and/CC
hire/VB taxi/NN

Determining set of subqueries In the discussed example, the input query is composed of three subqueries, where the user wants services that find hotel *or* resort in London, *and* then to hire a car. After AND/OR condition processing, the user request is split into three subqueries *"book hotel"*; *"book resort"* and *"hire taxi"*. This request is treated as a complex request and each query is run independently at first and then the results are put together as per its logical representation of $((SQ1 \vee SQ2) \wedge SQ3)$. In case of the *'or'* condition, the service which has the highest similarity value with the subquery vector is chosen, then the top most service(s) are added to the service(s) found for the *'and'* condition, and displayed as the best composable service set (illustrated in Fig. 3). This process is described in detail in section IV.

Stopword removal & stemming In this phase, stop words, excluding the coordinating conjunctions, if any, are filtered out from each subquery. Next, stemming is performed for each term in each subquery to take care of the different forms of a natural language word so they can be analyzed as a single item. Examples are term occurrences like *'books', 'booking', 'booked'* etc which are different forms of *'book'*.

Query expansion Each subquery will be processed independently and parallely. The WordNet synsets of each final subquery term are retrieved to capture the synonyms and added to the subquery feature vector. To capture user context correctly, the JCN Similarity algorithm (Jiang and Conrath 1997) is again used to choose the right meanings of each of the words in the subquery, which returns a score denoting how similar the word senses of the two input synsets are, similar to how the process was carried out for service descriptions. The score is used to choose the most relevant word sense for each subquery term.

Query feature vector generation For each subquery, the terms, their synsets and the identified senses together form its feature vector $\mathbf{V}_{sq_t}$ This represents each subquery in the vector space similar to the OWL-S documents, which allows vector operations like dot product and cosine similarity to be performed to determine level of similarity between service and query components.

3.3 Similarity computation

The proposed system is a web service discovery framework for both basic and composite web services as per query requirements. As both the services and the query are represented in vector space, vector operations like dot product can be performed to determine the angle between each vector, based on which a similarity value can be computed. To compute similarity between potentially relevant

Author's personal copy

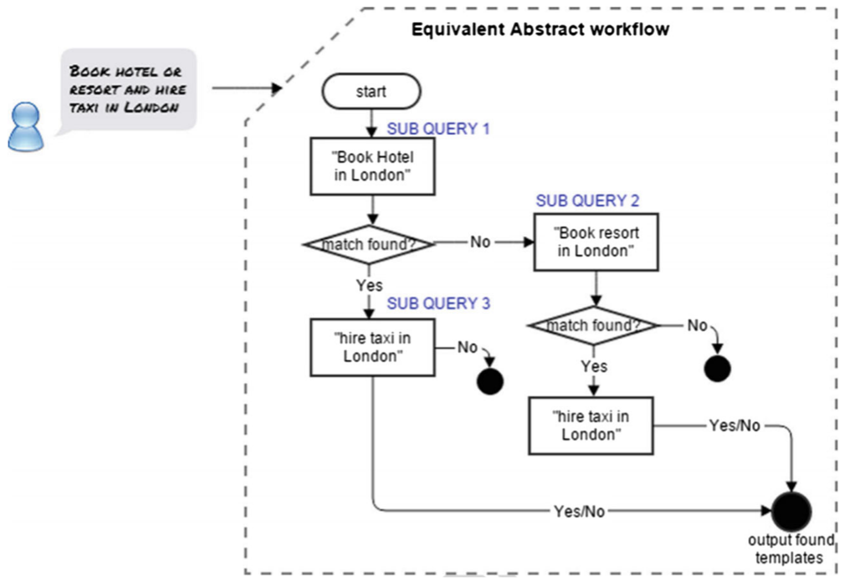


Fig. 3 Equivalent abstract workflow for a sample complex query

services and the query during the web service search process, the Cosine Similarity Measure is used. After the query vector generation, the *Cosine Similarity* as given by the Eq. 3 is computed between each subquery vector and service input/output vector, in order to determine potentially relevant services that serve the user's requirements.

$$\cos\theta(\mathbf{V}_{out_{S_i}}, \mathbf{V}_{in_{S_k}}) = \frac{\mathbf{V}_{out_{S_i}} \cdot \mathbf{V}_{in_{S_k}}}{\|\mathbf{V}_{out_{S_i}}\| \|\mathbf{V}_{in_{S_k}}\|} = \frac{\sum_{j=1}^{n}(w_{j,i}.w_{j,k})}{\sqrt{\sum_{j=1}^{n} w_{j,i}^2}.\sqrt{\sum_{j=1}^{n} w_{j,k}^2}} \tag{3}$$

where,

$w_{j,i}$ represents the relative weight of the output vector terms of service S_i,

$w_{j,k}$ represents the relative weight of the input vector terms of another service S_k.

3.4 Service matchmaking and ranking

Using the Eq. 3, the level of matching between the query and services in the dataset is to be calculated next. In case of a simple query, the cosine value between the query vector and the service dataset is found and thereafter, the similarity values are sorted from the greatest to the least and all matching services that have a similarity value greater than a given threshold are returned as a ranked list. The threshold value is currently set at 0.75 (range 0 to 1). In the case of a complex query, the service request has to be first processed (as described in Section 3, subsection B), to determine the constituent sub-queries and their respective feature vectors. After this, the service matching process takes place using a technique called the Service Interface Graph (SIG) which aims to capture each service's input/output dependencies. The process is described in detail in section IV.

4 Finding a composable service set

Very often, a single service may not be able to satisfy user's query completely, especially in a case of a complex query. In such cases, several different services that can work together in a given sequence, to provide the intended functionality may be need to be discovered. Based on this discovery, the user would be able to develop applications that make use of several basic services. Such a service set is called a composite service template or a composable service set. Discovering composite service templates is a process that has certain requirements like -

- searching for suitable web services that can together satisfy the user requirements.
- determining the sequence in which these different web services are to be placed to yield the desired output.
- capturing the service dependencies and data formats of each service such that, the output of the preceding service in the sequence matches the input format expected by the next service in the sequence.

Several solutions to capturing service dependencies for service composition problem have been proposed that are based on techniques like petrinets (Hamadi and Benatallah 2003), graph theory (Oh et al. 2005; Hashemian and Mavaddat 2005) and BPEL4WS (Business Process Execution Language for Web Services) (Wohed et al. 2003). We propose a methodology based on automatic construction of a *Service Interface Graph* to extract the service dependencies and data formats of each service, while searching for

Author's personal copy

multiple web services that can be composed together to serve to user's needs.

4.1 Constructing the service interface graph

The proposed technique, the SIG, is a directed acyclic graph (DAG) that is constructed to model services and the relation between their interfaces. Each node in this graph represents a web service. A node can have several incoming and outgoing edges. An edge from node 'N_i' to node 'N_j' signifies that the output of a service 'N_i' is similar to the input accepted by service 'N_j' i.e., service 'N_i' and 'N_j' can be chained together in order. Algorithm 1 depicts the process of constructing the service interface graph.

Algorithm 1 Service Interface Graph construction process

```
Input: Input & output vectors of all OWL-S services in dataset
Output: The Service Interface Graph, SIG
 1: Represent each service as a node in the SIG
 2: for each service node $S_i$ in the SIG do
 3:     Compute cos-sim of $\mathbf{V}_{out}$ of service $S_i$ with $\mathbf{V}_{in}$ of all other services $S_k$
 4:     for each service node $S_k$ do
 5:         if cos-sim($\mathbf{V}_{out_{S_i}}$, $\mathbf{V}_{in_{S_k}}$) > $cutoff$ then      ▷ Currently $cutoff$ is set to 0.75
 6:             Add an edge between SIG node $S_i$ and $S_k$
 7:             Check SIG for any cycles
 8:             if cycle is found then
 9:                 Delete last added edge
10:             else
11:                 Add new edge to SIG adjacency list
12: Save SIG in memory by its adjacency list.
```

To construct the graph, the input and output vectors of all the OWL-S services in the dataset are used. Then, matchmaking is performed between the output of each service to the input of every other service i.e., the cosine similarity between the output vector of first service and input vector of second service is determined. If the similarity is found to be greater than a pre-determined cutoff, then the two services are connected via a directed edge from the first service to the second.

After the addition of each edge, the graph is checked for cycles. If any cycles are found to exist in the graph, then the recently added edge is discarded. The main objective is to model services such that composite service discovery problem can be treated as a simple graph traversal problem. Thus, it is essential to have an acyclic graph. The similarity cutoff for adding edges is currently fixed at 0.75. A high value is chosen to ensure that there is an almost perfect match between the interfaces. Once constructed, the DAG of services and their interface dependency is stored in the memory using its adjacency list representation. Graph construction takes place only once when the server is started for the first time. Once the graph is constructed, it has to be traversed for each complex query. The graph resides in main memory and the same graph is used for each query. Within the graph, each node contains -

- Service Name or Service Identifier.
- Vector representation of service interfaces - input and output.
- A list of nodes to which the current node has an outgoing edge.

4.2 Serving complex queries

Whenever a user submits a request to the system, the first step is to process the query to determine if it is a simple or a complex query. For a simple query, there are no *'and'*/*'or'* keywords, so the cosine similarity value between query feature vector and output vectors of all services is recursively computed, then services are sorted by their cos–sim values and all services with similarity above a threshold of 0.75 are displayed to the user. In the case of a complex query, the DAG has to be traversed based on the subquery components. Three different types of queries were identified based on the occurrence of *'and'* & *'or'* - *sequence* (only *'and'* in the complex query), *choice* (only *'or'* in the complex query) and *mixed* (both *'and'* and *or* can occur). Algorithm 2 depicts the process of executing complex queries using the constructed service interface graph.

Firstly, it is assumed that the user specifies what is wanted (i.e. the output). Therefore, the terms obtained for each subquery are considered as part of its output vector. For a complex query, each subquery is processed and their feature vectors are generated. For the first subquery, cosine similarity is computed between subquery vector and the output vectors of all services and the top-10 services

Author's personal copy

Algorithm 2 Executing complex queries using SIG

```
Input:
    Service Interface Graph SIG
    Number of subqueries $n$ of complex query $Q$
    Subquery vectors $\mathbf{V}_{sq_n}$ representing each subquery of complex query Q
Output: Ranked list of all obtained composite service templates.
 1: Level $l = 0$                                                        ▷ Start SIG traversal
 2: for first subquery $sq_1$ of $Q$ do
 3:     Compute cos-sim of $\mathbf{V}_{sq_1}$ and $\mathbf{V}_{out}$ of all nodes     ▷ Services are the nodes in the SIG
 4:     Rank results by their cos-sim values
 5:     Select top-10 nodes as level 1 services
 6:     set $l = 1$                                  ▷ Relevant services for subquery $sq_1$ identified
 7: for $l$= 1 to $n$; $n++$ do            ▷ Traverse SIG levels for each subquery upto $sq_n$
 8:     for each node on level $l$ do                                    ▷ Input to next level
 9:         Perform DFS traversal of SIG                                  ▷ 10 iterations
10:         for every new node visited do
11:             Find cos-sim of $\mathbf{V}_{\mathbf{sq_1}}$ & $\mathbf{V}_{out}$ of service node
12:             Store node with highest cos-sim value
13:         return path and all nodes between source node and node with best output    ▷ Composite service
    templates obtained.
14: return all possible composite service templates to user ranked by aggregate cos-sim values.
```

with best similarity scores are identified. This gives the top-10 start-points for finding all potential composable service sets, for the rest of the subqueries. This is considered as the first level of services that satisfy a part of the user request.

For the next subquery, each of the first level services' outputs are considered as inputs. So, for each of these inputs, the graph is traversed using the well known Depth First Search (DFS) Algorithm starting from the input service as source. For every node visited, the similarity between the node's output vector and corresponding subquery vector is determined and the node with best output vector is selected. This process is continued for all the subqueries in the user request. Finally, the path between the corresponding source node and the node with best output yields the composite service template for the corresponding source service. After all iterations, all possible composite service templates are obtained. These are ranked by their aggregated path cos-sim values, and the ranked list of generated composite service templates is returned to the user. To optimize the result generation time when a complex query is received, the graph is traversed a predefined number of times (currently set to 10), for each possible source node, in parallel.

5 Experimental results and analysis

In this section, a discussion on the theoretical and experimental evaluation of the proposed approaches for discovering composable service set using functional semantics and service dependencies is presented. The proposed framework is primarily a web service search system where users can submit natural language queries and get recommendations on relevant basic services that satisfy the request or get more information on a set of basic services that may together be able to accomplish the goal. Hence, the performance of the system for various query classes using appropriate metrics is summarized in Section 5.1. The accuracy of the composition service templates generated by the system is presented in Section 5.2 and the theoretical analysis of the system's performance is discussed in Section 5.3.

5.1 Web service retrieval

To evaluate the web service retrieval performance of the proposed method, several experiments using different classes of queries were performed by varying the number of OWL-S documents taken from the OWL-S TC 4 dataset which provides a collection of 1076 web service descriptions from different domains like - Communication (58), Economy (359), Education (285), Food (34), Geography (60), Medical (73), Simulation (16), Travel (165) and Weapon (40). The experiments were carried out on a Intel® Core i7™ Quad-Core Workstation with 16GB DDR3 SDRAM and 1TB hard drive.

To evaluate the service discovery process, experiments were conducted using various types of queries as listed in Table 2. Since the quality of returned results as well as the time taken for generating the results may vary with the size of the dataset, each experiment is conducted for different number of OWL-S documents. For each test, simple queries basically can be satisfied by finding relevant basic services, while the complex queries (sequence, choice and heterogeneous) can be satisfied only by a set

Author's personal copy

Table 2 Experimental setup and sample queries used for different testcases

Case No.	No. of services	Service Domains	Complex Query Type	A Sample Query
1	100	Medical(60)	sequence	treatment and hospital room availability
		Geography(40)	choice	clinic or hospital address
			mixed	clinic or hospital in city and distance from address
2	300	Education(300)	sequence	degree and research funding offers
			choice	research funding or job opportunity in city
			mixed	scholarship or research funding and parttime job in city
3	500	Travel(165)	sequence	hotel in city and car hire
		Geography(60)	choice	sports or adventure activity near address
		Economy(275)	mixed	sports events and weather in city or country
4	1076	All domains	sequence	weather and sunrise time in city
			choice	address of bank or ATMs in city
			mixed	price and genre of book for given title or isbn

of basic services, composed in the right order as per their input/output dependencies. The number of OWL-S documents considered for the various testcases were 100, 300, 500 and 1076. For each testcase, both simple and all complex queries were submitted to the system to evaluate the performance for varied dataset sizes.

Since the proposed system is modeled as per IR principles, the most appropriate metrics to evaluate the performance are *precision* and *recall*. In this context, precision can be defined as the fraction of retrieved services that are relevant for the given query. Recall is the fraction of all relevant services to those that were successfully retrieved for a given query. Table 3 summarizes the observed experimental results for the various cases considered.

$$Precision = \frac{|relevantservices| \cap |retrievedservices|}{|retrievedservices|} \tag{4}$$

$$Recall = \frac{|relevantservices| \cap |retrievedservices|}{|relevantservices|} \tag{5}$$

$$F - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{6}$$

For each test, a set of 20 different queries for each query type were run on the system and the average value of precision, recall, and result generation time were noted. The best values for precision were observed during testcase 1 using 100 services at 95.94 % while the lowest precision value of 61.21 % was obtained for testcase 4, where 1076 services were considered. Figure 4 shows the precision-recall performance and the balanced F-measure values obtained during Web service discovery. It can be observed that the balanced F-measure values indicate good precision-recall performance initially, which deteriorate as the number of services considered for the experiment increases. The average value of observed precision for all the testcases was 79.28 % and average recall was 33.15 %. Figure 5 shows the time taken to generate the results in each testcase for different complex query types. The average result generation time was about 2.22 minutes.

Table 3 Experimental results for Web service discovery for various testcases

Case No.	No. of services	Complex Query Type	Precision (%)	Recall (%)	F-Measure (%)	Time taken (in mins)
1	100	sequence	95.94	43.03	59.41	00:26
		choice	94.3	39.39	55.57	00:33
		mixed	91.53	52.12	66.42	00:49
2	300	sequence	82.67	18.33	29.77	01:34
		choice	79.12	40.00	52.89	01:42
		mixed	78.04	53.33	62.39	02:02
3	500	sequence	78.33	16.60	26.36	02:24
		choice	75.14	14.40	24.15	02:34
		mixed	63.9	21.18	33.17	02:55
4	1076	sequence	74.8	11.50	19.37	04:17
		choice	76.4	9.65	17.13	04:25
		mixed	61.21	12.89	21.30	04:38

Author's personal copy

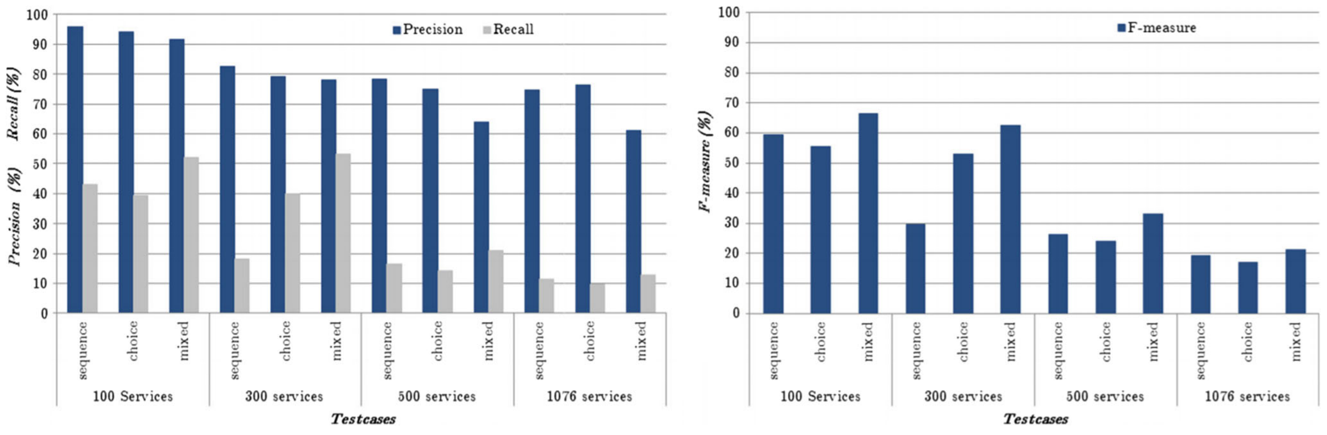


Fig. 4 Precision-recall and balanced f-measure values obtained during Web Service Discovery, for various testcases

5.2 Composition template generation accuracy

To evaluate the quality of composite service templates generated, some additional statistics were collected. These include -

1. *Number of correct templates generated* ($CT_{correct}$). Composite service templates that completely satisfy the requirements of the complex query.
2. *Any partial templates generated* ($CT_{partial}$). Templates that satisfy at least 75% of the complex query requirements.
3. *Any incorrect templates generated* ($CT_{incorrect}$). Templates with wrongly identified constituent services or incorrect invocation sequence.

Table 4 depicts the results of the composite service discovery process. The retrieved templates in each case were subjected to a human evaluation and categorized as *correct, partial* and *incorrect* templates. Templates that satisfy at least 75% of the complex query requirements were considered successful results, as they still provide significant help to application designers. As per this criteria, the accuracy of composite service template generation using the proposed methodology is calculated as per Eq. 7.

$$\%Accuracy = \frac{(CT_{correct} + CT_{partial} - CT_{incorrect})}{CT_{total}} * 100 \tag{7}$$

As seen from Table 4, the average composite service template accuracy observed for all the testcases considered was 75.09 %. It was observed that the accuracy can deteriorate due to I/O datatype mismatches in some of the identified templates. However, the number of incorrect templates generated was quite low, while partial templates that matched more than 75 % of the complex query requirements were also obtained. Hence, it can be concluded that the composite service template generation accuracy using the proposed methodology was satisfactory.

We noticed an additional issue, that of *I/O datatype conflicts* in some correct/partial composite templates generated. Templates with I/O datatype conflicts ($CT_{conflict}$) are cases where the templates may be correct at the semantic level of

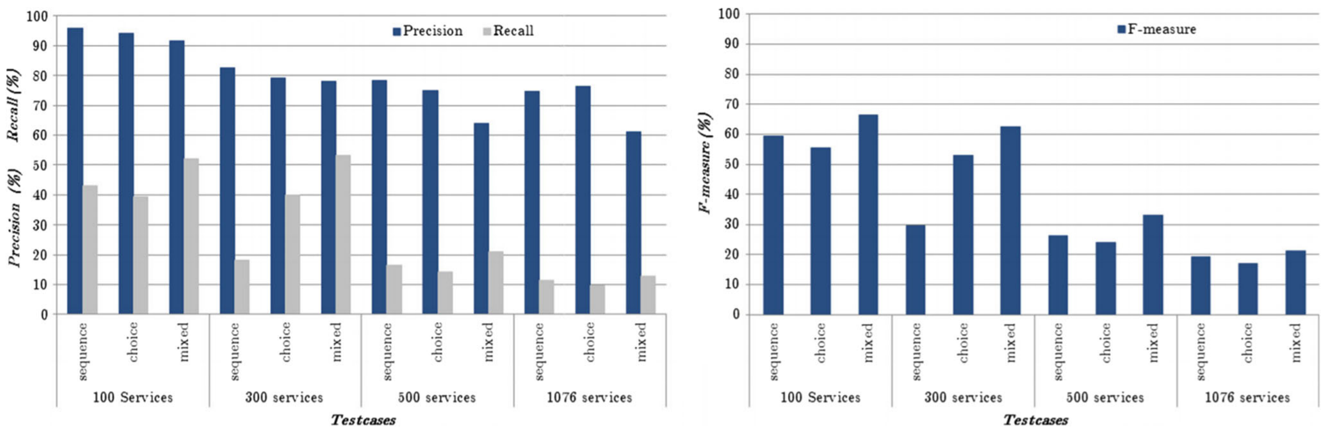


Fig. 5 Result generation time for the various testcases

Author's personal copy

Table 4 Composition template generation accuracy

			No. of Composite Service Templates generated per type			
Testcase No.	No. of services	Complex query type	Correct	Partial	Incorrect	Accuracy (%)
1	100	sequence	2	1	0	100
		choice	2	0	0	100
		mixed	1	1	0	100
2	300	sequence	3	2	0	100
		choice	3	1	1	60
		mixed	2	1	1	50
3	500	sequence	4	3	1	87.5
		choice	3	3	1	71.4
		mixed	2	3	1	66.67
4	1076	sequence	4	4	2	60
		choice	4	2	2	50
		mixed	3	4	2	55.56
						75.09

I/O matching, but the datatypes of the output parameter of first service and the input parameter of second service do not match. For example, consider a service *HotelRoomBooking*, which takes the booking number as input (*xsd:int*) and gives the allotted room details (*xsd:string*) as output. When this service is recommended in a composite service template, and the next service in the chaining requires a *xsd:string* booking number, then an I/O datatype conflict will arise. In this case, even if the correct constituent services and invocation order has been identified, the template may be useless to application designers. Currently, cases with I/O datatype conflicts have not been explicitly handled. In future work, we intend to address this case, to further improve the accuracy of the proposed method. Table 5 provides the statistics on I/O datatype conflict templates obtained during the experiments for each of the testcases.

5.3 Theoretical analysis

Generating composition templates that satisfy a given user request is a very complex problem. In general, time complexity is a crucial criterion for discovering composite service templates as every possible combination of available services must be considered to determine input-output dependencies. In this section, we analyse the performance of the developed approach, by considering each of its major critical processes and the time complexity with reference to them.

Table 5 Templates with I/O datatype conflicts obtained during experiments

Testcase No.	No. of services	Total templates with I/O Conflicts per testcase
1	100	0
2	300	1
3	500	3
4	1076	6

Indexing OWL-S services Indexing a single OWL-S document after extracting its functional semantics and semantic vector generation takes constant time and is of $\mathcal{O}(C)$ time complexity. Then, the process of indexing a dataset of N services takes $\mathcal{O}(N)$ time.

Query analysis This task involves semantically analysing the request to identify any subqueries and the query vector generation for each individual query. It is dependent on the number of subqueries in the request, which if present, may have a few words at the most, which can be processed in constant time. Hence, this task takes $\mathcal{O}(C)$ time.

Constructing the service interface graph While constructing the Service Interface Graph, the semantic input vector of each service has to be matched with the semantic output vector of every other service. Thus, the time complexity of graph construction is $\mathcal{O}(N^2)$, where N is the number of services in the dataset. The process of checking for cycle formation takes constant time $\mathcal{O}(C)$ and thus it doesn't add anything to the complexity.

Adding new services to the service interface graph The SIG is constructed only once when the server is started and

Author's personal copy

need not be constructed repeatedly as it is stored in the memory. Whenever a new service is added to the database, a new node representing the service data will have to be appended to the graph. Hence, its input/output vectors have to be matched with that of every other service already in the graph, therefore, adding a new node is also of $\mathcal{O}(N^2)$ complexity.

Traversing the service interface graph The task of SIG traversal to find composite service templates is to be carried when a query is submitted to the system. To search for matching services, the expected output is extracted from the user query. This process involves finding the cosine similarity between every reachable OWL-S document from source node, which is of time complexity $\mathcal{O}(N)$. As the SIG traversal is based on finding the topological order, i.e., the algorithm process all nodes first and then for every level 1 node, the same process is run on all adjacent nodes. Since total adjacent nodes or vertices in a graph is given by $\mathcal{O}(E)$, the overall time complexity of this process is $\mathcal{O}(V + E)$. Even if the entire SIG has to be traversed and a match is still not found, the process is of at most $\mathcal{O}(V + E)$ complexity. In the SIG, services are nodes/vertices, hence $V = N$. Thus, the time taken for this task is $\mathcal{O}(N)$. $\mathcal{O}(N + E)$.

Therefore, the overall complexity of the system can be presented as - $\mathcal{O}(N)$ + $\mathcal{O}(N^2)$ + $\mathcal{O}(C)$ + $\mathcal{O}(N^2)$ + $\mathcal{O}(C)$ + $\mathcal{O}(N)$ + $\mathcal{O}(N)$. $\mathcal{O}(N + E)$ = $\mathcal{O}(N^2)$. Hence, it can be concluded that the system is quite efficient and can support scalability. If deployed in a computing environment that supports large graph storage and processing, the proposed methodology would be able to handle a large number of web services and still return results in a reasonable amount of time.

5.4 Comparing proposed approach with existing techniques

Earlier approaches for composition-oriented service discovery can be categorized into *indexing based* (Brogi et al. 2005; Xie et al. 2006; Kuang et al. 2007), *graph based* (Liang and Su 2005; Cuzzocrea and Fisichella 2011; Shiaa et al. 2008), *semantics based* (Li et al. 2013; Fethallah and Chikh 2013b) and *non-functional parameter based* (Feng et al. 2013; Rong et al. 2015) approaches. Indexing based methods focus on using the service inputs and outputs for manual index creation, so that matching services can be retrieved based on user query. However, these natural language terms can have synonyms and hypernyms, or may be used in two different senses in two different services, so these two services cannot be considered similar. This distinction has not been considered in any approach discussed under indexing based methods. Semantics and ontology based methods alleviate the disambiguation problem to a certain extent, but the process of searching for the desired output for a given input can be exhaustive. To overcome this, computationally intensive techniques like AI planning (Akkiraju et al. 2006) and logic based reasoning (Li et al. 2013) have been proposed.

Graph based techniques have been found to be the most advantageous as these focus on representing the available services by their dependencies. With the inclusion of semantics based processing and intelligent indexing, graph based algorithms can be made very efficient, hence we adopted this approach in our work. Some problems faced by existing graph based approaches while searching for possible compositions, like cycle detection and avoidance, graph maintenance and optimized traversal techniques affect their accuracy (as low as 40% accuracy), and these were incorporated in our work. The SIG based technique incorporates all the best features of indexing based, graph based and semantics based approaches. Incorporating QoS and usage based parameters will be considered as an extension to the work presented in this paper.

6 Conclusion and future work

In this paper, a framework for discovering both simple services and composable service set as per user requirements is proposed, which uses natural language processing and semantics based techniques to extract the functional semantics of the service dataset and also to understand user context. In case of simple queries, basic services may be enough to satisfy the user request, however, in case of complex queries, several basic services may have to be identified to serve all the requirements. For this, the system uses the service dependencies of all the services for constructing a service interface graph, which is then traversed to determine suitable services based on the required output. To evaluate performance of web service discovery, we conducted several experiments in terms of retrieval precision and recall, f-score and time taken for result generation. In addition, the number of templates generated, number of correct templates, partial templates and incorrect templates generated were also observed, in the case of complex queries. The overall accuracy of composable service set generation of the system was found to be 75.09 %. The experimental results show that our method generates composable service templates with a good level of accuracy and also is scalable to handle larger datasets efficiently.

During composable service template generation, some generated templates exhibited I/O data type conflicts, i.e., a service that can be chained with another service in a given order was found to be incompatible due to datatype mismatch. This is because, currently we have considered only the OWL-S profile's <profile:hasInput>,

Author's personal copy

<profile:hasOutput> and <profile:textDescription>. As part of future work, this problem will be addressed so that datatype mismatch problem can be solved and the accuracy can be further improved through SIG optimization. Currently, optimized data structures like hash tables were used to store the adjacency list of the SIG in memory. Incorporating NOSQL databases like neo4j, mongodb, orientdb etc can make this more efficient and support higher scalability. Also, the user query processing and composition-oriented discovery can be further enhanced by allowing users to specify preconditions and effects implicitly, instead of just outputs. Other factors like Quality of Service (QoS) & user specified constraints can be incorporated.

References

Akkiraju, R., Srivastava, B., Ivan, A., Goodwin, R., & Syeda-Mahmood, T. (2006). Semantic matching to achieve web service discovery and composition., *E-Commerce Technology, 2006. The 8th IEEE International Conference on and Enterprise Computing, E-Commerce, and E-Services* (pp. 70–70).

Almasri, E., & Mahmoud, Q.H. (2009). A broker for universal access to web services., *Communication Networks and Services Research Conference, 2009. CNSR'09. Seventh Annual* (pp. 118–125).

Benatallah, B., Hacid, M.S., Leger, A., Rey, C., & Toumani, F. (2005). On automating web services discovery. *VLDB J.*, *14*(1), 84–96.

Bianchini, D., Garza, P., & Quintarelli, E. (2015). Characterization and search of web services through intensional knowledge. Journal of Intelligent Information Systems pp 1–27.

Brogi, A., Corfini, S., & Popescu, R. (2005). Composition-oriented service discovery., *Software Composition* (pp. 15–30): Springer.

Chan, N.N., Gaaloul, W., & Tata, S. (2011). A web service recommender system using vector space model and latent semantic indexing. In *2011 IEEE International Conference on Advanced Information Networking and Applications (AINA)* (pp. 602–609).

Cuzzocrea, A., & Fisichella, M. (2011). Discovering semantic web services via advanced graph-based matching. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC) 2011* (pp. 608–615).

Della-Valle, E., Cerizza, D., Celino, I., Turati, A., Lausen, H., Steinmetz, N., Erdmann, M., & Funk, A. (2008). Realizing service-finder: Web service discovery at web scale. In *European Semantic Technology Conference (ESTC), Vienna*.

D'Mello, D., Ananthanarayana, V., & Santhi, T. (2008). A qos broker based architecture for dynamic web service selection. In *Modeling & Simulation, 2008. AICMS 08. Second Asia International Conference on* (pp. 101–106).

Dong, X., Halevy, A., Madhavan, J., Nemes, E., & Zhang, J. (2004). Similarity search for web services. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30, VLDB Endowment* (pp. 372–383).

Elgazzar, K., et al. (2010). Clustering WSDL documents to bootstrap the discovery of web services. In *IEEE Intl Conf on Web Services (ICWS), IEEE*.

Fang, L., Wang, L., Li, M., Zhao, J., Zou, Y., & Shao, L. (2012). Towards automatic tagging for web services. In *2012 IEEE 19th International Conference on Web Services (ICWS)* (pp. 528–535).

Feng, Z., Peng, R., Wong, R.K., He, K., Wang, J., Hu, S., & Li, B. (2013). Qos-aware and multi-granularity service composition. *Infor. Syst. Front.*, *15*(4), 553–567.

Fethallah, H., & Chikh, A. (2013a). Automated retrieval of semantic web services: a matching based on conceptual indexation. *Int. Arab. J. Inf. Technol.*, *10*(1), 61–66.

Fethallah, H., & Chikh, A. (2013b). Automated retrieval of semantic web services: a matching based on conceptual indexation. *Int. Arab. J. Inf. Technol.*, *10*(1), 61–66.

Hamadi, R., & Benatallah, B. (2003). A petri net-based model for web service composition. In *Proceedings of the 14th Australasian database conference-Volume*, (Vol 17 pp. 191–200): Australian Computer Society, Inc.

Hao, Y., Zhang, Y., & Cao, J. (2010). Web services discovery and rank: An information retrieval approach. *Futur. Gener. Comput. Syst.*, *26*(8), 1053–1062.

Hashemian, S., & Mavaddat, F. (2005). A graph-based approach to web services composition. In *The 2005 Symposium on Applications and the Internet, 2005. Proceedings*. (pp. 183–189). doi:http://dx.doi.org/10.1109/SAINT.2005.4.

Jiang, J.J., & Conrath, D.W. (1997). Semantic similarity based on corpus statistics and lexical taxonomy. arXiv:9709008.

Klusch, M., Fries, B., & Sycara, K. (2006). Automated semantic web service discovery with owls-mx. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems* (pp. 915–922).

Kuang, L., Li, Y., Deng, S., & Wu, Z. (2007). Inverted indexing for composition-oriented service discovery. In *IEEE International Conference on Web Services, 2007. ICWS 2007*. (pp. 257–264).

Li, S., Huang, S.M., Yen, D.C., & Sun, J.C. (2013). Semantic-based transaction model for web service. *Inf. Syst. Front.*, *15*(2), 249–268.

Li, Y., Xiong, J., Liu, X., Zhang, H., & Zhang, P. (2014). Folksonomy-based in-depth annotation of web services. In *2014 IEEE 8th International Symposium on Service Oriented System Engineering (SOSE)* (pp. 243–249).

Liang, Q.A., & Su, S.Y. (2005). AND/OR graph and search algorithm for discovering composite web services. *Int. J. Web. Ser. Res. (IJWSR)*, *2*(4), 48–67.

Lin, M., & Cheung, D.W. (2014). Automatic tagging web services using machine learning techniques. In *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Volume 02* (pp. 258–265).

Ma, J., Zhang, Y., & He, J. (2008). Web services discovery based on latent semantic approach. In *IEEE International Conference on Web Services, 2008. ICWS08*. (pp. 740–747).

Nayak, R., & Lee, B. (2007). Web service discovery with additional semantics and clustering.

Oh, S.C., On, B.W., Larson, E., & Lee, D. (2005). Bf*: Web services discovery and composition as graph search problem. In *The 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service, 2005. EEE 05. Proceedings* (pp. 784–786). doi:10.1109/EEE.2005.41.

Pedrinaci, C., Liu, D., Maleshkova, M., Lambert, D., Kopecky, J., & Domingue, J. (2010). iserve: a linked services publishing platform. In *CEUR workshop proceedings*, Vol. 596.

Peng, Y., & Wu, C. (2010). Automatic semantic web service discovery based on assignment algorithm. In *2010 2nd International Conference on Computer Engineering and Technology*, Vol. 6.

Platzer, C., & Dustdar, S. (2005). A vector space search engine for web services. Third European Conference on Web Services DOI 0-7695-2484-2/05.

Author's personal copy

Rong, W., Peng, B., Ouyang, Y., Liu, K., & Xiong, Z. (2015). Collaborative personal profiling for web service ranking and recommendation. *Inf. Syst. Front.*, *17*(6), 1265–1282.

Sangers, J., Frasincar, F., Hogenboom, F., & Chepegin, V. (2013). Semantic web service discovery using natural language processing techniques. *Expert Syst. Appl.*, *40*(11), 4660–4671.

Shiaa, M.M., Fladmark, J.O., & Thiell, B. (2008). An incremental graph-based approach to automatic service composition. In *IEEE International Conference on Services Computing, 2008. SCC08.* (Vol. 1 pp. 397–404).

Song, H., Cheng, D., Messer, A., & Kalasapur, S. (2007). Web service discovery using general-purpose search engines. In *IEEE International Conference on Web Services, 2007. ICWS 2007.* (pp. 265–271).

Steinmetz, N., Lausen, H., & Brunner, M. (2009). Web service search on large scale. In *Service-Oriented Computing* (pp. 437–444): Springer.

Truong, H.L., & Dustdar, S. (2009). A survey on context aware web service systems. *Int. J. Web Infor. Syst.*, *5*(1), 5–31. doi:10.1108/17440080910947295.

Varguez-Moo, M., Moo-Mena, F., & Uc-Cetina, V. (2013). Use of classification algorithms for semantic web services discovery. *J. Comput.*, *8*(7), 1810–1814.

Wohed, P., van der Aalst, W.M., Dumas, M., & Ter Hofstede, A.H. (2003). Analysis of web services composition languages: The case of bpel4ws. In *Conceptual Modeling-ER 2003* (pp. 200–215): Springer.

Wu, C., & Chang, E. (2008). Searching services on the web- a public web services discovery approach. In *Third Conf. on Signal-Image Technologies and Internet based Systems, IEEE.*

Wu, C., et al. (2008). An empirical approach for semantic web services discovery. In *19th Australian Conference on Software Engineering, 2008. ASWEC 2008*. (pp. 412–421).

Xie, X., Chen, K., & Li, J. (2006). A composition oriented and graph-based service search method. In *The Semantic Web–ASWC 2006* (pp. 530–536): Springer.

Yang, J., & Zhou, X. (2014). Semi-automatic algorithm based on web service classification. In *Advanced Science and Technology Letters* (Vol. 53 pp. 88-91).

Sowmya Kamath S is currently Assistant Professor at the Department of Information Technology, National Institute of Technology Karnataka (NITK), Surathkal. She earned her bachelors and masters degrees from Manipal University and her doctoral degree from NITK Surathkal. Her research work is primarily focused on the role of semantics in Web-based applications like Web services. Her current interests lie in the areas of distributed Web service discovery, large-scale knowledge discovery, machine learning and natural language processing.

Ananthanarayana V. S. is Full Professor at the Department of Information Technology and currently the Dean (Research and Consultancy) at NITK Surathkal. He received his PhD from IISc Bangalore, India, and is a post doctoral fellow of the Memorial University of Newfoundland, Canada. His fields of interest include Web services, distributed computing & multi-database-based mining and he has published extensively in these fields.