DWDS - A SCALABLE, CONTEXT-AWARE FRAMEWORK FOR DISTRIBUTED WEB SERVICE DISCOVERY USING SEMANTICS

THESIS

Submitted in partial fulfilment of the requirements
for the award of the degree of

DOCTOR OF PHILOSOPHY

by

SOWMYA KAMATH S.

DEPARTMENT OF INFORMATION TECHNOLOGY
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
SURATHKAL, MANGALORE - 575 025

JUNE 2016

DECLARATION

by the Ph.D Research Scholar

I hereby declare that the Research Thesis entitled "**$\mathcal{DWDS}$ - A SCALABLE, CONTEXT-AWARE FRAMEWORK FOR $\mathcal{D}$ISTRIBUTED $\mathcal{W}$EB SERVICE $\mathcal{D}$ISCOVERY USING $\mathcal{S}$EMANTICS**", which is being submitted to **National Institute of Technology Karnataka, Surathkal** in partial fulfilment of the requirements for the award of the degree of **Doctor of Philosophy** in **Information Technology** is a bonafide report of the research work carried out by me. The material contained in this Research Thesis has not been submitted to any University or Institution for the award of any degree.

Sowmya Kamath S.
Reg.no. 110653IT11P02
Department of IT,
NITK Surathkal

Place: NITK, Surathkal
Date:

CERTIFICATE

This is to certify that the Research Thesis entitled, "$\mathcal{DWDS}$ **- A SCALABLE, CONTEXT-AWARE FRAMEWORK FOR $\mathcal{D}$ISTRIBUTED $\mathcal{W}$EB SERVICE $\mathcal{D}$ISCOVERY USING $\mathcal{S}$EMANTICS**", submitted by **Sowmya Kamath S (Reg.no.110653IT11P02)**, as the record of research work carried out by her, *is accepted as the Research Thesis* submission in partial fulfilment of the requirements for the award of the degree of **Doctor of Philosophy** in **Information Technology**.

Prof. Ananthanarayana V.S.
Research Guide
(Signature with date and seal)

Prof. G. Ram Mohana Reddy
Chairman - DRPC
(Signature with date and seal)

Dedicated to my family

Acknowledgements

First and foremost, I take this opportunity to express my heartfelt gratitude to God for granting me the wisdom, health and strength to undertake this research work and enabling its successful completion. I express deep gratitude and appreciation to my esteemed research guide, Prof. Ananthanarayana V.S, whose timely advice and constant encouragement has helped shape me as a dedicated researcher. I am very much obliged to him for his valuable guidance and scholarly inputs, always given selflessly, despite his busy schedule. I consider it a great privilege to have been able to pursue my doctoral studies under his guidance and am thankful for the opportunity. I also acknowledge the role of Dr. Prakash S. Raghavendra, in introducing me to the emerging area of Semantic Web research and for encouraging me to pursue doctoral studies.

I sincerely thank Prof. G. Ram Mohana Reddy (Chairman-DRPC & Head, Dept. of IT, NITK), Dr. S.S. Kamath (Dept. of MACS, NITK) and Dr. Annappa B (Dept. of CSE, NITK), for serving as my research committee members and for their valuable comments and suggestions. Special thanks to Dr. Santhi Thilagam for her kind words of motivation and advice. I also acknowledge Dr. Maya Ahmed's role in rekindling the passion for Mathematics in me.

I greatly appreciate the help and support of all my friends and peers during my time as a research student. I am indebted to Dr. Geetha V, for being such a good friend and for always offering her time & inspiration whenever required. I thank my fellow students, Ms. Pushpalata, Dr. Melwyn D'Souza and Dr. Poornalatha G, for the lively discussions on varied topics, research and otherwise. I also acknowledge my dear friends, Madhuri Velaga, Bhagi Chandra, Raksha V. Prabhu and Dr. Aruna Kamath, for their invaluable friendship. I particularly thank all my colleagues and the supporting staff of the Department of Information Technology, who have always provided their help and cooperation as needed, during my research work and beyond.

My family has always believed in my abilities, far more than I give myself credit for. Words cannot express my love and gratitude towards my parents, Mr. S.P.Kamath and Mrs. Vijayalakshmi P. Kamath, for all the sacrifices made on my behalf. A special mention for my siblings, Sushma and Krish, who willingly gave their time whenever I asked for it; be it boosting my morale, turning proofreaders for me, or simply listening to my woes. Their prayers for me and their best wishes are what sustained me thus far on this momentous journey. I also thank my in-laws for their best wishes and blessings. On this occasion, my beloved grandparents are also in my thoughts, who, I sincerely wish were with me in person today, to share this joyous moment.

And finally, I express my deepest appreciation to my husband, Krishnananda, for his encouragement and his whole-hearted support in all my endeavours. He has been a pillar of strength to me while I strived towards my goal. All my success and achievements are dedicated to my daughter, Ananya. For one so young, her love was always given freely and understandingly, during the toughest of times. At the end of a hard day, one look at her smiling face invariably made my biggest problems seem quite trivial.

Sowmya Kamath S

Abstract

Service-oriented Computing is a popular paradigm that lays the foundation for a robust distributed computing infrastructure for both intra- and cross-enterprise application integration and collaboration. It spans a diverse range of possible applications - standalone services, software mashups combining multiple Web services, micro-services used in the implementation of entire IT system landscapes etc. For any such service-oriented application design, the discovery of relevant services that can provide the desired capability is one of the basic tasks. In particular, Distributed Web service discovery is deemed to be one of the grand challenges in Web service research, due to the current distributed nature of services on the Web. This applies in particular to scenarios where a large number of service offers are available in an distributed environment like the WWW; making the process of making a selection difficult, especially given the limitations of keyword based matching. Hence, it is very desirable to improve this process to support efficient Web service discovery, using intelligent, semantics based techniques.

This thesis addresses the important aspects of Web service discovery, focusing on services available in a large scale, distributed environment, that is, the Web. The main research contributions are towards the improvement of service discovery based on implicit semantic information, inference of service domain knowledge, context-aware Web service discovery and composition-oriented Web service discovery. An efficient, scalable framework called $\mathcal{D}$istributed $\mathcal{W}$eb service $\mathcal{D}$iscovery with $\mathcal{S}$emantics ($\mathcal{DWDS}$), is presented, to address the issue of distributed Web service discovery using semantics. The $\mathcal{DWDS}$ framework also provides autonomic features like automated repository management, redundancy control and verification, in order to maintain the validity of the service repository. Based on the statistics collected over the course of 3 years, it was found that the techniques proposed for developing the $\mathcal{DWDS}$ framework and adding autonomic features were effective and supported scalable service discovery.

To support similarity based service discovery and effective categorisation of services in the repository, automatic metadata generation and similarity computation techniques, that use the inherent functional semantics of the services were incorporated.

As traditional unsupervised categorisation mechanisms like K-means clustering and supervised techniques like Classification cannot deal with the dynamic nature of the framework, a bio-inspired incremental clustering algorithm, BI^2C, based on the flocking behaviour of birds was proposed. BI^2C incrementally clusters the service collection after each change introduced by periodic crawler runs, thus, supporting the scalability of the service repository. BI^2C achieved an average speed up of more than 57% over traditional clustering algorithms and was also able to deal with the large volume of service descriptions available in the $\mathcal{DWDS}$ repository.

To enable context-aware, natural language based querying during Web service discovery in $\mathcal{DWDS}$, semantics based query analysis techniques were developed. Also, any complex queries were automatically processed to determine their constituent sub-queries, so that composite service discovery can be supported. It was found that the semantic analysis of user query to capture context were effective as it resulted in 16% improvement in precision and over 37% increase in recall, over keyword matching based approach.

To extend composite service discovery, a concept of capturing the service input/output dependencies formally through a Service Interface Graph (SIG) was proposed. In the SIG, services are represented as nodes, and their dependencies are captured as edges. Any user queries that have sub-queries are answered by traversing the SIG, so that the correct invocation sequence required to satisfy the user query requirements can be identified. Experimental evaluation showed that the proposed method achieved an accuracy of 70.68% and effectively identified correct composition templates in $\mathcal{O}(N^2)$ time.

To summarize our contributions, our work focuses on developing a semantics based distributed Web service discovery framework that can automatically retrieve service descriptions available in heterogeneous sources on the Web, to build a scalable service repository. Automated metadata generation and dynamic categorisation techniques enable the framework to support efficient basic and composite Web service discovery in a context-aware manner.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

ABC	Artificial Bee Colony Algorithm
ACO	Ant Colony Optimization
ANN	Artificial Neural Network
API	Application Program Interface
B2B	Business to Business
B2C	Business to Customer
BFS	Breadth First Search
BI^2C	Bird Inspired Incremental Clustering Algorithm
BPEL	Business Process Execution Language
CoS	Cost of Service
CT	Composite Service Templates
DAG	Directed Acyclic Graph
DFS	Depth First Search
DHT	Distributed Hashing Tables
DoM	Degree of Match
DOM	Document Object Model
$\mathcal{DWDS}$	Distributed Web service Discovery with Semantics
DWSRIE	Distributed Web Service Retrieval and Indexing Engine
EAI	Enterprise Application Integration
ebXML	Enterprise Business eXtensible Markup Language
FCA	Formal Concept Analysis
FIPA	Foundation of Intelligent Physical Agents
GA	Genetic Algorithm
HAC	Hierarchical Agglomerative Clustering
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure

IR	Information Retrieval
IOPE	Input Output Precondition Effects
IT	Information Technology
JADE	Java Agent Development Framework
LDA	Latent Dirichlet Allocation
LSI	Latent Semantic Indexing
MIME	Multi-purpose Internet Mail Extensions
MSF	Multi-species Flocking Model
NAICS	North American Industry Classification System
NGD	Normalized Google Distance
NLI	Natural Language Interfaces
NLP	Natural Language Processing
NLTK	Natural Language Tool Kit (Python)
OWL	Web Ontology Language
OWL-S	Web Ontology Language for Services
P2P	Peer to Peer
PLSI	Probabilistic Latent Semantic Indexing
POS	Parts of Speech
PSO	Particle Swarm Optimisation
QoE	Quality of Experience
QoS	Quality of Service
QT	Quality Threshold
RDF	Resource Description Format
SAWSDL	Semantic Annotations for WSDL and XML Schema
SIC	Standard Industrial Classification
SIG	Service Interface Graph
SKOS	Simple Knowledge Organization System
SOA	Service-oriented Architecture
SOAP	Simple Object Access Protocol
SOBL	Semantic Object Behavior Language

SOC	Service-oriented Computing
SOE	Service Oriented Enterprises
SSC	Specialized Service Crawler
SVD	Singular Value Decomposition
SWS	Semantic Web Service
TAO	Transitioning Applications to Ontologies
TTA	Tree Traversing Ant
UBR	Universal Business Registries
UDDI	Universal Descriptions, Discovery and Integration
UNSPSC	United Nations Standard Products and Service Codes
URI	Uniform Resource Identifier
WSD	Word Sense Disambiguation
WSDL	Web Service Definition Language
WSFL	Web Services Flow Language
WWW	World Wide Web
W3C	World Wide Web Consortium
WSMO	Web Service Modeling Ontology
XML	eXtensible Markup Language

Nomenclature

Notations used in Chapter 4

p	a Webpage considered for binary categorisation during WSDL existence check
b	branching factor of a particular Webpage
d	depth of crawling when BFS is used

Notations used in Chapter 5

r	rank of a word in a dictionary of N words
N	number of distinct words in a dictionary
tf	term frequency of a tag candidate t in a WSDL document d
tf_{max}	highest frequency of any term in d
df	frequency of occurrence of the term t in other WSDL documents
D	number of WSDL documents in the dataset.
U	tf-idf matrix
V	semantic-relatedness matrix
K	a particular domain considered for SVD generation
$sim(t_i, t_j)$	similarity between any two terms t_i and t_j
s_i	some service s_i in the dataset
f_{s_i}	Feature vector of of service s_i
$sim_{oo}(s_i, s_j)$	similarity between the operations of s_i and s_j
$sim_{o\ i/o\ type}(o_i, o_j)$	similarity between the input/output messages of s_i and s_j
$sim_{wsdl}(s_i, s_j)$	total similarity between the WSDLs of two services s_i and s_j
$sim_{cluster}(c_i, c_j)$	similarity of a pair of clusters for merging for HAC
t_{merge}	computed threshold for merging two clusters for HAC
$sim_{tag}(g, s)$	how well a tag g represents service s
$csim_{service}(s, c)$	how well the given service s represents its cluster c

Notations used in Chapter 6

$\overrightarrow{v_{al}}$	Alignment Velocity
$\overrightarrow{v_{ch}}$	Cohesion Velocity
$\overrightarrow{v_{sp}}$	Separation Velocity
$\overrightarrow{v_{sim}}$	Attraction Velocity
$\overrightarrow{v_{dsim}}$	Repulsion Velocity
a_c	agent being considered currently.
n_i	i^{th} neighbour of a_c
$\overrightarrow{v_c}$	velocity of the current agent a_c
$\overrightarrow{v_i}$	velocity of n_i, within the defined locality of a_c
p_i	position of n_i in virtual space
p_c	position of a_c in virtual space
$sim(n_i, a_c)$	pair-wise similarity between agents a_c & n_i
$d(p_i, p_c)$	Euclidean distance between p_i and p_c in virtual space
S_r	Sensor Range for nearest neighbour search
SF	Speedup Factor
t_{HAC}	Cluster time of HAC Algorithm
t_{BI^2C}	Cluster time of BI^2C Algorithm
$JM(C_i, C_j)$	Jaccard Measure value for two clusters C_i and C_j
$DB\text{-}Index(C_i, C_j)$	DB Index value for two clusters C_i and C_j
$S_k(C_i)$	average distance of all services in a cluster to their cluster centre
$S(C_i, C_j)$	distance between two different cluster centres C_i and C_j

Notations used in Chapter 7

/CC	Part-of-speech tag for coordinating conjunctions
$\cos\theta_{c_j,q_t}$	Cosine similarity between cluster c_j and query terms q_t
$\cos\theta_{s_k,q_t}$	Cosine similarity between service s_k and query terms q_t
$w(s_k, q_t)$	relative weight of a term with respect to s_k and q_t.
$relR$	Relative Recall
F_β-*measure*	F-measure value at various values of β
F_1-*measure*	Balanced F-measure ($\beta = 1$)
$F_{0.5}$-*measure*	Precision-oriented F-measure ($\beta = 0.5$)
F_2-*measure*	Recall-oriented F-measure ($\beta = 2$)

Notations used in Chapter 8

S_i	a service node in the Service Interface Graph
$\vec{V}_{out_{S_i}}$	Output vector of service S_i
$\vec{V}_{in_{S_k}}$	Input vector of service S_k
$cos\text{-}sim(\vec{V}_{out_{S_i}}, \vec{V}_{in_{S_k}})$	Cos-sim of output vector of S_i and input vector of S_k
$CT_{correct}$	Correct composite service templates generated
$CT_{incorrect}$	Incorrect composite service templates generated
$CT_{partial}$	Partial composite service templates generated
$CT_{conflict}$	Composite service templates with I/O conflicts
CT_{total}	Total composite service templates generated

PART I

Introduction & Background

Chapter 1

Introduction

With over a trillion pages and billions of users, the Web is undoubtedly one of the most popular technological inventions in history. It has currently grown into the most important medium for information exchange and already is one of the core environments for business communications & social interactions. While initially intended for enabling information sharing among scientists, the Web has since then evolved to cater to governments, businesses, and individuals to make their data and applications easily accessible. The original design of the Web served its purpose and went beyond anticipated predictions in its popularity. However, as the number of applications available and the volume of data on the Web saw an exponential increase, it become apparent that the Web could no longer sustain its growth in its present form.

Currently, the majority of data on the Web is '*understandable*' to humans or to custom developed applications, thus rendering the current Web mostly '*syntactic*'. Much of the effort in actually reading and understanding content available on the Web is delegated to humans, while, computers, with their immense processing power are limited to mostly displaying information as instructed. So far, the main impediment to automated machine processing has been the limited availability of semantics, which can enable machines to '*understand*' Web data. The *Semantic Web* (Berners-Lee et al. 2001) is an emerging paradigm shift towards the Next Generation Web, that aims to overcome this limitation. Berners Lee et al, in this seminal article of 2001, defined the Semantic Web as -

> *"...an extension of the existing Web, in which information is given a well-defined meaning....a new form of Web content that is meaningful to computers, that will unleash a revolution of new possibilities"* (Berners-Lee et al. 2001).

To this end, the ultimate goal of a Semantic Web is, to transform the Web as we know today into a medium through which data and applications can be automatically understood and processed.

The real power of the Semantic Web will be realized when, automated programs that can collect Web content from diverse sources, process the information automatically and exchange the results with other programs are available (Hendler 2001). The effectiveness of such software programs will increase exponentially as more machine-readable Web content and automated services become available. Even programs that were not expressly designed to work together can transfer data among themselves when the data is enriched with semantics. In view of this ultimate goal, the development of technologies for supporting the envisioned Semantic Web has been the priority of various research communities (e.g., database, artificial intelligence, intelligent agents). As of today, several technologies like the Resource Description Framework (RDF), and the Web Ontology Language (OWL) have been standardized and are available to support the basic functionalities of the Semantic Web. This effort is the focus of numerous research projects, with the aim to integrate semantic technologies in various fields, like, knowledge management, business intelligence, Web 2.0 and service-oriented computing (Benjamins 2008).

Service-oriented Computing (SOC) and Web services will be integral in bringing together distributed applications that consume the semantic data available on the Web (Hendler 2001; McIlraith et al. 2001). Berners-Lee et al. (2001) put forth the idea of '*software agents*' or '*services*' that could work together in order to collect and process (semantic) data. Discussing the potential benefits of such autonomous agents, they also noted that -

> *"In a process called 'service discovery', it would be necessary to describe a service's functionalities semantically by a common language in a way that lets other agents 'understand' both the function offered and how to take advantage of it".*

McIlraith et al. (2001) illustrated why adding semantics to Web services is a fundamental need for achieving automation over the Web, as envisioned by the Semantic Web. They coined the term, *'semantic Web services'*, to describe such a class of intelligent services; whose functionalities, interfaces, and effects are described in an unambiguous, machine-understandable form. As Web service standards are based on XML[1], which is one of the fundamental building blocks of the Semantic Web, it would be a natural progression to combine Web service technologies & Semantic Web technologies to transition towards semantic Web services (SWS).

In the area of Web service research, the inclusion of semantics in various tasks in the Web service life cycle is often seen as one of the grand challenges that need to be met

[1]eXtensible Markup Language

in order to facilitate the success of Web services on a broad scale (Krummenacher et al. 2005; Martin et al. 2007; Papazoglou 2003). The W3C is currently actively engaged in the process of semanticising Web service standards to facilitate service foundations, service management, and service engineering. As background information, we briefly discuss some important concepts and technologies in the field of Web services, in Section 1.1. In Section 1.2, a discussion on the need for semantics in Web Services and related issues is presented.

1.1 Web Services

Service-oriented Computing (SOC) (Papazoglou 2003) is one of the enabling technologies for distributed computing and is expected to be a major player in enabling the Semantic Web (Hendler 2001). The core idea in SOC in the concept of a *service oriented architecture (SOA)*, which is composed of services. A service is an abstracted unit of some system functionality, which can be called, consumed and reused without needing to know the implementation details of the underlying system. Since services may be offered over specific protocols and can communicate over the Internet, they provide a distributed computing infrastructure for both intra- and cross-enterprise application integration and collaboration.

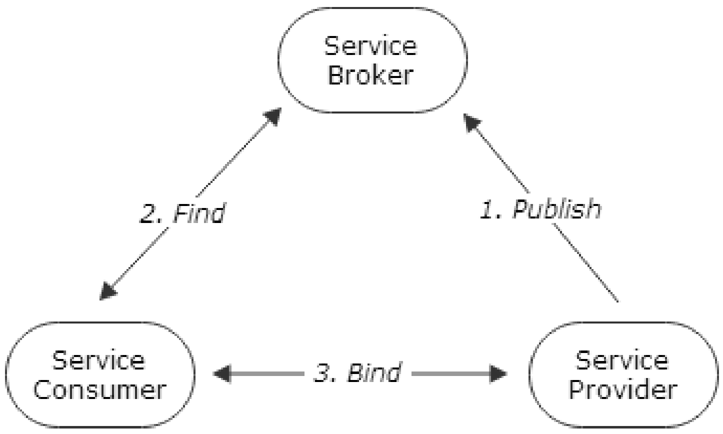


Figure 1.1: The SOA Publish-Find-Bind Model (adapted from Gottschalk et al. (2002))

The basic model that describes the actors and the actions in a SOA scenario is the publish-find-bind model (Gottschalk et al. 2002), shown in Figure 1.1. The model consists of three different *actors*, each with its own associated *operations* and *actions*.

1. The *Service Provider*, which are a software program with a known interface, offers one or more services. These could be published in a *service registry*, which is often hosted by a service broker.

2. The *Service Consumer* (or Service Requester/Service Client) searches for required services and consumes selected services. If the service to be used is known, the service consumer directly binds the service. If not, a service broker may aid in the process of finding the service, after which the binding is performed and service is invoked.

3. The *Service Broker* facilitates service management and can have different roles. In most scenarios, the broker provides a service registry where services can be published and found.

In the model shown in Figure 1.1, the service consumer can either be a human or a software client. The service provider can be a human, organization or a software agent. The service broker or service registry is a necessary component in order to publish, find and consume distributed services based on the business process. Hence, effective description of the service's functionalities is important for finding or discovering services for binding. This is where Web service standards come into play. Following the SOA principles, a Web service is a set of related functionalities that can be programmatically accessed over the Web. To formalize the definition, we use a more refined definition by the W3C Web Service Activity Group, which defines a Web service as -

> *"a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artefacts. A Web service supports direct interactions with other software agents using XML based messages exchanged via Web-based protocols"* (Austin et al. 2002).

This stresses the role of XML in the Web service standards. A Web service primarily binds to Web protocols like HTTP[2] or HTTPS[3] and uses the HTTP primitives of request/response methods for data transfer between consumer and service endpoint or between two service endpoints. The service interface hides the implementation details, allowing it to be used independently of the platform where it is deployed. As services are loosely coupled and self-contained, it is possible to dynamically invoke and substitute services, for e.g., in a business process, or across the borders of a single company or organization (Gottschalk et al. 2002).

Several technologies form the core of the Web service technology stack, all of which are primarily based on XML. Figure 1.2 extends the minimalist SOA model in Figure 1.1 to illustrate the Web services conceptual architecture. XML is used for structuring the data to be transferred, SOAP[4] for transferring data between remote systems, and WSDL[5] for

[2]HyperText Transfer Protocol
[3]HyperText Transfer Protocol Secure
[4]Simple Object Access Protocol
[5]Web Service Description Language

describing the capabilities of the service. A service is invoked by sending SOAP messages to the service endpoints on the remote system. Another optional component of the SOA triangle (Michlmayr et al. 2007) is the UDDI[6], that plays the role of a service broker in a service discovery scenario over the Web. It acts as a *business registry* and provides a directory service for locating required services and to support dynamic binding, by allowing service consumers to query the registry. Companies like Microsoft, SAP and IBM spearheaded the movement for an UDDI Universal Business Registry (UBR), to provide world-wide free publishing and querying Web services.

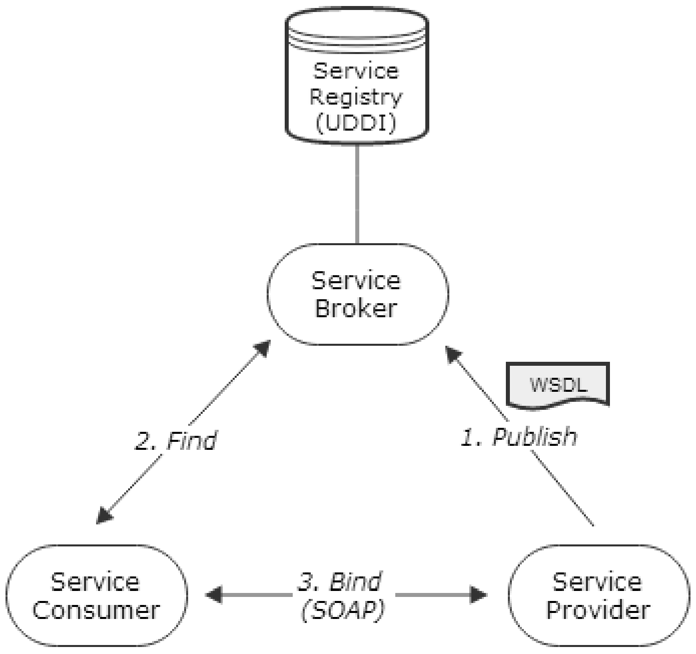


Figure 1.2: Web Services Conceptual Architecture (extended from Fig. 1.1)

A Web service can be used in an application by itself (e.g., currency conversion service), or with other services, to carry out a complex aggregation or a business workflow. Services belonging to the first category are called *basic* or *atomic* services, as their implementation is self-contained and does not invoke any other services. Services in the second category are linked together in a workflow, in a pre-defined sequence, and are called as *complex* or *composite* service. Here, each constituent service's execution invokes other services to perform the next sub-task. Composite Web services are more common and are used in various domains like government (e.g., tax payment, automated document processing), in e-commerce (e.g., order processing and logistics), B2B scenarios etc.

[6]Universal Description, Discovery and Integration

1.2 Semantics for Web Services

The term 'semantics' is defined as the *meaning* of information, as opposed to syntax, which defines the *structure* of data. In a Web service, *syntax* is represented by the structure of its service description (WSDL) and *semantics* refers to something extra about this syntactic data, i.e., *metadata.* Sheth et al. (2006) identified different levels of semantics for Web services - semantics for data, functional/non-functional parameters and execution.

i. *Data semantics* - a formal definition of the data that is referenced in the input and output messages of the Web service.

ii. *Functional semantics* - a formal definition of the functionalities of the Web service, by annotating standard vocabulary concepts to the operations, preconditions and effects of a service, as per its domain.

iii. *Non-functional semantics* – used to formally define any references to service quality like Quality of Service (QoS) and other user given constraints.

iv. *Execution semantics* - a formal definition of the process of service execution and actions associated with this execution.

To achieve machine-processability of service content, this metadata has to be based on well-defined, standardized vocabularies like ontologies and concept taxonomies (Cardoso et al. 2006). Currently, three ontology based approaches are available for generating semantic annotations for Web services - OWL-S (Web Ontology Language for Services), SAWSDL (Semantic Annotations for WSDL and XML Schema) and WSMO (Web Service Modeling Ontology). These are formalisms for annotating different parts of a service description based on semantic technologies.

1.2.1 Need for Semantics

A Web service based application consists of several distinct phases depending on the activities performed during its operation. This is referred to as the Web Service Life cycle, and the activities performed during the various phases can be categorized as *design-time* activities, *build-time* activities and *deployment/run-time* activities (Akkiraju 2007).

- *Design-time activities* - service modelling and creation activities.
- *Build-time activities* - publication of the service advertisement, service discovery, selection and composition related activities.

- *Deployment/Run-time activities* - service binding, service invocation, execution monitoring.

For enabling complete automation, which is currently an idealistic scenario, each of these phases and activities must be powered by explicit semantics. This could be potentially achieved by annotating the appropriate parts of the Web service with concepts from a richer model to unambiguously determine its domain. During service creation, the service provider must also incorporate domain modelling while choosing datatype terms, element names and documentation. The service capabilities must be described in domain-specific natural language terms as per the identified semantic model. During service advertisement, the service must be placed in the most relevant registry category as per its identified domain. For service discovery, reasoning techniques need to be used to find the semantic similarity between the service description and the request.

Further, in some cases where no direct matches are found, the functionality of multiple services may need to be aggregated and composed. During composition, the functional aspect of the annotations may need to be used to aggregate the functionality of multiple services to create useful service compositions, while the semantics of non-functional aspects of services can help in determining whether these compositions are valid. During invocation, semantics can be used to represent data transformations. In case of failure of a service during run time, semantics can enable automatic service discovery and binding to find suitable alternate services. Therefore, once represented, semantics can be leveraged by software programs to automate service discovery, mediation, composition, execution and monitoring. We discuss an example scenario to further stress the need for semantics in Web service life cycle tasks in Section 1.2.2.

1.2.2 A Motivating Example

We consider a tourism related scenario to illustrate the importance of semantics in Web services domain. This scenario will be treated as a running example throughout this thesis. Consider that Alice and Bob want to take a vacation and they hire a travel agent to take care of trip booking. Alice describes their needs (where they want to go and when), the restrictions (budget, number of days), and some personal preferences (hotel with a pool, preferred airlines). It is the job of the travel agent to use various information repositories (flight schedules, hotel bookings, destination guides) to plan the trip to suit Alice and Bob's specifications and take their consent before booking the trip.

Consider that, the above scenario is to be performed using a virtual travel agent. Figure 1.3 depicts the process of enacting the travel scenario using such a virtual travel agent along with other Web services. The main actors in this scenario are - the tourist, the virtual travel agent service, the service providers of the various travel services as requested by the tourist (airlines, hotel, activities, cab) and other additional services (payment services, airport pickup, local sightseeing services). The tourists are required to provide their requirements in a formal manner to the virtual travel agent service. Assuming that all needed services are available in a central registry, the work of the travel agent service then, is to effectively find the services for each of the consumer requirements, in accordance to the additional restrictions imposed by the user. In the idealistic scenario discussed, all these services would be semantically annotated to support unambiguous discovery and selection. For instance, the different airline companies may use varied terms in their service descriptions (e.g., departure, destination, airfare, baggage), which are all explicitly defined as belonging to the air travel domain. Also, the travel agent service is aware of the vocabulary used in the various domains, and using this shared knowledge, it can communicate effectively with the different services to perform its duties.

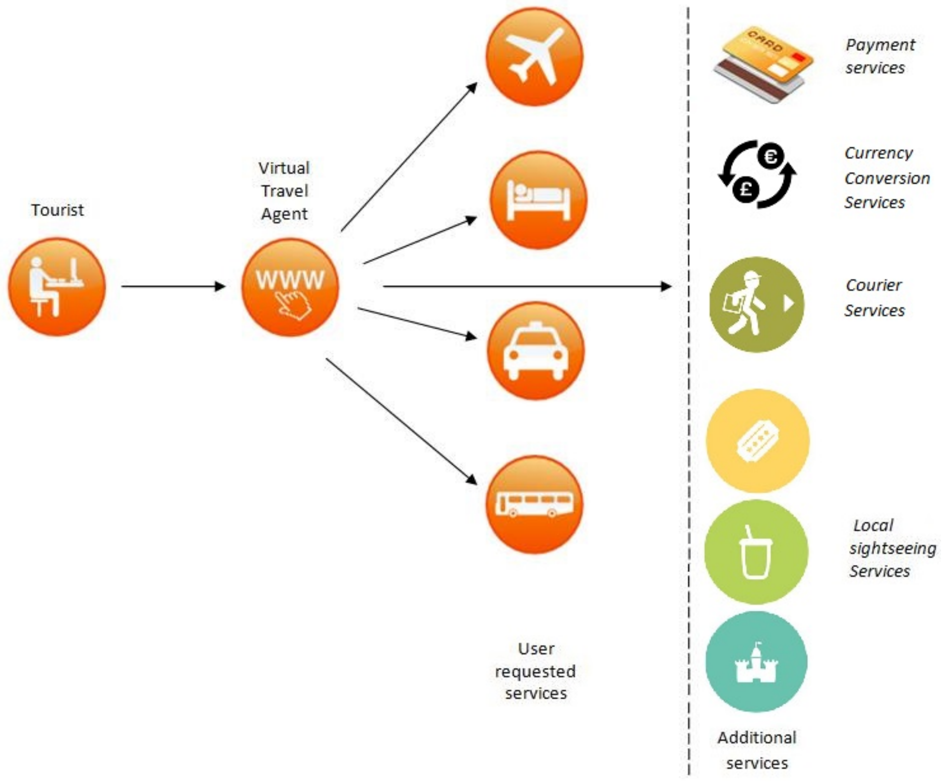


Figure 1.3: A Motivating Example - Travel scenario

However, the current situation is far from the idealistic one. Very few services currently on the Web have explicitly defined, standard semantics that clearly identify its capabilities and domain. As per published studies (Fan et al. 2005; Kim et al. 2004; Li, Zhang, et al. 2007), there are more than 20,000 services on the Web, most with WSDL based service descriptions, without any associated semantics. In UDDI registries and also in service portals, the services are manually categorized by service providers, which compounds the problem due to the scope for misclassification, making matching that much more difficult. Some problems that may arise due to lack of semantics are listed below and discussed on a case by case basis.

(i) *Syntactically similar but semantically different services.*

In the absence of explicit semantics, two service descriptions that may seem to have similar datatypes and structure, but may actually be dissimilar when their functional semantics is considered. For example, consider two services - a *getSeatAvailability* service that takes the flight number (xsd:string) and the date of travel (xsd:date) as input and returns the number of available seats (xsd:integer). The other, a *flightAvailability* service provides parameters such as, the date of travel (xsd:date), sourceCity (xsd:string), destinationCity (xsd:string), and returns the flight number (xsd:string). Syntactically, that is, just considering the datatype of the input and output parameters in this example, the interfaces and also the service names may seem to be a reasonable match, but their intended functionality is quite different. A syntactic matching engine could still consider this a potential match — if not a good one. But clearly, these two services are meant to perform very different functionalities and this could lead to false positives during matching.

(ii) *Syntactically different but semantically similar services.*

In contrast to the first case, two syntactically dissimilar services may be intended for similar functionalities, semantically. For example, two services, both meant for hotel amenity checks may have completely different syntactic structures. The service *getHotelFacilities* takes the hotel name as input in string form (containing the city name, state name, country name in an XML document represented as xsd:string) and returns the list of available facilities as a xsd:string. Another service, *verifyPoolAvailability* exposes the individual parameters separately instead of putting them in an XML document. Structurally these two services look different even though they perform similar functions of checking the available amenities at a hotel. Thus, a syntactic matching engine may not consider these two services as good matches, leading to false negatives. A semantic matchmaking engine based on lexical and term similarity may find the services to be a partial match depending on the similarity of the service terms.

(iii) *Syntactically different and seemingly semantically different services.*

When designing services for specific purposes, different service providers may choose different terminologies to express the same concepts. For example, one service provider may refer to the confirmation number sent to a customer as *bookingId*, where as a train booking service may call it *PNRnumber* (PNR - Passenger Name Record; indicates the current booking status of a passenger). In the absence of domain information indicating that PNR is a `subClassOf` bookingId, specifically related to train bookings in India and that, both are different ways of uniquely identifying a ticket booking, these two terms may be considered as unrelated terms, potentially leading to false negatives. These kinds of superficial differences may stand in the way of identifying suitable services if domain semantics are not explicitly specified.

(iv) *Syntactically similar and seemingly semantically similar services.*

In some scenarios, the interfaces of services may look similar at first glance, both syntactically and semantically, but may actually represent very different things. To illustrate this, consider two services: *getBookQuoteAmount* and *checkBookPrice.* Both services take a parameter bookID (xsd:string) and return an (xsd:float) and the names of the parameters are similar as well. However, these services perform completely different functions. The *getBookQuoteAmount* service from the travel domain that takes the unique ID of a customer's travel itinerary as input and returns the total amount charged to the customer. The *checkBookPrice* service is a publishing related service that returns the price of a book given its unique ID (ISBN). If a syntactic matching is performed on these two services based on the number of parameters and datatypes, they could incorrectly be considered a match because of structural similarity. In case of semantic matching based on lexical and name-similarity, without considering their domain and context, these could still be considered a partial match, thus resulting in false positives.

Table 1.1: Potential Issues due to Service/Datatype Naming and Interface Structure during Web Service Discovery

		Interface Structure	
		Similar	*Different*
Service/ Datatype Naming	*Similar*	False Positives *(due to inadequate domain context)* e.g. case (iv)	False Positives *(due to lexical matching)* e.g. case (i)
	Different	False Negatives *(due to lack of domain knowledge)* e.g. case (ii)	False Negatives *(due to syntactic matching)* e.g. case (iii)

In summary, there is a possibility for falsely identifying some services as matches when they are not (false positives as in cases (i) and (iv)); and incorrectly identifying some services as poor or no match when they could indeed be a good match (false negatives (as in cases (ii) and (iii)). Table 1.1 presents a summary of the various scenarios discussed. The vision of semantic Web services is to enable better automatic matching of services based on semantic context, so that the cases of false positives and negatives can be minimized as much as possible. However, before this vision can be a reality, there are several challenges that have to be addressed, which are discussed in Section 1.2.3.

1.2.3 Major Challenges

As highlighted earlier, there are several major hurdles that are to be overcome before the ideal scenario of automated, semantics based Web service task execution can be realized. Adding semantics to Web service descriptions is a time consuming task especially since automated tool support for large-scale annotation is not yet available. A majority of the current approaches (Nayak et al. 2007; Sangers et al. 2013; Srinivasan et al. 2006) that aim to improve service tasks like discovery and selection, were applied to small datasets of semantic Web service descriptions (i.e., OWL-S, SAWSDL or WSMO) to test their techniques. However, these approaches have several limitations.

First and foremost, it is impractical to expect all new services to have semantic descriptions (Paliwal 2012). Secondly, descriptions of the vast majority of already existing Web Services are specified using Web Services Description Language (WSDL) and do not have associated semantics (Crasso et al. 2008b). Hence, the greatest challenge facing the Web service community today is the monumental task of adding semantics to already published services. This problem is compounded by the fact that, the number of services published on the Web is quite large (more than 20,000 as per certain published estimates (Fan et al. 2005; Li, Zhang, et al. 2007)). The task of annotating these services with their domain model would fall to the service providers, whose motivation for doing so is very low, due to the manual effort required (Paliwal 2006). The time and human effort involved in manually or semi-automatically annotating service interfaces is quite massive, with currently available tools (Farrag et al. 2013). Furthermore, lightweight and standard, service specific ontologies that can be used to annotate service interfaces and the tools to do this automatically on a large-scale, are not yet available (Plebani et al. 2009).

The conceptual Web service architecture discussed in Section 1.1 offers limited support for semantics based service discovery, matchmaking, selection and composition (as seen from the travel scenario example discussed). The solution to the challenges identified here, requires an augmentation of the 'traditional' Web service architecture, by incorporating effective mechanisms for optimizing service-specific processes. In particular, the Web service life cycle tasks of service discovery and composition, are particularly challenging and have generated a great deal of interest in the research community. A key requirement for a successful Web service based application is that, potential users/client are able to discover and use it. If interaction needs to be accomplished in an automated way, other applications should be able to find the right service and obtain the information necessary to interact with it. Semantics can play an important role in accomplishing discovery in these settings. In this thesis, we concentrate on the problem of Web service discovery and examine the critical issues and research challenges, in Section 1.3.

1.3 Web Service Discovery

In this thesis, we focus on the problem of semantics based automated Web Service discovery. Automatically discovering services is the process of finding a service that matches a given set of requirements. These requirements may be - *functional* (Crasso et al. 2008b; Plebani et al. 2009; Rajasekaran et al. 2005) or *non-functional* (Birukou et al. 2007; Kuang 2012; Ran 2003). Services may be discovered from a *central repository* (Kourtesis et al. 2008; Sivashanmugam et al. 2004; Yu et al. 2012)) or in an *distributed* manner (Dong et al. 2004; Schmidt et al. 2004; Wang 2006). Service matching could be *syntactic using keywords* (Song et al. 2007; Wu and Chang 2007), using *structure matching* (Kokash et al. 2007; Stroulia et al. 2005) or *interface matching* (Hao et al. 2007; Wang and Stroulia 2003) and/or based on semantic matching, like *lexical similarity* (Elgazzar et al. 2010; Fang et al. 2012; Sangers et al. 2013) and *ontology matching* (Nayak et al. 2007; Paolucci et al. 2002; Segev et al. 2012).

In the travel scenario example discussed in section 1.2.2, two business models can be identified - the scenario were the tourist interacts with the virtual travel agent service is a business-to-consumer (B2C) model, where finding a service provider that can offer a service whose capabilities match the specified requirements is required. For example, this may be finding an airline ticket booking Web service given a source, destination and payment method *(atomic service)*. The interaction between the virtual travel agent service and the other service providers (airlines, hotel, cab, sightseeing etc) is a business-to-business (B2B) setting, where much more complex tasks of integrating multiple services with additional constraints may be required, thus requiring composition of services *(composite service)*. We discuss example Web service discovery scenarios that can be served by basic and composite services in the next section.

1.3.1 Basic and Composite Services

In the traditional mode of service discovery as per the conceptual Web service architecture (as shown in Figure 1.2), the service provider uses the UDDI Publish API to register and advertise their service in the service broker. The advertisement is listed in one of the standard categories available in the UDDI. This means that the listing is dependent on the service provider's discretion (and limited) knowledge of the UDDI classification scheme (manual categorization) (Akkiraju, Goodwin, et al. 2003; ShaikhAli et al. 2003). A service consumer interested to find certain services, can use the UDDI's Inquiry API to submit a keyword query to the registry, and all services matching those specific keywords are returned as a ranked list. Of these, the service consumer can choose a service and request for its endpoint information from the service broker, using which a bind action can be performed. This is an example for *Basic* or *Atomic Service Discovery*, and the query that is submitted to the system is called a *simple query*.

Composite service discovery is the process of determining several basic services that can be invoked in sequence to satisfy some complex task (Benatallah, Dumas, et al. 2002). A composite process explicitly defines which component services should be invoked, in what order and how to handle any exceptional situations that may arise during execution (Alonso et al. 2004). From application development perspective, supporting composite service discovery can help business application developers in minimizing the amount of work required to develop applications, ensuring a rapid deployment of designed SOA systems.

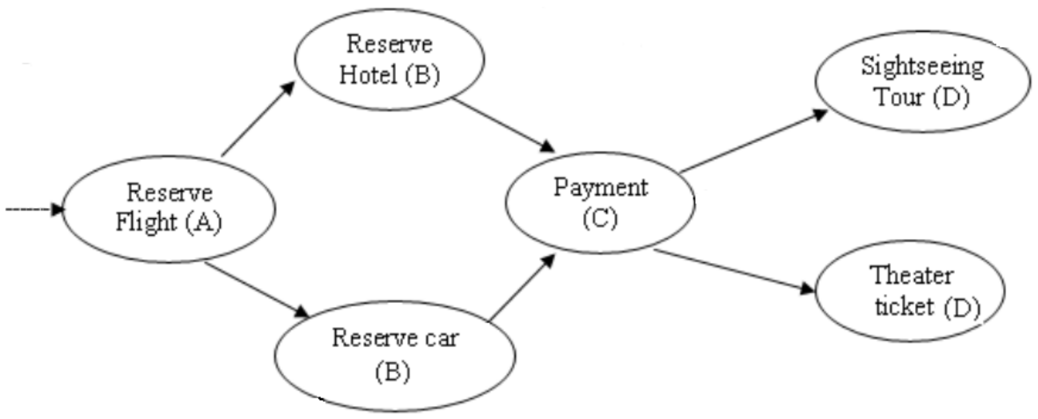


Figure 1.4: A composite service workflow

Figure 1.4 depicts a common scenario in real-world travel booking example discussed in Section 1.2.2, where individual booking services are performed by travel agents based on customer preferences. In the e-travel booking scenario, the customer request has to be carried out in a well-defined sequence, for it to be correct and valid. The requirements are to first book the flight; if it is available and booked, only then the hotel booking and the car rental services are to be invoked, and finally payment service for completing the transaction. These tasks have to be carried out, in a pre-defined order, to prevent undesirable effects and financial loss to the customer.

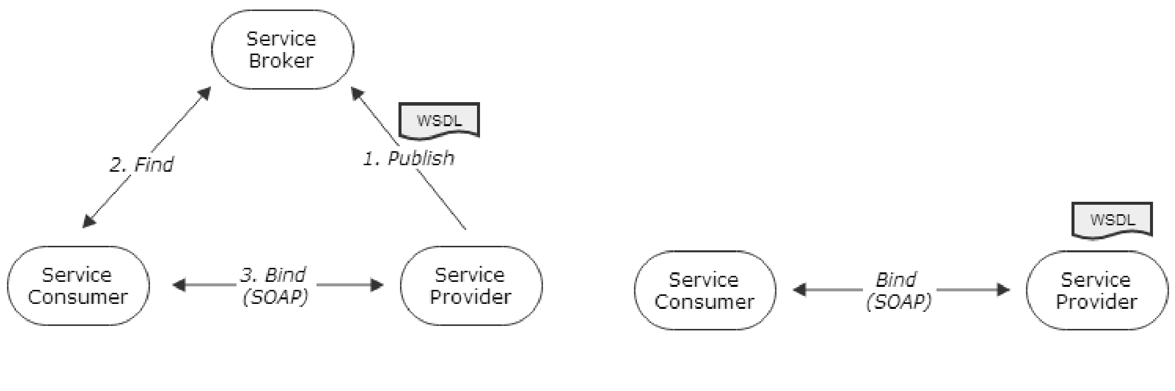
In order to automate such a process in a SOA system, the main requirement would be semantically defined services that can be invoked in order as per the composite workflow definition, in accordance with the user directives. Critical to such a system, is context sensitive processing of the user's request, to determine both the constituent services and also the correct invocation sequence.

As most business processes are composite services, 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 required by their complex requirement, are incorporated. Due to increasing number of published web services in e-business domain, the demand for automated techniques to identify composite services for fast application development has also increased.

Most existing works in the area of web service discovery currently, deal with the problem of identifying relevant atomic services for a given user requirement. Researchers have used various approaches – keyword based search (Song et al. 2007), service broker based approaches (D'Mello 2008; Al-masri et al. 2009), service tagging based techniques (Fang et al. 2012; Li, Xiong, et al. 2014; Lin et al. 2014), service classification (Corella et al. 2006c; Varguez-Moo et al. 2013), semantics based search (Chan et al. 2011; Wu, Chang, and Aitken 2008), domain ontology based matching (Benatallah, Hacid, et al. 2005; Nayak et al. 2007), non-functional parameters based selection (Almasri et al. 2007; Alrifai et al. 2010; Xu 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 to be composed in a valid sequence to provide the intended functionality. Automated techniques to identify services that can be coupled together to provide the requested functionality, especially when the potential candidates are too many, would be very beneficial to application designers. However, the current scenario is not very conducive to such automated systems, as existing standards-based approaches do not offer support for composition-oriented service discovery. We discuss the limitations of existing frameworks in Section 1.3.2.

1.3.2 Current Scenario

Current Web service registries, especially the two standardized efforts, UDDI (mostly B2C scenarios) and ebXML[7] (for B2B environments) suffer from several shortcomings. The UDDI offers users a unified and systematic way to find service providers through a centralized registry of services, that is often termed as an automated online 'directory' for Web services (Curbera et al. 2002). ebXML envisioned the creation of a single global electronic marketplace where businesses can find each other, agree to become trading partners, and conduct business (Hofreiter et al. 2002). Both have seen significant standardization efforts and have been recognized as the standards for web service registry implementations. However, in practice, ebXML is in use only in certain highly specialized domains like supply chain management and logistics, while the UDDI is rarely used any more. The Universal Business Registries (UBRs) maintained by IBM, Microsoft, and SAP failed to gain adoption and were shut down in 2006 after only 5 years in operation (Krill 2005). Some reasons for the failure of the UBRs were lack of active maintenance by many service providers, keyword based matching approach, manual classification, limited support for expressive queries and for composite services (Al-Masri et al. 2007a).



(a) SOA in theory (b) SOA in practice

Figure 1.5: Basic SOA Model - Theory vs. Practice (adapted from Michlmayr et al. (2007))

The main aim of the service broker in the SOA triangle is to support the publish-find-bind architecture in a truly distributed way. However, in practice, a major share of today's SOA applications have increasingly adopted an abridged model (Michlmayr et al. 2007) consisting of the service provider and service consumers (as shown in Figure 1.5). This is possible by providing the exact service endpoint information required to invoke the service to their known service consumers, which is for all purposes, static binding. Any change in the service's endpoint data will require all clients also to be

[7]Electronic Business using eXtensible Markup Language

reconfigured manually, to avoid "service unreachable" errors. Due to this, the service and the consumers are tightly coupled. Hence, the basic principle of SOA, that of building loosely-coupled systems supporting dynamic lookup and binding, is defeated (AbuJarour, Naumann, and Craculeac 2010).

The downsized model shown in Figure 1.5b has gained popularity in recent times primarily due to the lack of intelligent broker-based frameworks, that can overcome the issues that existed in the UDDI and the UBRs. In the current decentralized scenario, discovering Web services when there is no prior information about their endpoints has become quite an impossible task. An application designer who wants to find a service for a particular task may have to use a known service, without having the means to find potentially better, unknown services available over the Web. This has given rise to several difficulties, which are discussed in Section 1.3.3.

1.3.3 Prevalent Issues in Web Service Discovery

As discussed earlier, a major percentage of service providers currently seem to prefer hosting their services on their own servers or make them available through some third party portals like ProgrammableWeb, XMethods etc. Thus, application designers searching for new and potentially relevant services have to deal with these varied sources available in a distributed manner (Al-Masri 2009). Service discovery in such a distributed environment as the open Web has hence become quite challenging. Until 2005, the public UDDI registries (UBR) served as the primary source of service advertisements. Intended as a public platform for publishing services, the service repository of the UBR eventually became a source for mostly misclassified and incomplete service descriptions due to lack of enthusiastic adoption. Due to this, the UBRs that were hosted by Microsoft, IBM and SAP were discontinued. Since then, several issues have emerged in the area of Web service discovery, in light of the lack of popularity of existing registry approaches, and have garnered active research interest. Some of these are discussed below.

1. **Escalating volume of published services on the Web.**

   Fan et al. (2005) and Steinmetz (2009) conducted exploratory surveys on public Web services and presented statistics on the projected number of services available on the Web. A survey of services available in the UDDI/UBRs in 2006 and 2007 (Li, Zhang, et al. 2007) and those indexed by search engines (Song et al. 2007) reported that the number of published services has consistently doubled once in three years. Zheng et al. (2014) reported that the current number of available services on the Web is in tens of thousands. As services proliferate at such an exponential rate, the problem of finding them to serve a given requirement becomes harder and more time consuming.

2. **Current non-availability of an efficient mechanism for distributed Web service discovery and retrieval.**

   Currently, a common approach used by service providers that offer public Web services is to put up a list of services with a textual description attached to each service to explain their functionality on their own websites (AbuJarour, Naumann, and Craculeac 2010). Many of these descriptions may also be available in service portals, often in multiple service portals, thus compounding the problem of service discovery.

3. **Low quality of Web service documentation.**

   Kim et al. (2004) and Li, Zhang, et al. (2007) examined the complexity, number of operations, quality of documentation and the functional diversity of public Web services. These studies conclusively found that the services that were hosted on service providers' websites were the ones that were most up-to-date and had the most content-rich documentation. These services are typically found embedded in HTML Web pages, containing the descriptions about the services' functionalities etc. The service end-points were found quite poor in terms of the contents of the <documentation> tag, meant to contain a detailed description of the service capabilities. Of the 3500 service descriptions considered for Li, Zhang, et al. (2007)'s study, only about 15% were found to have documentation of more than 100 words, while about 67% of the services had less than 50 words. About 900 services did not have any descriptions at all. Also, the majority of the services did not have any documentation for the service operations. This means that the traditional methods of keyword based search will potentially overlook relevant services, which may not have the required terms to be discovered, during querying (Stroulia et al. 2005).

4. **Limited availability of semantic service descriptions.**

   As discussed earlier in Section 1.2.3, the vast majority of already existing Web services are WSDL based and do not have explicitly defined semantics. The process of adding semantic annotations to the thousands of published service descriptions is a massive undertaking and will require extensive time and effort from the service providers, which is unreasonable to expect.

5. **Lack of adequate domain information.**

   Without an adequate service organizing architecture and domain knowledge, the actual domain which a service belongs to is hard to capture, as similarity based categorization cannot be performed. Without domain-specific grouping, the process of querying a service collection to find relevant services for a given query becomes very tedious and time-consuming (Nayak et al. 2007).

6. **Limited context-sensitive querying support.**

   Considering the service consumer's perspective during the service discovery process, it is quite possible that the user may not be aware of all the knowledge that constitutes the domain of the services they are seeking to find. Specifically, the user may not be aware of all the terms related to the service request. As a result of this, many services relevant to the request may not be retrieved. For example, a user may need services that find '*zipcode*' of a city anywhere in the world, not knowing that some countries use the term '*pincode*' and '*postal code*' to refer to the same information. Hence, only services that support zipcode retrieval services for cities in USA may be returned, ignoring other potentially relevant services. When the user uses the selected service to find the '*zipcode*' of Mumbai city, then a "*No data found*" error may be returned, as services serving countries other than the USA, would use country-specific terminology.

7. **Limited support for composition-oriented service discovery.**

   One of the limitations of the UDDI and UBRs was their lack of support for discovering services for serving complex queries. Most existing approaches for service discovery focus only on simple service discovery given a set of user requirements. Being able to identify services that could be potentially linked together in sequence to perform certain aggregated functionalities is a critical requirement that is missing in the current scenario.

In view of these core issues in the Web service discovery phase, it is crucial to address them to ensure the continuing growth and popularity of Web service based application development. Intelligent solutions to enable semantics based distributed Web service discovery and retrieval are the need of the day. In the next chapter, we present a state-of-the-art review of the existing research work in the area of Web service discovery and then, formulate the problems addressed in this thesis.

1.4 Summary

In this chapter, the concepts of the Semantic Web and the Semantic Web services were discussed. The need for inclusion of semantics in the process of critical service life cycle tasks has become paramount due to the lack of adequate support in the existing Web service conceptual model. Based on the challenges discussed in the previous section, we summarize that there exists scope for improving the process of Web service discovery by addressing new research problems in the area, specifically in the directions listed below:

- Developing effective mechanisms for distributed Web service discovery to address the issues in finding and retrieving service descriptions available on the open Web.
- Developing techniques for automatically adding semantic annotations and generating metadata for large service collections to address the lack of explicit semantics in service descriptions and low quality of service documentation.
- Developing algorithms to effectively categorize large service collections to support domain specific discovery.
- The exploitation of background knowledge and meta-data for providing intelligent querying mechanisms for service consumers.
- Exploring methods for semantics based composite Web service discovery.

1.5 Organization of the Thesis

This thesis is structured as follows -

In Chapter 2, a review of various solutions existing in literature for semantics based Web service discovery, including composition-oriented service discovery, domain specific categorization, and context-sensitive user query handling is presented. Based on our observations on research gaps, we discuss the identified research directions in view of our work. In Chapter 3, the problems addressed in this thesis are formally defined and the scope of the work is discussed. We also present the general solution methodology and the definitions that are necessary to understand the defined problems.

In Chapter 4, we discuss the proposed framework for distributed Web service discovery for finding and retrieval of services available from varied sources and the process of efficient indexing of these services in a central repository. We also discuss the various problems associated with distributed service discovery and present certain optimization to the proposed framework. We prove that this framework is scalable, helps in eliminating redundancy and also deals efficiently with any changes in the service collection. Empirical evaluations and experiments are presented to validate the proposed approach.

In Chapter 5, we propose similarity analysis and semantics based processing techniques for automatically generating metadata for indexed service collection to address the lack of semantics in the descriptions themselves. Chapter 6 presents a novel, semantics based service categorization algorithm for domain specific grouping of similar services. Chapter 7 discusses the proposed intelligent querying functionalities for adding knowledge based support to the proposed framework.

In Chapter 8, a detailed discussion of the process of Web service discovery for both atomic and composite services using the developed framework is presented. The efficiency of this entire system is validated by both theoretical analysis and experimental analysis on the framework's service collection and using standard datasets. In Chapter 9, we summarise the contributions of our research work and highlight the possible future research directions.

Chapter 2

Literature Review

In Chapter 1, a discussion on the background, scenarios, challenges and research motivations in semantics based automatic service discovery was presented. The problems addressed by this thesis lie in the wider context set by SOA, Semantic Web and Context-aware computing. Our research work is contextualized within the current trend of semantically enriching information published on the Web, to enable automated processing and autonomous interaction between systems, through Web services. We focus on the phase of *Service Discovery* as this plays a very important part in the Web service life cycle.

In this chapter, a review of existing work dealing with various challenges and issues (as summarized in section 1.4 of Chapter 1) in the process of large-scale, distributed Web service discovery is discussed. The state of the art in the area of web service discovery architectures and techniques in literature are presented in Section 2.1. The problem of semantic annotation and metadata generation for service collections and a review of the existing approaches is discussed in Section 2.2. The problem of categorizing large service collections semantically and the available techniques is studied in Section 2.3. Following this, research works that focus on improving the querying process from the user's perspective in the area of service discovery are discussed on Section 2.4. The problem of composition-oriented service discovery is discussed in Section 2.5, followed by identified research directions and summary.

2.1 Web Service Discovery - A Review

Discovery of relevant Web services for given requirements has been a fundamental area of research in service-oriented computing. As stated before (in Chapter 1), after the failure of the UBRs in 2006, services are truly distributed over the Web and are available from heterogeneous sources like service portals, providers' Web servers etc, making discovery a very challenging task. Current Web service technologies do not support the automation of critical processes such as discovery and ranking, which are necessary for later stage tasks like selection, composition and execution. In literature, researchers have tried to address this problem in different ways and the pros and cons of these works is presented in the following section.

2.1.1 Web Service Discovery Architectures

Several early works in the area of Web service discovery can be categorized based on the methodology adopted in finding them. These works are based on the locality of services, due to which there are several diverse categories proposed. Such architectures can be broadly classified as *Centralized* and *Decentralized* (or *Distributed*) architectures. Figure 2.1 shows the classification of the various approaches discussed.

2.1.1.1 Centralized Architectures

In a centralized Web service discovery architecture, service descriptions are available in a central repository or registry, using which clients can articulate their requests and find required services. Some techniques that propose centralized repository based architectures for Web service discovery are discussed here. Xu (2007) proposed a centralized approach that stores service reputation information within the repository to enable QoS based Web service discovery. The reputation scores of a service in this approach are computed through user feedback, and are used during service matchmaking and ranking. Skoutas (2008) proposed an index-based method in centralized environments, for fast selection of services when a query is submitted. This approach relied on matchmaking of service request and the service advertisements of a limited set of indexed services, and focused on fast retrieval of services using efficient data structure. Their approach was later extended for service discovery in peer to peer overlay networks. Yu et al. (2012) designed and developed a centralized approach based on a service repository that allows faceted search and visual discovery of service descriptions for both providers and consumers. While their system supported operation-level matching of services indexed in the repository, it provided only keyword based search, thus affecting recall.

1. *UDDI based Approaches:* UDDI is an initiative to bring standardization to the process of service advertising and discovery. The UDDI standard allows businesses and individuals to publish service advertisements in a standardized manner that allows clients to invoke APIs to perform various tasks like publishing, inquiry, subscription etc. It is intended to give a well-organized access to the

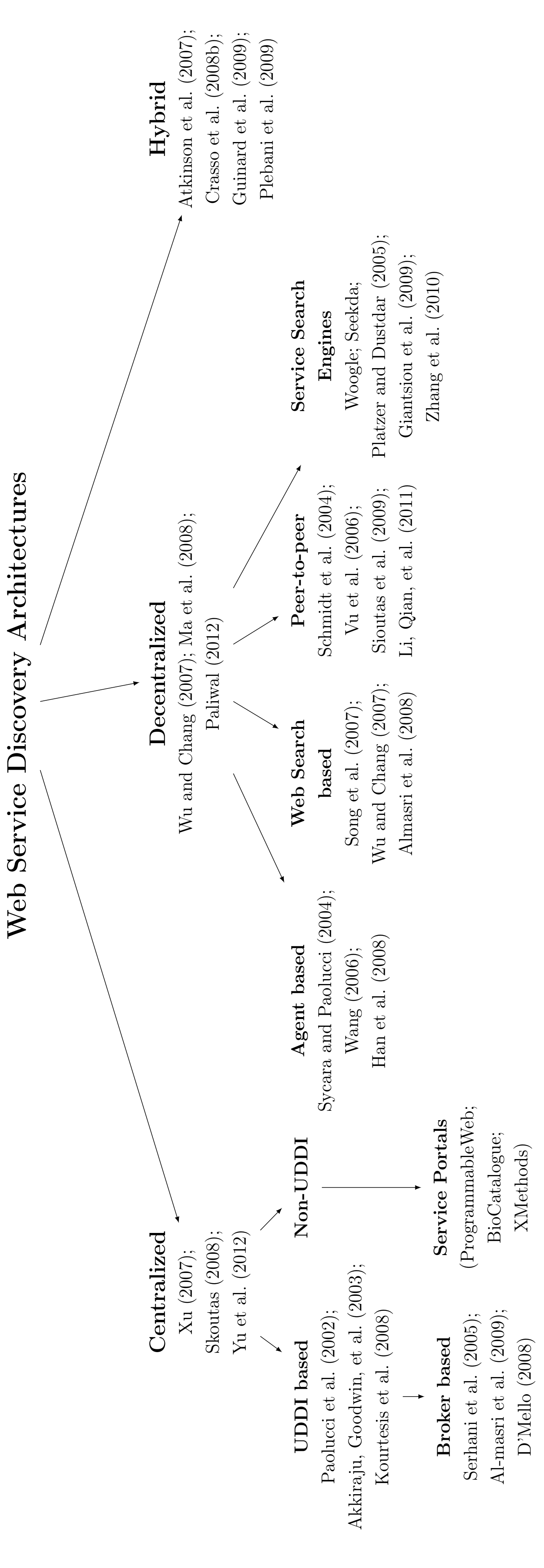

Web Service Discovery Architectures
Centralized
Xu (2007); Skoutas (2008); Yu et al. (2012)
UDDI based
Paolucci et al. (2002); Akkiraju, Goodwin, et al. (2003); Kourtesis et al. (2008)
Broker based
Serhani et al. (2005); Al-masri et al. (2009); D'Mello (2008)
Non-UDDI
Service Portals
(ProgrammableWeb; BioCatalogue; XMethods)
Decentralized
Wu and Chang (2007); Ma et al. (2008); Paliwal (2012)
Agent based
Sycara and Paolucci (2004); Wang (2006); Han et al. (2008)
Web Search based
Song et al. (2007); Wu and Chang (2007); Almasri et al. (2008)
Peer-to-peer
Schmidt et al. (2004); Vu et al. (2006); Sioutas et al. (2009); Li, Qian, et al. (2011)
Service Search Engines
Woogle; Seekda; Platzer and Dustdar (2005); Giantsiou et al. (2009); Zhang et al. (2010)
Hybrid
Atkinson et al. (2007); Crasso et al. (2008b); Guinard et al. (2009); Plebani et al. (2009)

Figure 2.1: Classification of Web Service Discovery Architectures

human-related, business-related and technical information of a service. The UDDI based Universal Business Registries (UBRs) were intended as a central directory of all available services. But they suffered from several major problems, resulting in their discontinuation. These issues are listed here.

- UBR enrolment was voluntary, gradually rendering its collection stagnant, due to disinterest in maintaining already published service advertisements.
- UBR enrolment was public, leading to the availability of both genuine business services and spurious/incomplete services published by novices, thus affecting the quality and dependability of the collection (Almasri et al. 2008).
- UBR depended on service providers to correctly categorize their services, often leading to misclassification (Klusch et al. 2006). Services with similar functionality may be listed in different categories (e.g. Service *S1* for finding local attractions may be listed in a different category than a related service *S2*, that finds local events), while dissimilar services may be listed in same category (e.g. services for currency conversion and HTML to XHTML conversion may both be listed under a generic category called 'conversion').
- UBR being public and open, could not guarantee the validity or quality of information it contained.
- Finding services published in the UBR was primarily keyword based resulting in low recall, thus affecting precision (Akkiraju, Goodwin, et al. 2003).
- The UDDI/UBR offers no support towards finding composite Web services.

Aiming to solve these issues, researchers proposed various enhancements to the UDDI. Paolucci et al. (2002) proposed the concept of adding semantic capabilities to the UDDI to enhance service discovery by using the OWL-S ontology. Akkiraju, Goodwin, et al. (2003) extended this idea by proposing enhancements to the UDDI inquiry API for supporting explicit user requirement specification. They also developed techniques for semantic matchmaking and automatic service compositions using BPEL4WS. Both these approaches required some changes to the UDDI APIs and implementation of a separate UDDI server. Kourtesis et al. (2008) proposed an extended UDDI registry called FUSION that uses SAWSDL and DL based reasoning for effective matchmaking.

2. *Broker based Approaches:* In *Broker based* methods (D'Mello 2008; Al-masri et al. 2009; Serhani et al. 2005), an additional component called the *service broker* acts like a middleware that facilitates service related tasks. Additional quality

assurances can be provided by the broker by incorporating QoS based execution and verification.

Centralized approaches offer several advantages over other approaches. Service descriptions are available in at one site, in a repository or a registry. Users can articulate their queries at this point to retrieve required service information. However, the registry can become a bottleneck or single point of failure in case of high access traffic, thus affecting performance. In fact, several researchers proposed the concept of distributed and/or replicated registries to overcome this issue. For addressing the problem of searching in these multiple registries, techniques for propagating search queries over multiple federated registries to return overall results were proposed (Al-Masri et al. 2007b; Sivashanmugam et al. 2004; Verma et al. 2005; Zhang, Akkiraju, et al. 2007).

The core idea of these approaches was facilitating efficient interaction between different distributed public/shared/private registries for query propagation and service discovery. Also, newer UDDI versions added support for accommodating interactions between distributed registries. In B2B scenarios, different enterprises may host their own UDDI servers as private/shared deployments, due to which, the number of UDDI registries themselves may be more, needing distributed service discovery. These developments ultimately paved the way to decentralized and hybrid architectures.

2.1.1.2 Decentralized Architectures

Decentralized or distributed approaches focus on the problem of finding services available at providers' sites. These can be classified into four different categories, based on the type of access methods employed in finding and retrieving distributed service descriptions and also managing service collections.

1. *Agent-based Architectures:* These methods make use of intelligent, software agents for distributed Web service discovery (Garcia-Sanchez et al. 2009; Han et al. 2008; Sycara and Paolucci 2004; Wang 2006). Agent-based algorithms facilitate interaction between multiple distributed agents that can remotely interact with their peers for dynamic activity coordination to complete a given task (in this case, finding and returning relevant Web services). An agent-based broker can perform very complex reasoning tasks like - *matchmaking* between service capabilities and requester queries; finding the best service based for a particular query *(ranking)*, invoking the selected service on the requester's behalf *(invocation)*, interacting with the service whenever required *(execution)*; and returning query results to the requester (Sycara and Paolucci 2004).

Recent works in this area have concentrated on autonomous multi-agent interaction and automated Web service composition (Klusch 2012).

2. *Web search based Architectures:* Several public Web service exploratory surveys are available in literature that were based on keyword based queries fed to conventional search engines like Google, to find services. Such surveys (Fan et al. 2005; Kim et al. 2004; Li, Zhang, et al. 2007) focused on analysing the number, quality, geographic distribution, complexity, development environment, message characteristics, responses, versioning etc., of publicly accessible Web services, using primarily Web search. Song et al. (2007) used conventional search engines to discover services, using keyword based queries, aimed at finding service descriptions embedded with Web pages. Wu and Chang (2007) proposed a system for retrieving relevant web services directly from the Web using data mining techniques. Al-Masri et al. (2008) conducted a similar experiment to determine the liveliness quotient of services published on the Web and reported that the rate at which number of services accessible through search engines grew at almost 286% when compared to a modest 12% growth in UDDI registries.

3. *Peer-to-Peer based Architectures:* Peer-to-peer (P2P) computing is "*a distributed application architecture that partitions tasks or workloads between equivalent peers in a network*". A P2P approach offers some advantages like improved scalability by avoiding dependency on centralized points; eliminating the need for costly infrastructure as direct communication among clients is enabled and resource aggregation (Milojicic et al. 2002). In literature, some proposals for P2P enabled approaches for Web service discovery exist, that aim to overcome the issues associated with centralized approaches. Such works are on efficient indexing data structures like Internet-scale distributed hashing tables (DHT), that efficiently map a service's data elements to physical peers to enable Web service discovery. Improvements proposed for optimizing the distributed indexing schemes led to the development of robust P2P overlay frameworks like *Chord* (Stoica et al. 2001; Stoica et al. 2003), *SCRIBE* (Castro et al. 2002), and *Tapestry* (Zhao et al. 2004).

   Based on these frameworks, several techniques for P2P based scalable Web service discovery were proposed, that can support keyword based searches (Schmidt et al. 2004), QoS based selection (Vu et al. 2006), similar service search (Li, Qian, et al. 2011) and semantic service publication (Si et al. 2013). Here, the peers store the service information robustly so it can be quickly retrieved using the distributed hashing structure. However, implementing and maintaining P2P networks as the number of peers increase in number and locality. Also, there may be trust issues while sharing service information among peers, which have to be addressed effectively.

4. *Service Search Engines:* In recent research, a new paradigm for Web service discovery has emerged, that is based on applying information retrieval methodologies similar to search engines to find services published on the Web. The focus in these approaches is to find and retrieve services and create a centralized repository where users can articulate their search queries. The forerunners of these approaches were service search engines like Woogle (Dong et al. 2004), Service-finder (Brockmans et al. 2009; Della Valle et al. 2008) and Seekda (Steinmetz 2009), but none of these are available online currently. Other researchers proposed different techniques to retrieve service descriptions, like vector space model based indexing (Platzer and Dustdar 2005) and conventional search engine based discovery (Song et al. 2007; Wu and Chang 2007).

2.1.1.3 Centralized vs. Decentralized Approaches

Table 2.1 summarizes the various features and distinctions in centralized and distributed service discovery approaches. Each category is suited for different scenarios and have their own advantages and disadvantages, some of which are discussed here.

Table 2.1: Centralized vs. Decentralized Web Service Discovery Architectures

Criteria	**Centralized**	**Decentralized**
Facility for publishing services?	Yes.	No. *(Indexing based)*
Availability of business data?	Yes.	No.
Provides search features?	Yes. *(Keyword based)*	Yes. *(Keyword/Semantics)*
Provides service categorization?	Manual *(uses standard taxonomies.)*	Intelligent categorization techniques have been used.
Are WSDLs stored?	No.	Yes.
Guarantee of quality of service description?	No.	No. *(can be incorporated)*
Support for versioning?	No.	No. *(can be incorporated)*
Caching supported?	No.	No. *(can be incorporated)*
Automatic updates supported?	No.	Limited. *(Intelligent techniques need to be incorporated)*

2.1.1.4 Hybrid Architectures

Hybrid architectures form the third category of Web service discovery architectures, consisting those works that incorporate the best features of both the centralized and decentralized architectures. Research work in this area employ varied methods for finding and discovering distributed Web services and then provide registry like indexing facilities similar to that of UDDI, and user interfaces for querying (Atkinson et al. 2007; Guinard et al. 2009). In literature, there exist approaches that augment the underlying hybrid architecture by the use of ontology based categorization (Crasso et al. 2008b; Garcia-Sanchez et al. 2009), domain-specific clustering (Ma et al. 2008; Nayak et al. 2007), semantic matchmaking (Paliwal 2012; Plebani et al. 2009) and machine learning based classification (Chen et al. 2013a; Crasso et al. 2008b; Skoutas 2010a).

The main advantage of hybrid architectures is that, they are well suited for the current scenario where services are available from heterogeneous sources and are also numerous in number. Most of these approaches also focus on overcoming some problems of the UDDI (like keyword based searching and manual categorization) by applying intelligent techniques like domain-specific clustering and classification. Such techniques are discussed in more detail in Section 2.2.

2.1.2 Web Service Discovery Techniques

The objective of Web service discovery is to facilitate access to available service descriptions for binding. Service descriptions essentially encapsulate the functional capabilities of a service, while other non-functional aspects such as QoS, QoE (Quality of Experience) and CoS (Cost of Service) are handled separately, for example, by a QoS broker. Hence, techniques focusing on Web service discovery can be categorized into *Functional Aspects based* and *Non-functional Aspects based* techniques. Figure 2.2 summarizes the various techniques classified as per these two aspects.

2.1.2.1 Functional Aspects based Discovery

The functional capabilities of a Web service are represented by its description (WSDL), which is available at registries/portals/service endpoints. Such functional semantics can be extracted for serving user queries. Two main approaches can be identified herein - *Syntactic Matching* and *Semantic Matching*.

A. Syntactic Matching

Syntactic matching mechanisms are primarily based on using the keywords, categories or service interfaces of available services to facilitate service discovery process. We discuss existing works under this category next.

Web service discovery techniques

- ***Functional aspect based methods***
  - **Syntactic matching**
    - *Keyword based matching* (Song et al. (2007); Wu and Chang (2007); Al-Masri et al. (2007b))
    - *Indexing based* (Dong et al. (2004); Della Valle et al. (2008); Steinmetz (2009))
    - *Interface/signature matching* (Wang and Stroulia (2003); Gao et al. (2005); Hao et al. (2007))
    - *Similarity based matching* (Stroulia et al. (2005); Kokash et al. (2007); Grigori et al. (2006); Berardi et al. (2005))
  - **Semantic matching**
    - *Functional semantics based* (Ye et al. (2006); Shin and Lee (2007); Shin (2009); D'Mello and Ananthanarayana (2009))
    - *Using additional annotation* (Heß et al. (2003); Oldham et al. (2005); Elgazzar et al. (2010); Fang et al. (2012))
    - *Ontology generation based* (Nayak et al. (2007); Sabou et al. (2005); Segev et al. (2012))
    - *Context-aware* (Broens et al. (2004); Xiao et al. (2010); Rasch et al. (2011); Sangers et al. (2013))
    - *Using semantic descriptions* (Sycara (2003); Klusch et al. (2006); Keller et al. (2006); Plebani et al. (2009); Schulte et al. (2010))
- ***Non-functional aspect based methods***
  - **QoS** (Ran (2003); D'Mello (2008); Torres et al. (2011); Alrifai (2012); Zheng et al. (2014))
  - **Service usage statistics** (Birukou et al. (2007); Chan et al. (2012); Zhou et al. (2013))
  - **Usability** (Namgoong (2006); Stollberg et al. (2007); Mohebbi et al. (2013))
  - **User preference/Personalization** (Shao et al. (2007); Liu (2013); Kuang (2012))

Figure 2.2: Hierarchy of Web Service Discovery Techniques

Keyword based Matching. In this category, the keywords that are present in the WSDL's *documentation* and *categoryBag* tags are extracted and utilized for service discovery (Al-Masri et al. 2007b). Other approaches include conducting keyword based searches using conventional search engines at regular intervals to search for services embedded within webpages and available within HTML forms (Song et al. 2007; Wu and Chang 2007). These approaches focused on using the efficient indexing of search engines to conduct faceted search for only WSDL documents. Some of the issues with these approaches are -

- All relevant services may not be retrieved as keywords may have many synonyms and associated senses (verb/noun).
- Mis-categorized services may not be returned even though they may be relevant, in case of category-based matching.
- There is no way to check validity/liveliness of the services retrieved by conventional search engines. Also, these approaches depend on the usage of wildcard characters (e.g. *keyword * keyword*) etc., for conducting Google/Yahoo searches, which can result in higher recall, but can adversely affect precision.

Indexing based Matching. Several service search engines (Della Valle et al. 2008; Dong et al. 2004; Steinmetz et al. 2008) that used traditional information retrieval techniques for finding services on the Web and indexed them into a repository, come under this category. Their main aim was to provide querying interfaces to the users, then quick matchmaking of the query terms with the indexed services to generate a ranked list of matching services. However, these frameworks are no longer available online currently and it is not possible to evaluate their performance.

Interface/Signature Matching. The Web service interface is a collection of functionally related operations. Wang and Stroulia (2003) proposed techniques for assessing the structural similarity of two WSDL documents by matching its operations and the element types, like input/output operations etc, using signature matching and type mapping. This was to be an extension to the UDDI API for improving the service discovery process. Other approaches proposed schema matching algorithm for supporting web-service operations matching that also considered schema semantics (Hao et al. 2007) and using the interface matching technique for selecting service for compositions (Gao et al. 2005). However, interface matching methods suffer from several issues -

- They assume that service designers have assigned meaningful names to the element names, however this is most often not the case.
- Similar element names may in fact be having different datatypes, which reduces the degree of similarity.
- Operations defined for similar functionalities may in fact have different signatures. For example, a city search service can use different operation signatures (using city and state name, using zipcode, using longitude/latitude, etc). In such a scenario, the requester could be satisfied with one option for serving the query, or may require all the options, depending on the requirement. It would be difficult to identify all such options in all cases, by just interface matching.

Similarity based Matching. Soem methods that aim to alleviate the issues associated with interface matching methods have focused on computing similarity between each service description and their natural language descriptions through partial-semantic matchmaking (Kokash et al. 2007; Stroulia et al. 2005). The problem of identifying similar behaviours when more than service is found relevant, generated a lot of interest and some research works used behaviour matching algorithms to evaluate the semantic distance between two service interactions (Berardi et al. 2005; Grigori et al. 2006). These approaches considered the messages exchanged between two services during a valid conversation and the activities performed within a service, to determine even partial matches. However, these approaches were computationally expensive and do not scale well, especially when the number of services and service interactions is large.

B. Semantic Matching

Web service discovery through semantic matching aims to the semantic context (or meaning) of terms used in a service description to determine a service's functionality without ambiguity. Depending on the type and level of semantics employed, these can be classified into seven broad categories. These are discussed below:

Functional Semantics based. Some available works in this category proposed varied approaches like - representing a service's functionality semantically, either by explicitly determining each service's domain and adding this information as its metadata (Ye et al. 2006), by considering the functionality of a Web service as a one-to-one relationship of action (input) and object (output) by analysing the natural language description of all services in a dataset (Shin 2009; Shin and Lee 2007) and through extensible functional knowledge based mapping of the operations in published descriptions into an abstract operation description for better mapping (D'Mello and Ananthanarayana 2009). Some limitations of such functional semantics based approaches are -

- Most available works are based on the definition of a functional domain ontology and the mapping of service element concepts to that of the ontology concepts. If the service element has more than one associated concept, then the mapping may not be valid.
- Creation and maintenance of domain ontologies for different services is an additional burden. Context aware systems were developed to overcome this limitation.

Using Additional Annotation. Researchers tried to organize or categorize service collections by generating additional annotation or metadata for each service by automatic analysis of service descriptions. Frameworks like Meteor-S (Oldham et al. 2005) and Assam (Heß et al. 2003; Heß et al. 2004) proposed techniques to annotate syntactic service descriptions with semantic concepts to enhance service discovery. Other approaches include techniques for analysing the meaning of the terms extracted from service descriptions and using them as features for clustering services as per their similarity (Elgazzar et al. 2010; Fang et al. 2012). These approaches have focused on generating additional explicit knowledge using the implicit data available in service functional descriptions to aid service discovery.

Ontology Generation based. Service ontology plays a major role in matching service functionality and the request. Some approaches propose the automatic generation of the domain ontology of the service through machine learning techniques based on the problem domain (Nayak et al. 2007; Sabou et al. 2005; Segev et al. 2012). Based on the textual description of the services, the relevant terms are identified and extracted, using which the domain is learned and the ontologies of the services are generated for enhancing processes like discovery and selection.

Context-aware. Context-awareness refers to the property of a system where the system can adapt its behaviour as per the change in circumstances. In the context of the service discovery problem, context-aware techniques aim to provide different results as per the change in service user preferences or service provider functionalities/conditions. For example, the discovery mechanism may offer support for discovery based on user location or service's availability timings etc. Some such techniques proposed consider matching of query concepts with that of service's domain ontologies, for identifying level of matching (Broens et al. 2004), context modelling for dynamic handling of various context types & their relationship to user request (Xiao et al. 2010) and user profile based context-aware discovery (Rasch et al. 2011).

Using Semantic Service Descriptions. In literature, several researchers have used the proposed semantic Web formalisms for measuring their suitability for Web service life-cycle tasks. Service descriptions defined using the three formalisms - OWL-S, WSMO and SAWSDL have been used in various literature, as the base dataset over which various techniques for discovery have been proposed.

Sycara (2003) presented an OWL-S based automated discovery, selection and composition framework using the OWL-S/UDDI matchmaker which adds semantic capabilities to the UDDI. The automatic service mediation was managed using the OWL-S process model. Klusch et al. (2006) also used OWL-S for semantic matchmaking over syntactic service descriptions with a matchmaker called OWLS-MX. The requests in this approach were specified in OWL-S while the dataset was WSDL documents, and the matchmaker was needed to achieve approximate matching between the request and the service documents.

Keller et al. (2006) proposed a method based on logical reasoning for the *Goal* to *Service* matchmaking process, as per the WSMO specification. Plebani et al. (2009) proposed a framework called URBE (UDDI Registry by Example) that uses a bipartite graph matching algorithm for both the inputs and outputs of a given service request and offer. They used SAWSDL descriptions to evaluate the performance and suitability of their approach for enhancing the UDDI registry. Schulte et al. (2010) used SAWSDL descriptions for matchmaking by combining syntax and semantic similarity measures with subsumption reasoning-based Degree of Match (DoM). Each of these approaches used the different semantic Web formalisms on datasets and conducted experiments to evaluate their adaptability to real-world service usage scenarios.

2.1.2.2 Non-functional Aspects based Discovery

Non-functional properties refer to those aspects where metrics like quality, efficiency, preferences, usage etc., can be considered and the process of service discovery can be enhanced further. Based on the non-functional parameters considered during the discovery process, this category can be sub-divided into four main sections - *QoS based*, *service usage statistics based*, *usability based* and *user preference based.* Non-functional approaches are discussed briefly here.

QoS based. QoS is the most commonly used parameter and several works exist in literature in the area of quality driven service discovery. Researchers proposed techniques for discovering services that offer the user desired level of QoS for various parameters like response times and locality (Kritikos et al. 2009), reputation based trustworthy service discovery (Ran 2003), self-adaptive service discovery using user-defined QoS constraints (Torres et al. 2011), efficiently representing quality requirements using a Quality Constraint Tree (QCT) (D'Mello 2008) and using QoS constraints for composing services (Alrifai et al. 2010) are some relevant works in this area.

Service Usage Statistics based. Recommending services based on their usage statistics and history is another interesting area in this category. Birukou et al. (2007) proposed a recommender system to help service-based application developers discover appropriate services using a task description and the history of previous decisions made for similar objectives. Their approach is based on observing service invocations over time to collect usage data so as to use this recommend the best services when new requests arrive. Chan et al. (2012) proposed a web service discovery system based on recommendation using collaborative filtering of user interactions with the services and historical usage data. Zhou et al. (2013) used service data and popularity in a service network for ranking services for a given user request.

Usability based. Usability measures the ease of use of performing a particular task. In the context of Web service discovery, techniques proposed in this category, take user/client system's convenience, capabilities or limitations into consideration during the discovery process. Namgoong (2006) proposed an matchmaking algorithm that computes the semantic matching between the client's requirements and the available service capabilities and ranks services based on a factor called *usability score*, which measures the level of convenience of the client while using the service. Stollberg et al. (2007) proposed a caching based technique to improve the subsequent discovery tasks using previous discovery results. This WSMO based system keeps track of identified *goal* templates for each query and use this information for reducing the search space of similar discovery requests that are submitted later in time. Mohebbi et al. (2013) used a technique called a *pre-matching filter* to reduce the search space independently of the user's querying process to achieve better response times during actual discovery.

User Preference/Personalization based Personalized service discovery focuses on emphasizing user's requirements beyond what is explicitly stated. Shao et al. (2007) proposed collaborative filtering based approach for assisting users in selecting services with best QoS by using similarity mining and prediction from past consumers' experiences. Kuang (2012) extended the idea of collaborative filtering-based service recommendation, by using a technique called *Bayesian Inference* to predict the QoS of an unused service for current service consumer. Liu (2013) proposed a situation aware collaborative filtering algorithm that returns personalized service results to a user, based on what other users invoked in various situations.

2.1.2.3 Functional vs. Non-functional Service Discovery

Several varied techniques are available in both functional and non-functional aspects oriented service discovery. The main advantage of functional service discovery techniques is that they are well suited for distributed, real-world Web services, most of which will not have any associated QoS, invocation data, usage statistics etc. These discovery techniques primarily are based on using the available service description documents and the elements that indicate the service's functionality (through natural language names as given by their service provider) to determine most relevant services for a given task. Hence, these techniques are well-suited for large service collections, without any additional associated metadata. Since, the current scenario is exactly this, and most services are simply available on the Web, functional semantics based Web service discovery is more practical and scalable. Table 2.2 presents an overall comparison of the two approaches and their relative merits/demerits.

Table 2.2: Functional *vs.* Non-functional Service Discovery

Criteria	**Functional**	**Non-functional**
Dependency	Only service descriptions are required	Service descriptions and their associated quality data (QoS/Cost/History/Invocation data etc.)
Data Availability	Available in repositories/ service portals or in other sources over the Web	QoS values have to be generated from specific real-world Web services during operation.
Suitability	Well suited for both centralized & distributed architectures	Suited for small service datasets with known QoS values.
Support for Personalization	Limited (context-aware techniques have to be incorporated)	Yes (users can select desired QoS parameters)
Scalability	Yes.	Limited (dependent on availability of QoS data).

In case of non-functional aspects based discovery, most works have used small datasets and have applied their techniques to closed environments, rather than to real-world Web services. It is difficult to capture QoS data, usage statistics and invocation details, unless specifically made available by the service providers or bought from a third party company that sells QoS data at premium rates. Gathering the required non-functional data itself is the biggest challenge and quite a daunting task. Understandably, most techniques proposed have been validated for small datasets and can be applied by administrators of closed repositories where relevant quality data is available.

Recently, there have been some concentrated efforts in gathering service QoS values for real-world Web services. The QWS dataset (Al-Masri et al. 2008) contains QoS data of 365 Web services, for nine different QoS parameters like latency, throughput, availability etc. The WS-DREAM dataset (Zhang et al. 2011) contains QoS data of 4532 services/142 users in terms of response times and throughput values. Currently, these are the only datasets available for QoS based discovery experiments to the best of our knowledge.

2.1.3 Web Service Discovery - Remarks

In Section 2.1, a comprehensive review of Web service discovery architectures and techniques was presented. A comparative study of centralized and decentralized approaches revealed several research gaps. Centralized approaches offer the advantage of standard taxonomy based classification, business oriented organization etc. Also, these offer a central point where users can articulate their search queries. But, they are mainly dependent on service providers for keeping the published data organized and up-to-date, which became the main point of failure. Decentralized approaches are more suited for the current scenario, as they support discovery over services available in distributed sources. However, existing approaches suffer from several issues, like lack of support for versioning, no caching, and no automatic updates. Hence, a hybrid architecture that incorporates the best features of both centralized and decentralized approaches, with additional improvements to overcome their negative aspects would be very advantageous.

Comparing functional and non-functional aspects based service discovery techniques, it can be said that functional approaches are more suited for distributed Web service discovery than non-functional approaches. Non-functional techniques have the limitation of unavailability of QoS data for unknown, distributed services, due to which they are not scalable, and hence are not well suited. Using semantics based processing and metadata for service descriptions during functional aspects based discovery can further enhance the discovery process. In this context, we explore existing techniques for adding semantics to service descriptions and for generating metadata, in Section 2.2.

2.2 Adding Semantics to Web Service Descriptions

Today's Web service standards are primarily motivated by the interoperability of software components over the Web and are built on XML. Services are usually described using natural language which is often too imprecise. As a result, WSDL-based service descriptions cannot be interpreted without human intervention. Accordingly, further functionalities such as automated Web service discovery, execution, or composition are very difficult to achieve (McIlraith et al. 2001).

As the set of available Web Services grows, it becomes increasingly important to have automated tools to help identify services that match user requirements. Finding suitable Web services depends on the facilities available for service providers to describe the capabilities of their services and for service consumers to describe their requirements in an unambiguous, and ideally, machine-interpretable form (Berners-Lee et al. 2001). Adding semantics to represent the requirements and capabilities of Web services is essential for achieving unambiguity and machine-interpretability.

2.2.1 Using Semantic Web Service Formalisms

Today, Semantic Web Services are a prominent field of research and have resulted in a number of different approaches and standards such as the Web Ontology Language for Services (OWL-S), the Web Service Modeling Ontology (WSMO) and Semantic Annotations for WSDL and XML Schema (SAWSDL). These are formalisms that explicitly make use of semantic technologies to semantically describe different parts of a service description. Here, we examine the salient features of each of these formalisms.

1. Web Ontology Language for Services (OWL-S)

OWL-S (Burstein et al. 2004) is an ontology within the OWL-based framework of the Semantic Web, for describing Semantic Web Services. Its aim is to enable users and software agents to automatically discover, invoke, compose, and monitor Web resources offering services, under specified constraints. The class *Service* provides an organizational point of reference for a declared Web service; each distinct published service will have one instance of *Service* ontology, which has three properties - *presents*, *describedBy* and *supports*. The classes *ServiceProfile*, *ServiceModel*, and *ServiceGrounding* are the respective ranges of these properties. Each instance of *Service*, *presents* a *ServiceProfile* ontology, be *describedBy* a *ServiceModel* ontology, and *supports* a *ServiceGrounding* ontology. Figure 2.3 shows the components of the OWL-S Ontology.

i. *ServiceProfile* is used to describe what the service does. This information is primary meant for human reading, and includes the service name and description, limitations on applicability and quality of service, publisher and contact information.

ii. *ServiceModel* describes how a client can interact with the service and provides the sets of inputs, outputs, pre-conditions and results of the service execution.

iii. *ServiceGrounding* specifies the details that a client needs, to interact with the service, such as, communication protocols, message formats, port numbers, etc.

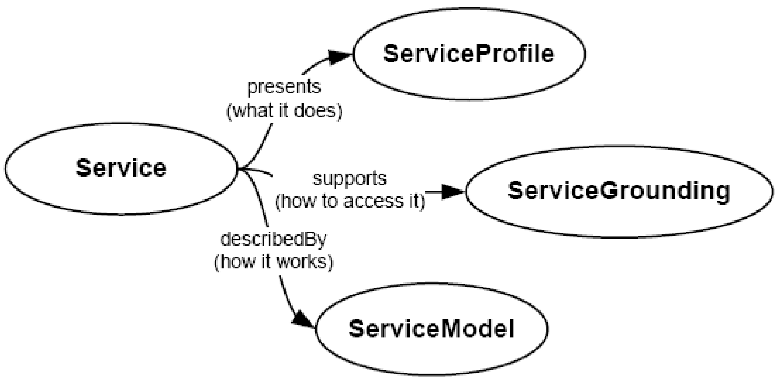


Figure 2.3: OWL-S Ontology Components

2. Web Service Modeling Ontology (WSMO)

WSMO (De Bruijn et al. 2005) is the other major approach for modelling services semantically. WSMO provides a framework for semantic descriptions of Web Services and acts as a meta-model for such Services based on the Meta Object Facility (MOF). Ontologies are described in WSMO at a meta-level. A meta-ontology supports the description of all the aspects of the ontologies that provide the terminology for the other WSMO elements. Figure 2.4 shows the components of WSMO.

Semantic service descriptions, according to the WSMO meta model, can be defined using the language defined by WSML (Web Service modelling Language), which consists of four core elements deemed necessary to support Semantic Web services: Ontologies, Goals, Web Services and Mediators.

i. *Goals* are defined as the objectives that a client may have when invoking a service.

ii. *WebServices* provide a semantic description of services on the Web, including their functional (Capability) and non-functional (Interface) properties, as well as other aspects relevant to their interoperation.

iii. *Mediators* in WSMO are special elements used to link heterogeneous components involved in the modelling of a Web service. They define the necessary mappings and transformations between linked elements.

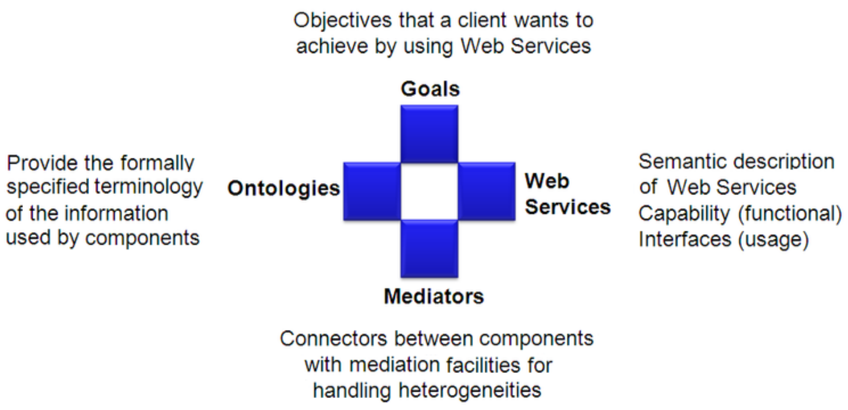

Objectives that a client wants to achieve by using Web Services
Goals
Provide the formally specified terminology of the information used by components
Ontologies
Web Services
Semantic description of Web Services Capability (functional) Interfaces (usage)
Mediators
Connectors between components with mediation facilities for handling heterogeneities

Figure 2.4: Web Service Modeling Ontology Components

3. Semantic Annotations for WSDL and XML Schema (SAWSDL)

SAWSDL (Kopecky et al. 2007) defines how to add semantic annotations to various parts of a WSDL document such as input and output message structures, interfaces and operations. The extension attributes defined in this specification fit within the WSDL 2.0, WSDL 1.1 and XML Schema extensibility frameworks. The annotations on schema types can be used during Web service discovery and composition. In addition, SAWSDL defines an annotation mechanism for specifying the data mapping of XML Schema types to and from ontology; such mappings could be used during invocation, particularly when mediation is required. To accomplish semantic annotation, SAWSDL defines extension attributes that can be applied both to WSDL elements and to XML Schema elements.

i. *modelReference* - an extension attribute to specify the association between a WSDL or XML Schema component and a concept in some semantic model. It is used to annotate XML Schema type definitions, element declarations, and attribute declarations as well as WSDL interfaces, operations, and faults.

ii. *liftingSchemaMapping* and *loweringSchemaMapping* - two extension attributes that are added to XML Schema element declarations and type definitions for specifying mappings between semantic data and XML.

Comparing Semantic Web Service Formalisms. Table 2.3 compares the three semantic web service formalisms on the basis of common parameters. Lara et al. (2004) reported that users often have difficulty recognizing the differences between these formalisms and choosing the appropriate paradigm for their application. Hence, a systematic comparison between the three approaches in the context of different views: the service consumer, provider and broker, is presented.

As seen from the Table 2.3, SAWSDL is currently a W3C recommendation, as it aims to add semantic elements to WSDL itself. However, when a service is to be advertised, SAWSDL is intended to be used along with OWL-S or WSMO. From both consumers' and provider's viewpoint, OWL-S seems to be better choice, currently, as some semi-automated tools are available for one-on-one mapping between WSDL and OWL-S. There still require manual intervention, but can still reduce the extensive work required in manually generating semantic service descriptions.

Table 2.3: Comparison of OWL-S, WSMO and SAWSDL approaches.

Context	**OWL-S**	**WSMO**	**SAWSDL**
W3C Status	Submission	Submission	Recommendation
Usage	As an additional description	As an additional description	as an enhanced WSDL
Basis	Based on the W3C standard Web Ontology Language (OWL)	Based on Web Service modelling Language (WSML)	Focus is on adding new elements to WSDL itself while developing it.
Consumer's viewpoint	Profile specifies the service objectives for both user's requests and provider's advertisements.	The Goal is defined to describe users' needs.	To be used along with OWL-S, WSMO or other ontologies for user advertisements.
Provider's viewpoint	Service Model is used to represent the service's control and dataflow	defines a special constructor *webService* to describe the service.	Not Applicable
Inputs, Outputs, Preconditions & Effects (IOPE)	Explicitly defined as per the IOPE Model	Defined by a Capability constructor	Defined by WSDL itself (only input and output constructs), without any semantics.
Orchestration	uses a process based description of how complex services invoke basic services	details of how WSMO orchestrations will be described are not fully defined	Not Applicable

2.2.2 Generating Service Metadata

In the absence of semantic annotations in most service descriptions, researchers tried to compensate by using intelligent processing techniques to capture the inherent functional semantics of a service available in its description. Some studies (AbuJarour, Naumann, and Craculeac 2010; Li, Zhang, et al. 2007) reported that, even though the quality of natural language documentation for services is poor, service designers mostly use well-defined element names and the quality of WSDL is generally good.

In this context, some works focused in adding metadata automatically to services without the use of semantic Web service formalisms. Heß et al. (2003) and Heß et al. (2004) proposed a machine learning based approach that used Naive Bayes and SVM algorithms to automatically discover the semantic categories of Web Services. They used both the WSDL and the UDDI entry of 364 services to train the ensemble classifier, to train it to automatically attach such domain labels to new test data. They reported a maximum precision of 0.68 when used on a dataset of 100 test services. Since they did not consider the operation and message level semantics of the WSDL, this approach could not achieve better precision during service discovery.

Agarwal et al. (2004) proposed OntoMat-service, a tool intended for service workflow designers for manually adding domain metadata to model service workflows. Bau III et al. (2008a) and Bau III et al. (2008b) proposed an annotation framework for stateless and stateful services, where a developer can explicitly define the web service logic using a standard programming language. These constructs are augmented with declarative annotations specifying preferences for exposing the logic as a stateless/stateful web service. At compile time, an enhanced compiler analyses the annotated source file and automatically generates the mechanisms required to expose it as a stateless/stateful web service. However, these tools require manual effort from service designers at development time, hence are unsuited for already published services.

Elgazzar et al. (2010) and Fang et al. (2012) used the service's WSDL and similarity measures like Normalized Google Distance (NGD) (Cilibrasi et al. 2007) to measure the similarity between terms to automatically annotate the service domain to a given dataset of services. NGD is a similarity metric based on the concept of information distance, and uses the Google search API to automatically compute distance between two words via Google indexed terms. Using NGD, the authors calculated the similarity between every term extracted from the WSDL to determine relative frequencies, using which the domain was inferred and the service was annotated with its domain information. The authors claimed good accuracy at inferring the domain, however, this method is very time consuming as NGD of every term pair has to be computed by using the results returned by the Google API. However, the advantage of these approaches is that semantics and metadata can be generated from available data, i.e., the WSDLs, thus at least partially overcoming the problem of current non-availability of explicitly defined semantics in service descriptions.

2.2.3 Adding Semantics to Web Service Descriptions - Remarks

As discussed in Section 1.2.3, automatically generating semantic annotation for service descriptions is quite challenging. Due to the volume of published service descriptions, the time and effort required for manually/semi-automatically generating these descriptions is quite massive, due to which adopting semantic Web service formalisms for large-scale service annotation, especially for distributed service discovery, may be difficult. Adopting generated metadata based approaches using intelligent processing techniques to capture the functional semantics of a service from its service description may be potentially more suited for current scenario. In the next section, we discuss existing approaches for categorizing Web services available in published literature.

2.3 Review of Web Service Categorization Approaches

One of the major tasks of service management is effective service categorization to facilitate service retrieval. Categorization enables domain specific searching, discovery and search space reduction by eliminating irrelevant categories for a given user request. Most categorization techniques aim to find those parameters based on which two objects that need to be categorizes are sufficiently different, so they can be said to belong to different categories. As such, current Web service categorization techniques can be broadly classified into four categories - *Taxonomy based*, *Clustering based* and *Classification based*. Some works that use hybrid techniques that use the merits of other approaches are discussed under *Hybrid approaches*. Some relevant work from published literature in each of these categories are discussed in this section.

2.3.1 Taxonomy based Web Service Categorization

The earliest efforts at categorization used the concepts of a taxonomy or hierarchical structuring. Taxonomies may refer to a classification of things or concepts, as well as to the principles underlying such a classification (Grossi et al. 2004). Some standard taxonomies that are available currently are UNSPSC[1], NAICS[2] and SIC[3]. Some systems that categorize services using taxonomies are presented below.

[1]United Nations Standard Products and Service Codes
[2]North American Industry Classification System
[3]Standard Industrial Classification

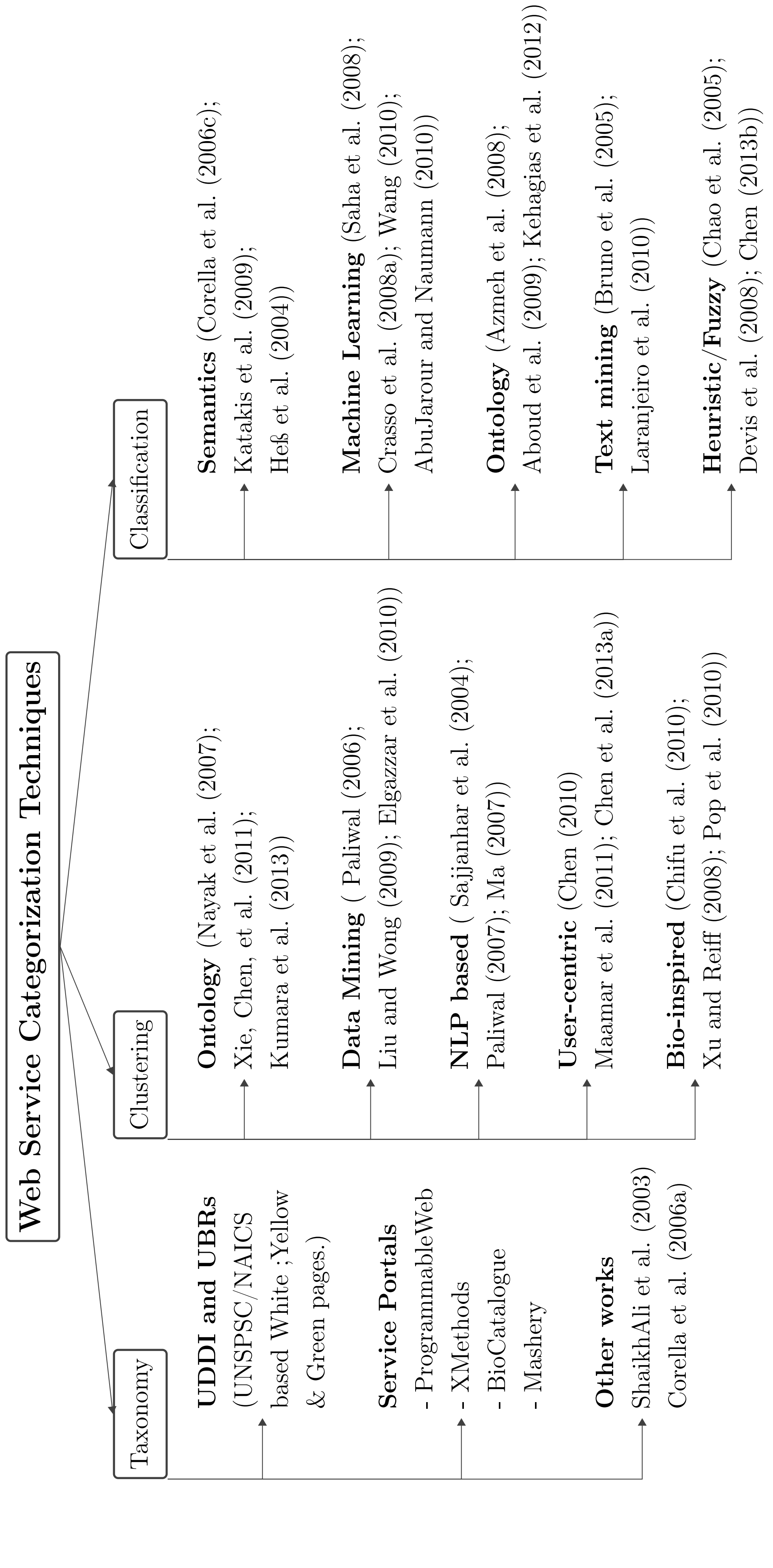

Web Service Categorization Techniques
Taxonomy
Clustering
Classification
UDDI and UBRs (UNSPSC/NAICS based White ;Yellow & Green pages.)
Service Portals
- ProgrammableWeb
- - XMethods
- BioCatalogue
- Mashery
Other works
ShaikhAli et al. (2003)
Corella et al. (2006a)
Ontology (Nayak et al. (2007); Xie, Chen, et al. (2011); Kumara et al. (2013))
Data Mining (Paliwal (2006); Liu and Wong (2009); Elgazzar et al. (2010))
NLP based (Sajjanhar et al. (2004); Paliwal (2007); Ma (2007))
User-centric (Chen (2010) Maamar et al. (2011); Chen et al. (2013a))
Bio-inspired (Chifu et al. (2010); Xu and Reiff (2008); Pop et al. (2010))
Semantics (Corella et al. (2006c); Katakis et al. (2009); Heß et al. (2004))
Machine Learning (Saha et al. (2008); Crasso et al. (2008a); Wang (2010); AbuJarour and Naumann (2010))
Ontology (Azmeh et al. (2008); Aboud et al. (2009); Kehagias et al. (2012))
Text mining (Bruno et al. (2005); Laranjeiro et al. (2010))
Heuristic/Fuzzy (Chao et al. (2005); Devis et al. (2008); Chen (2013b))

Figure 2.5: Web Service Categorization Techniques

UDDI and UBRs. The UDDI and the UBR were envisioned as a business registry for enabling businesses to publish service listings and discover each other, and to define how the services or software applications interact over the Internet. The UDDI being an open industry initiative, standard taxonomies were adopted for organizing the business and service listings in its core data structures - the White pages, the Yellow pages and the Green pages (*Universal Description, Discovery and Integration (UDDI)* n.d.).

Some of the core problems associated with adopting standard taxonomies is that, most are extremely large, consisting of thousands of categories, within multiple hierarchical levels. The placement of a service under the most suitable category requires a considerable amount of knowledge of the taxonomy, the service characteristics, the application domain, the overall organization of the repository, implicit guidelines, etc., in order to make good classification decisions. Since UDDIs allowed publishers to classify their services themselves, this often led to sub-optimal classification and sometimes, misclassification.

Service Portals. Service portals follow their own classification criteria, and also allow service publishers to choose the categories in which their services are placed in. Publishers manually submit their service/API details like service endpoint, descriptions etc., through a form. During this process, they select one of the several categories that ProgrammableWeb defines. Currently, there are more than 480 categories, ranging from 'Mapping', with the largest collection of 3892 services, while the smallest category is named 'Aes' and has a total of 1 service. BioCatalogue contains primarily Life Sciences related services, and provides 85 categories like protein synthesis, genomics and sequence analysis. XMethods has no categorization as all services are displayed as a list.

Other Taxonomy based Works. The major advantage of using standard taxonomies is that ambiguity can be prevented, however, the users have to be experts in the organization and hierarchical structure of the taxonomy to use it optimally. Some researchers tried to solve the issues in UDDI's taxonomy-based categorization methods by proposing enhanced methods. ShaikhAli et al. (2003) proposed an enhanced UDDI registry called UDDIe, that introduced the idea of *Blue pages* to hold user-defined parameters associated with their service. It also allowed users to define a *leasing period* for services published in the registry to specify the length of time the service advertisement should remain active. Corella et al. (2006c) proposed a heuristic system that aids the registry administrator by recommending a ranked list of potentially relevant classes where a new service fits better. This is done by comparing the new service with classified services already in the registry.

2.3.2 Web Service Categorization by Clustering

Clustering is a unsupervised technique for categorizing similar data items. Using Web service datasets, researchers have proposed different methods for clustering services and these can be majorly divided into five categories are discussed below.

Ontology based Clustering. An ontology is a representation or a taxonomy of concepts to which a domain conforms to. Identifying services that match a given domain ontology can help in efficiently clustering and categorizing them. Nayak et al. (2007) proposed a method called Semantic Web Service Clustering (SWSC) method, which uses the OWL-S service ontology. They used a small dataset of 35 services belonging to 8 different domains and applied hierarchical clustering to these to achieve 78% accuracy. Xie, Chen, et al. (2011) used the length of the path between two ontology concepts, the path weight, the density of concepts, and relationship between nodes in two service ontologies to determine similar services for clustering. Kumara et al. (2013) used a hybrid method based on ontology learning and term similarity to cluster similar services. If ontology concept matching fails, then Term frequency/inverse document frequency (Tf-idf) is applied to the service names to find functionally similar service clusters.

Data Mining based Clustering. Data Mining based techniques use concepts like association rule mining and text mining in categorizing Web services. Paliwal (2006) used hyperclique patterns based on frequent itemsets, for Web service clustering and discovery. The hyperclique patterns capture items that are highly associated with each other, the strength of which is given by its *support* and *h-confidence* values, using which functionally similar services could be determined. However, the authors did not provide any experimental results to support their claim.

Liu and Wong (2009) used text mining techniques and a two-phase, tree traversing ant (TTA) algorithm to cluster functionally similar services. In the first phase, the TTA algorithm is applied to the mined terms, where a term similarity measure called Normalized Google Distance (NGD) (Cilibrasi et al. 2007) partitions the term clusters, and in the second pass, a distance measure called n° to refine the formed clusters. Only 22 services were used for evaluating the proposed techniques and the authors reported good clustering results. However, for bigger datasets this technique cannot scale due to the computationally intensive nature of TTA algorithm. Elgazzar et al. (2010) used K-means and Quality Threshold (QT) clustering algorithm to cluster Web services using NGD. They too reported good results but the QT clustering algorithm is very computationally intensive and time consuming when compared to other traditional clustering algorithms.

NLP based Clustering. Web service descriptions are primarily written in natural language like English, as they are made of XML based elements. Also, the WSDL documents have a standard structure and contain elements like service name and documentation, which can be used for understanding its functional semantics. Some researchers like Ma et al. (2008), Paliwal (2007), and Sajjanhar et al. (2004) applied Latent Semantic Indexing (LSI) on the terms extracted from the service names and documentation. LSI is a statistical technique that captures the underlying semantics between a set of terms, as highly correlated words indicate the existence of a topic. Based on this, the domain of a service was determined and categorization was achieved as per some domain ontology. Ma (2007) proposed the use of Probabilistic LSI (PLSI) for improving the LSI model, however, failed to give adequate experimental analysis of their technique.

User-centric Clustering. Recently, a trend of involving users in actively contributing in the process of categorizing service collections exists, as evidenced in Web 2.0 applications. These approaches are based on the concept of social tagging and community filtering for generating metadata for services within large service collections. One such approach, augments social tagging and user-contributed data by using statistical techniques like Latent Dirichlet Allocation (LDA) to identify most relevant tags based on which services can be clustered (Chen et al. 2013a). A small collection of 185 services were used to verify the performance of the proposed approach, which resulted in better tagging accuracy. But, to be useful in real-world scenarios, the strategy has to be suitable for larger collections, which requires concentrated efforts of dedicated social users.

Chen (2010) proposed a collaborative filtering approach based on the nearest neighbour approach for Web service recommendation. Their idea was to cluster users to identify location-sensitive Web services, and then recommend services that were most suited to user needs. Maamar et al. (2011) proposed a novel idea by which service engineers can build a social network for their Web services which can help in discovering these services' peers. This proposes the use of two types of service social networks - *similar* (competition and substitution) and *different* (composition). The idea is that, during discovery, the similar peers can be used as either substitutes (in case original service fails) or as competitors (may have better QoS values etc). Similarly, in case of processes, the difference social network may be traversed to find possible composition candidates.

Bio-inspired Clustering. Nature-inspired methods have shown great promise for their varied applications in areas like load balancing, searching, path finding etc. The behaviour of social insects in nature has been studied and several algorithms have been proposed for solving categorization problems. Xu and Reiff (2008) presented an immune-inspired approach for clustering services as a preceding step to composition. The central idea of the immune-inspired clustering is to first specify the ideal candidate with the best fitness function. Then, during each iteration, a random individual is selected and if the fitness function is better than the original, the algorithm stops, or else continues till a better candidate is found, or the termination point is reached. This is clearly a very time-consuming process and is not suitable for large service collections.

Chifu et al. (2010) used an ant-inspired method for clustering services based on degree of match between input and output concepts. In nature, ants exhibit a highly selective cemetery building behaviour, where random ant workers start to pick up ant corpses and drop them in separate heaps. In time, these heaps start to merge and become bigger heaps. This behaviour is adapted to web services, by using the metric of degree of match. A set of artificial ants carry a service description each and create random heaps during the first iteration. This process is carried out for a pre-defined number of iterations and the resultant clustering is observed. The authors claim about 80% accuracy in the clustering, but they did not incorporate match dynamic thresholding to identify the best clustering. They note that if the threshold is too high, very few clusters will be formed and vice versa, if the threshold is too low. They have heuristically selected the threshold to be 0.1, at which the algorithm performed best.

The authors extended the work proposed earlier in (Pop et al. 2010) to incorporate dynamic thresholding based on the Intra-Cluster Variance (ICV) and Average Item-cluster Similarity (AICS) metrics. The number of iterations considered is quite high (2000 x Number of services considered), thus making the method very time consuming. They also observed that smaller iteration numbers led to large, low quality clusters, whereas, large iteration numbers resulted in smaller, better-defined clusters.

2.3.3 Web Service Categorization by Classification

Classification is a fundamental technique used to disambiguate between two objects by capturing their similar and dissimilar features. In general, classification is a supervised task, which means that, a new entity (*test data*) is categorized based on the experience obtained while classifying previous candidates (*training data*). Hence, classification is generally suitable for scenarios were certain ground truth values like manually classified services, or other similarly annotated datasets are available. Some classification based service categorization works in literature are discussed in this section.

Semantics based Classification. Semantics based approaches focus on capturing the inherent semantics and similarity between services to categorize them into functionally similar classes. Some approaches in this area are - semi-automatic technique based on heuristic matching between new services and already classified services (Corella et al. 2006c), using the textual description and annotation available in OWL-S advertisements for classifying services using the I/O relationships of *exact*, *subsume* and *plugin* (Katakis et al. 2009), semi-automatically classifying new services based on a manually classified dataset using machine learning techniques, which is discussed next.

Machine Learning based Classification. Machine learning based techniques are one of the common approaches in service classification. Saha et al. (2008) used tensor space model for service data representation. The tensor space model takes the structure of the XML based document structure of the WSDL, over which, a rough set based ensemble classifier for classifying services. Crasso et al. (2008a) used text mining to generate a training dataset by capturing the category information of published Web services and then used generic machine learning classifiers to categorize the services. AbuJarour, Naumann, and Craculeac (2010) used a pre-defined list of categories (obtained from ProgrammableWeb.com) to classify about 100 services. Similarly, Wang (2010) used the UNSPSC standard taxonomy to define the categories for their classification algorithm.

Ontology based Classification. This category consists of approaches that use ontology based concepts like concept hierarchies and lattices. Azmeh et al. (2008) proposed a technique that uses QoS based filtering to first select and then uses formal concept analysis (FCA) to classify similar services. The services are first arranged in partially ordered concept lattice, using a binary relation between services and operation signatures. Using this, services that share common attributes are identified and classified. The authors used only 5 services and the operations of all these services to show the viability of this approach and also note that the construction of a concept lattice, when the number of objects (services) and attributed (operation signatures) are large becomes very cumbersome and tedious. Aboud et al. (2009) used a similar approach that aimed at building various specialization lattices that offer human-readable views and also programmatically browsable indexes to search for suitable components.

Text Mining based Classification. Bruno et al. (2005) used text mining to automatically classify services to specific domains and then to generate additional metadata about services by identifying key concepts within each category. They used Support Vector Machines and Formal Concept Analysis in the proposed approach.

Laranjeiro et al. (2010) applied text classification algorithms like Näive Bayes and k-Nearest Neighbours to segregate services that were fail-safe and failure-prone in an attempt to analyse the robustness of service collections.

Heuristic/Fuzzy Classification. Heuristic methods employ several different parameters for classifying services like fuzzy logic (Chao et al. 2005; Corella et al. 2006b; Devis et al. 2008), rough set theory (Chen 2013b) etc. The primary goal is to capture the abstract definition of service functionality for the goal of concrete service categorization and discovery. Chao et al. (2005) and Chen (2013b) used fuzzy logic to capture QoS concepts like 'high', 'good', 'most' etc, which are relative measures and not absolutely measurable. Using fuzzy logic and semantics based processing, their work tried to address these issues, so users could use imprecise terms in their query and still get relevant results. However, this work lacked experimental results and the effectiveness of the approach could not be verified. Devis et al. (2008) proposed flexible service discovery framework which uses multiple matchmaking strategies, like domain ontology based, business domain modelling and logic reasoning. Corella et al. (2006b) also proposed a heuristic matching between category to service, service to service and concept to concept matching for correctly classifying services in an UDDI based setup.

2.3.4 Hybrid Approaches for Web Service Categorization

Some approaches in literature do not fall under the three particular categories identified, and so can only be termed as hybrid due to their fusion based methodologies. Some of them are discussed in this section. Fenza, Loia, et al. (2008) proposed a framework that uses fuzzy logic and agent based architecture towards similarity based grouping and matchmaking of relevant semantic Web services. The process of discovering services that can satisfy a given user request is delegated to task-specific agents that can identify even partial matches if no matches are found. This is because; the agents are modelled as per the fuzzy logic rules defined over the OWL-S descriptions of the services. In another paper, the same authors proposed an alternate hybrid approach based on the fusion of fuzzy logic and formal concept analysis that aims to automatically suggest service relevant terms to the user while querying the system, that may result in potentially relevant services (Fenza and Senatore 2010). An adhoc query can then be generated, without requiring strict adherence to formal syntax, to find recommendations on similar services.

Skoutas et al. (2010b) presented an aggregate ranking mechanism based on both functional and non-functional parameters of a service, which can be used to re-rank service discovery results based on user preference. In an extended paper (Skoutas 2010a), the authors proposed techniques that use multiple parameters and variable weights assigned to each of them to perform clustering and ranking of services with respect to a submitted user query. Multiple parameters that were considered include user preferences, QoS values etc and the underlying relationship between these parameters is dynamically measures using a set of logical inference rules in order to remove undue bias towards any one parameter.

Cao et al. (2013) proposed a novel approach that focuses on service discovery for consumers (from user's perspective) and consumer discovery (from service provider's perspective) using hybrid collaborative filtering algorithm. The approach considers the consumer-service data, service-provider data and provider-consumer data for generating recommendations for potential services that match client needs, and also potential clients for generating revenue for service providers. Platzer and Dustdar (2005) and Platzer, Rosenberg, et al. (2009) proposed a statistical clustering method using multi-dimensional angles as the proximity measure for clustering service documents represented in the vector space, aimed at efficient indexing to optimize querying.

2.3.5 Service Categorization Approaches - Summary

In summary, several varied techniques are available in the three main categorization techniques - taxonomy based, clustering based and classification based approaches. Based on techniques available in literature, each may be suited for different scenarios, and this is probably why, hybrid approaches have been proposed.

Taxonomy based approaches are more suitable for large scale collections where a standards based approach is used. This is why the UDDI and UBR adopted standard taxonomies, as the categories are well defined and hierarchically organized. However, this is also the biggest challenge, as the standard taxonomies have thousands of categories, and domain experts are required to correctly classify published services in the most fitting category for improved discovery accuracy. Some classification based approaches tried to use taxonomies for defining the categories for their classification algorithms, however, without extensive, properly classified service data as training datasets, these approaches have not achieved good accuracy. Also, classification approaches are not well suited for large, unlabelled, distributed service collections, as there may not be enough ground truth data against which the new services may be classified.

Clustering based approaches are well suited for the current scenario, where large amounts of services are available in a distributed fashion and decentralized approaches like service search engines and portals exist. Clustering, being an unsupervised approach, is a technique that is meant to capture inherent similarity features for grouping similar entities, hence, clustering based distributed Web service categorization techniques may perform better and also scale better than the other two approaches. However, traditional clustering algorithms have several limitations and can be computationally intensive. Also, any changes in the service dataset will require the clustering process to restart once again; hence incremental approaches may be needed for real-world applications like service search engines and portals.

In the next section, another major issue, that is, extending semantics based querying support to service consumers is discussed. Most existing systems have limited support for context-sensitive handling of service discovery requests, which itself can affect the accuracy of the services retrieved, if ambiguity is not resolved. Section 2.4 presents a review of existing techniques for better understanding a user query.

2.4 Understanding User Requirements

As the volume of data on the Web keeps increasing, one of the primary problems reduces to how to effectively find it, using the unstructured queries that users provide. One of the primary problems faced by service based application designers is the primarily keyword based queries and the lack of expressiveness in the querying interfaces provided by traditional service repositories like service registries and portals. Hence, service discovery techniques need to adopt intelligent techniques to process user requests to understand their context and requirements.

One of the easiest ways of communicating with search engines and other search based systems has been one's natural language. Currently, multilingual search is also supported by most major search engines. The advantage of a natural language querying interface to any system is that most users can use it intuitively, without any need for additional structuring to their request. However, in order that the system understands the query submitted by the user, the query may have to processed and transformed into a structured query for best matching.

Context-awareness in general has been defined as - *"any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves (Abowd et al. 1999)"*. In Web service discovery, context-awareness can refer to several factors, as is evidenced in current literature. In mobile service discovery scenarios, context-awareness refers to personalization based on user device type, mobility, preference etc, before identifying the best service to perform the task (Doulkeridis et al. 2006) (Sheshagiri et al. 2004). In pervasive environments, this might mean considering user's location and environment (Mokhtar et al. 2008; Pawar et al. 2006). In dynamic service invocation scenarios, this means a preference to adaptability and robustness, to ensure task completion (Hong et al. 2009; Spanoudakis et al. 2007). In case of user-centric query processing systems, context-awareness refers to understanding user requirements specified in an unstructured manner, to correctly identify the desired service domain and recommend most relevant services (Paliwal 2012; Sangers et al. 2013). The problem we are trying to address in this thesis pertains to the last scenario and we focus the discussion in this area.

Most current literature in Web service discovery discussed in previous sections, tried to address the issues of effectively capturing service functionalities for relevant service discovery. For best matching, the same analysis has to be applied to the user request too, especially in the case of natural language queries. Several researchers tried to address these challenges and relevant works can be subdivided into three main categories as shown in Figure 2.6.

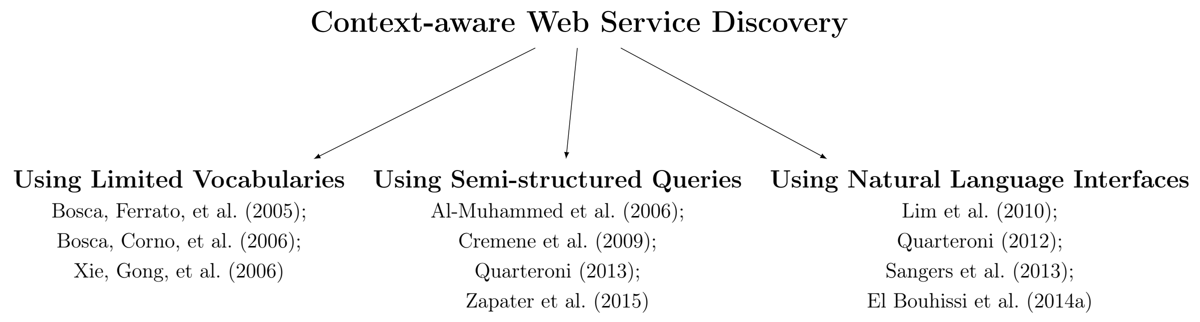

Context-aware Web Service Discovery
Using Limited Vocabularies
Bosca, Ferrato, et al. (2005);
Bosca, Corno, et al. (2006);
Xie, Gong, et al. (2006)
Using Semi-structured Queries
Al-Muhammed et al. (2006);
Cremene et al. (2009);
Quarteroni (2013);
Zapater et al. (2015)
Using Natural Language Interfaces
Lim et al. (2010);
Quarteroni (2012);
Sangers et al. (2013);
El Bouhissi et al. (2014a)

Figure 2.6: Context-aware Web Service Discovery - Approaches

2.4.1 Using Limited Vocabulary Requests

Some researchers used a restricted vocabulary of keyword set with which users could express their requests, for example, explicitly defined input and output parameters. These approaches can be categorized as limited vocabulary based approaches. Bosca, Ferrato, et al. (2005) proposed a technique that used a controlled subset of natural language that was to be applied to a closed, limited set of services like those managed by a single operator or at a company's registry. Each of the services are annotated with the applicable vocabulary terms as per their domain, which allows effective categorization and retrieval. They extended this approach to support service compositions where the each controlled user query is processed for lexical constructs designed to convey the operations' semantics, in order to recognize and extract fundamental functional requirements implied by the request, and associate them to entries in the service catalogue (Bosca, Corno, et al. 2006). Xie, Gong, et al. (2006) used a technique that used a vocabulary called SOBL (Semantic Object behaviour Language) to transform the user's natural language request to a formal specification that can be used to find services that can satisfy the request.

2.4.2 Using Semi-structured Queries.

Approaches that focus on imposing some structure on user queries, for easier handling and processing come under this category. Al-Muhammed et al. (2006) proposed the use of certain constructs for limiting the expressiveness of natural language queries and adding some structure to free form queries. They used these constructs when the number of solutions presented by the system are too many *(under-constrained query)*, or when no solutions are presented at all *(over-constrained query)*. Hence, in the case of over-constrained queries, the proposed approach identifies those conditions that need to be removed, so some results could be returned. Similarly for under-constrained queries, a few additional conditions are suggested to the users, so unrelated results can be pruned and to retain the best-m solutions only. Using this technique, the authors developed a dialogue based system to capture user requests, where the system is able to give feedback to the users based on the problems noticed during the result generation process.

Cremene et al. (2009) used a pre-defined template based approach, by which users can specify their requests for candidates for service composition in a structured manner. The authors claimed that the system was 100% accurate when used in a simulated, intelligent home environment. However, it is not scalable and is applicable only when a known set of services is used, that can be automatically invoked. Quarteroni (2013) processed the repository of services first to identify their functionality and to annotate the services with the associated metadata during registering the service itself. The queries are submitted to the system in free form, but are processed, segmented, classified, and mapped to an execution strategy modelled by the service domain metadata to perform matchmaking. Zapater et al. (2015) developed a travel information system using semantic Web services that supports semi-structured queries. Users can specified their travel requests using pre-defined constructs like *to, from*, *via*, *through* etc, which are considered connecting points to parts of the query, using which the query can be segmented. Various services are then invoked to satisfy the user's request. Again, these types of approaches are suitable for closed domain datasets who characteristics are known and can be easily modelled.

2.4.3 Using Natural Language Interfaces.

Dealing with primarily unstructured service repositories is a highly challenging task due to the large number of services involved and their varied domains. Natural language interfaces (NLIs) support completely free form requests that allow users to query the underlying repositories without needing to know domain specifics. Most user-focused applications like popular Web search engines like Google, Bing and Yahoo, provide NLIs, which help users in conducting their search activities in a very intuitive manner.

Lim et al. (2010) proposed a technique to automatically generate an abstract service workflow from a user's natural language complex request. The generated workflow also specifies all the constituent tasks and their transitions such that multiple services can be identified for each of the tasks. Quarteroni (2012) designed techniques for identifying the important focal terms in user queries. Once identified, these are classified as per the service taxonomy and then used for extracting additional relevant information in order to match them to suitable services. Sangers et al. (2013) used common natural language processing techniques like named entity recognition, anaphora resolution and word sense disambiguation for correctly understanding the context of the user's request for matchmaking with WSMO described services. El Bouhissi et al. (2014a) and El Bouhissi et al. (2014b) used the concept of user goal modelling using ontologies, that is used to match and discover services. Here, the user request is in the form of a goal that is to be achieved, which is annotated with ontology terms, to effectively match concepts with the services in the repository.

2.4.4 Semantics-based Querying - Remarks

Effectively understanding the user query can enhance the service discovery process to a large extent as unambiguity can be eliminated and the accuracy of search results returned can be boosted. Of the available techniques, natural language interfaces are most flexible and support a completely intuitive querying from the user perspective, thus improving ease of use and user experience in the process. In Section 2.5, a review of existing techniques for specifically discovering composite Web services in literature are presented.

2.5 Composite Web Service Discovery - Review

It is a fact that most applications need more than one service to offer the intended functionality. Hence, application designers have to perform the critical task of identifying the best services for a particular task in the process workflow. For this, they try to discover available services for each individual subtask, manually. To automatically identify those services that can invoked in the required sequence to achieve the desired functionality is a much sought-after feature.

A common example is, an application that offers customers a facility to plan an entire holiday by booking flight tickets as well as hotel accommodation while taking into account various parameters such as family activities, prices, special offers, and so on. A major requirement for supporting such a feature is also identifying the sequence in which each of the constituent services have to be invoked (or chained in a workflow) to get the desired results. Table 2.4 shows a scenario where four services {WS1, WS2, WS3, WS4} have to be invoked in order to satisfy the user's requirements.

Table 2.4: Service composition required to serve the query *"doctor to treat high fever and chills"*

Service no.	Input(s)	Output(s)
WS1	list of symptoms	disease information
WS2	disease information	recommended treatment
WS3	name of treatment, zipcode	list of hospitals
WS4	nearest hospital	list of doctors

In literature, very few works focus on the problem of composite Web service discovery. This is because constraint based service composition is a NP-hard problem that requires extensive knowledge of the available service domain and service capabilities; and hence is computationally intensive. Existing approaches in the area of composition-oriented Web service discovery can be categorized under three groups as shown in Figure 2.7.

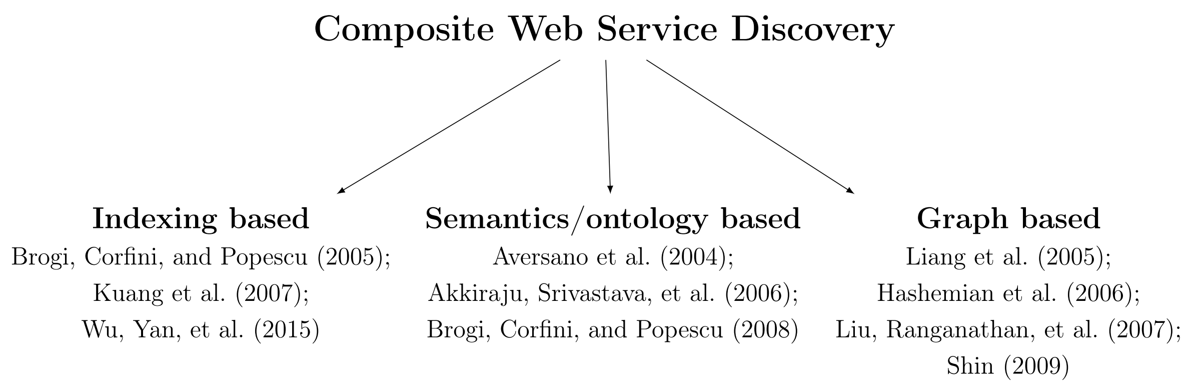

Composite Web Service Discovery
Indexing based
Brogi, Corfini, and Popescu (2005); Kuang et al. (2007); Wu, Yan, et al. (2015)
Semantics/ontology based
Aversano et al. (2004); Akkiraju, Srivastava, et al. (2006); Brogi, Corfini, and Popescu (2008)
Graph based
Liang et al. (2005); Hashemian et al. (2006); Liu, Ranganathan, et al. (2007); Shin (2009)

Figure 2.7: Composite Web Service Discovery - Approaches

2.5.1 Indexing based Composite Service Discovery

Indexing based approaches use the information contained within the service descriptions, extract it and store it within an index for fast retrieval during the process of discovery. Brogi, Corfini, and Popescu (2005) proposed an algorithm called Service Aggregation Matchmaking (SAM) that indexes OWL-S process models of services in the repository as a tree structure and stores it in the memory. Then, for a given query, SAM can perform a fine-grained matching at the level of simple processes, and can also return a list of partial matches, when no full match is possible. Besides returning partial matches, it also suggests to the client additional inputs that would help in getting a full match. No experimental results were presented for verifying the validity of their approach.

Kuang et al. (2007)'s approach is based on using an index of OWL-S described outputs of registered services. A service list is maintained for each output that contains all the services that deliver that output, in a given repository. Thus the output acts as the key of the sorted index, so that the matchmaking with the query is optimized. If found, only those services on the identified output list are considered relevant, and so, others can be filtered out. The authors claim that this method is faster than sequential composition-oriented discovery, however, the amount of manual work required while first creating the index and the service list for each output is quite high, so this method may not be suitable for large service repositories.

Wu, Yan, et al. (2015) used a multilevel index based model for processing service output information, to overcome the redundancy problem found in sequential and inverted index methods. They proposed a 4-level indexing scheme, where the first two levels are based on the extracted service inputs and outputs. The next two levels use a concept called as a 'key' which is used to eliminate redundancy from the indexed space. This multi-level indexing representation speeds up the process of finding service composition candidates by matching terms in the first two indexes. The indexing scheme can be used over existing service registries for published service management by using four basic operations provided - retrieve, insert, delete and replace, just like how relational databases are managed. The authors presented a theoretical analysis of the approach, to prove that redundancies can be effectively eliminated, but no experimental results were provided in this paper to support their claim.

2.5.2 Semantics/Ontology based Composite Service Discovery

Ontology based approaches exploit semantic Web formalisms like OWL-S and WSMO for discovering services for composition. In Aversano et al. (2004)'s approach, the services are represented using OWL-S ontology. If a single service cannot fulfil the requested input and output descriptions, then services that fulfil at least one output are searched for. If a discovered services can fulfil the output, but not the original input, then the process is repeated by searching for new services, that fulfil the new input as output, until the original request is satisfied through discovered service composition. Hence, due to the recursive search the proposed algorithm is polynomial in complexity.

Akkiraju, Srivastava, et al. (2006) proposed a Web service discovery and composition technique based on semantic matching and AI planning. To understand ambiguous terms a combination of domain-independent and domain-specific ontologies were used to facilitate correct service identification. Brogi, Corfini, Aldana, et al. (2006) extended their previous index based approach for semantics based composition-oriented discovery, to enhance the algorithm SAM (Brogi, Corfini, and Popescu 2005) proposed earlier, to also find potential, multiple executions of services, to provide alternate composition plans to users. They intended SAM to be part of UDDI registries, to enhance service discovery. In SAM, the discovery takes place on the server-side and the actual composition process is performed on the client-side by employing the service list returned by the algorithm. However, the generation of the service list in the extended version is based on the concept of a hypergraph. In a hypergraph, an edge (called an hyperedge) can connect any number of vertices, thus the graph construction process and the result generation becomes polynomial in complexity.

2.5.3 Graph based Composite Service Discovery

Graph based approaches redefine the Web service composition problem as a graph search problem by representing the services are nodes and the edges are the dependencies between the services. Most of the approaches are intended to be deployed at a intelligent service registry/server, which implements the algorithm required to represent all indexed services as a graph.

Liang et al. (2005) presented a novel graph based service composition template generation technique. For a given query, possible composition sets are found by performing a graph traversal using the AND/OR graph search algorithm. Once an ideal template is selected, the system then attempts to bind the template's service operations to registered services. If the binding is successful, the bound template is used to generate a WSFL document (Web Services Flow Language), which can be used by a WSFL execution engine to invoke the component services, as per user command.

Hashemian et al. (2006) represent available Web services as an input/output graph. The behaviour of basic services is studied and the expected behaviour of a composite service is modelled as a 'process algebra', which is generated by a Web service composition planner component. Using the I/O graph and the generated process algebra, a search for a suitable composition is carried out and if found is returned to the user in the form of a BPEL process. This approach was based on the forward chaining process, due to which cycles may sometimes appear, which can result in incorrect composition templates. Also, the complexity is a polynomial function dependent on the size of the graph and number of inputs/outputs.

Liu, Ranganathan, et al. (2007) method consists of modelling services using a semantic graph and the connections between services as transformations. They also annotate each operation of the service with additional metadata using RDF graph patterns, so inputs and outputs are explicitly expressed, in its domain. The domain itself is defined using OWL-S ontologies. Using this setup, a composition model is proposed that considers the user given conditions before generating an abstract workflow. A planner engine uses this model to automatically compose services based on the abstract workflow. The authors reported that, while the pre-reasoning approach was quite time consuming, it can result in multiple composition plans based on the identified matching.

Shin (2009) also considered the functional semantics of services while determining the correct sequence of services for satisfying a user query. WSMO was used to explicitly define the functional semantics and category of a service. Each service is stored as an *object* and *action* pair in a two-layer graph model, which is traversed to find possible matching object/action pairs. This approach considered only a single action per object (WSMO service), however, there may be multiple associated actions, which can lead to ambiguity and additional processing time. Hence, the composite service discovery process is of polynomial complexity, and is a function of the graph size and number of inputs, due to which it is not very scalable. Since the matching is semantics based, this paper can also be considered under the semantics based composite service discovery category.

2.5.4 Composite Service Discovery - Remarks

As previously stated, few composite Web service discovery techniques exist in published literature. Available approaches use different methods like indexing, graph construction and semantics for identifying composite services. Most have high complexity and are time consuming, which gets worse as the number of services considered increases. Indexing based methods focus of using the terms extracted from the service inputs and outputs for 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 is not possible using 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 due to which computationally intensive techniques like AI planning and logic based reasoning have to be used. Graph based techniques are the most efficient as they focus on representing the available services by their dependencies, and then use various graph traversal problems. With the inclusion of semantics based processing and intelligent indexing, graph based algorithms can be made very efficient. Some problems faced by existing graph based approaches while searching for possible compositions, like cycle detection and avoidance, graph maintenance, and optimized traversal techniques need solutions and there is scope for much improvement in these areas.

2.6 Research Directions

A review of published literature in the area of Web service discovery shows an inclination towards semantics based techniques for enhancing the performance of service discovery. We presented a comprehensive review of these approaches and compared the categories in each area based on several important parameters. Some of the important research directions that need to be pursued to further optimize service discovery related tasks are presented here:

- Earlier service discovery approaches relied on the public/shared business registries like UDDI and UBRs which are no longer available. Also, most practical SOA implementations currently prefer a direct interaction between requester and provider (AbuJarour and Naumann 2010; Michlmayr et al. 2007), due to which, a central point of access to published services is no longer available. Some surveys on public web services have reported conclusively that the recent trend observed in the domain of service discovery is increasingly towards information retrieval based approaches to find services over the Web (Fan et al. 2005; Al-Masri 2009; Al-Masri et al. 2008). *Hence, there is definite scope for applying intelligent information retrieval and semantics based techniques to build a large-scale, distributed service discovery framework.*

- The standards based UDDIs and UBRs failed primarily due to their dependency on service providers to actively manage their published service descriptions. Unless these providers keep their service data updated, it was impossible to maintain the repository. *There is scope for incorporating intelligent processing techniques that can minimize this dependency on humans to update and maintain their data.*

- The UDDI and UBRs did not enforce any quality checks on the services published in their repository (Almasri et al. 2008). Due to this, invalid and incomplete service descriptions could also be published in the registry, thus affecting the integrity and standard of the service collection. *It is critical to enforce strict quality checks on the service descriptions to be indexed within a service repository and there is scope for the development of such measures.*

- There are no explicit version control and duplicate management checks in the UDDI/UBRs and also in any of the existing service search engine approaches. These are essential, especially in the current scenario, as same services may be published in multiple, varied sources on the Web. A service provider company may make its services available on its own website, and then may decide to advertise these services on several third party service portals. Also, a new version of an existing service may be released, which supports more (or lesser) functionalities. When these are indexed by a distributed discovery mechanism, if each copy is treated as a new service, then the service repository will be highly redundant and unproductive. *Hence, effective version control and duplicate detection mechanisms need to be developed for efficient registry management.*

- Some surveys reported that only about 15% of the services published on the Web have good quality natural language documentation. The vast majority of services on the Web exist without proper natural language documentation to effectively indicate their capabilities to a service consumer. As most existing functional aspects based discovery approaches deal with synthetic service datasets which consist of well documented service descriptions, these have failed to handle situations when such good documentation is not available. *Hence, there is scope of developing alternate methods to capture the functional semantics of a service, without relying exclusively on the service provider's documentation, as is the case with several existing works.*

- Most approaches for distributed Web service discovery deal with small synthetic or real-world Web service datasets. This means that their service collection is essentially static and does not change over time. Dealing with such a static dataset is straight forward and effective categorization can be achieved using traditional approaches (discussed in Section 2.3). However, large scale service discovery will have to deal with the massive collections of services available on the Web. If these are to be found and indexed in a central repository using a distributed discovery approach, such a framework will have to deal with constantly changing service collection. There may be a newer service version available, older services may no longer be offered and may have been removed from their servers. A distributed service discovery framework must be able to deal with these constant changes and be able to effectively categorize services incrementally, rather than having to restart after each change. *There is scope for intelligent categorization algorithms that can deal with constantly changing, dynamic service collections effectively.*

- Often, the results of a service discovery process are dependent on the terms used by the user during the search process. As users may not be aware of all the domain-specific terms required for optimal results, *it is beneficial to extend context-sensitive querying support to users.*

- As discussed in Section 2.5, very few works have focused on the problem of discovering relevant services for composition (Composition-oriented discovery). Of the existing works, most have very high complexity and are not easily scalable to larger service datasets. *Hence, there is scope for developing semantics based composition-oriented discovery techniques for recommending composite service to application designers.*

Considering these observations, several possible research gaps have been identified in existing literature, which has provided some promising research directions that are yet to be explored. These include issues pertaining to distributed Web service discovery and the limitations observed in existing literature. Possible solutions are offered by the inclusion of semantics, enabling domain-specific categorization and semantic querying support during basic and composite service discovery.

2.7 Summary

In this chapter, we examined the basic concepts and the state-of-the-art in published literature in the area of Web Service Discovery. We classified Web service discovery approaches as centralized, decentralized and hybrid, and under each, discussed various techniques used for service discovery, ranging from traditional keyword based matching to social network based, user-centric discovery. In the current scenario, the type of architecture that is most suited is a *Hybrid Architecture* that encapsulates distributed algorithms for finding and indexing published services, while supporting a centralized repository as a point of access where user requests can be articulated. There is also a need for efficient semantics based similarity analysis, indexing and categorization within the repository to support both basic and composite service discovery, based on natural language requests.

In the next chapter, we formulate these research issues in a formal manner and present the solutions proposed to identified research directions in the subsequent chapters of this thesis. We believe that, Web service discovery being the most critical task in the Web service life cycle, has a deep impact on the success and failure of a SOA based application, and that, the work presented in this thesis will make a positive contribution to research in this area.

Publications

1. Sowmya Kamath S and Ananthanarayana V.S, *"Semantic Web Services Discovery, Selection and Composition Techniques"*, at the Third International Conference on Computer Science and Information Technology (CCSIT 2013), pgs 151 -158, 2013. DOI : 10.5121/csit.2013.3616.

2. Sowmya Kamath S and Prakash S. Raghavendra, *"Semantic Web - Applications, Challenges and Directions"*, at the 3rd International Conference in Information Technology and Business Intelligence (ITBI 2011), November 25 - 27th, 2011, Hyderabad, India

Chapter 3

Problem Description

3.1 Problem Definition

Web service technology promotes component reusability through loose coupling in application to application interactions. The success of the Web service based SOA systems and dependent on effective solutions for prevalent issues in critical Web service life cycle tasks like service discovery and composition. In this regard, these prevalent issues in distributed Web service discovery were discussed and summarized in the previous chapter. In this chapter, we describe the scope of this thesis and formally define the identified issues for consideration as an investigation problem. We also briefly describe the overall methodology adopted for the research work presented in this thesis.

3.2 Scope of the Work

Web service discovery is a fundamentally important task because of its pervasive nature in all Web service based SOA systems. Having been investigated extensively in the past, the current scenario warrants a fresh look at the problem of finding services distributed over the Web. Current Web service conceptual model does not support distributed, semantics based discovery on the large scale. The most critical issue compounding the discovery problem further is the exponential growth in the number of published Web services over the Web and their diversity.

In this thesis, we discuss the problem of effectively finding distributed Web services from heterogeneous sources over the Web. For this purpose, a distributed Web service discovery mechanism is required that incorporates certain intelligent techniques for enabling semantics based service discovery for service consumers. The main emphasis is given to addressing the following identified problems:

1. **A distributed mechanism to identify services published in varied sources over the open Web and retrieve them.**

   The retrieved services would be indexed using intelligent processing techniques for best retrieval performance. The framework should support automatic version management, duplicate detection, and minimize the level of human intervention in managing the service data. Another major requirement is strict adherence to quality checking to ensure the integrity of the repository. This framework would support service discovery using the well-organized service repository. The main idea here is that a service consumer can use the framework's user interface for articulating service discovery requests. Figure 3.1 depicts the distributed approach of finding and retrieving published services from the Web.

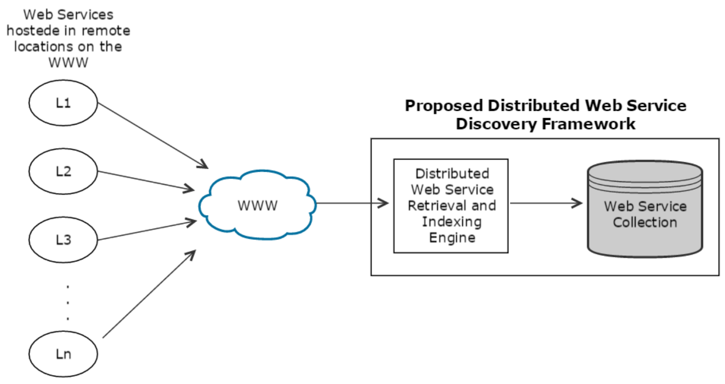

Web Services hostede in remote locations on the WWW
L1
L2
L3
Ln
WWW
Proposed Distributed Web Service Discovery Framework
Distributed Web Service Retrieval and Indexing Engine
Web Service Collection

Figure 3.1: Process of building the service repository

2. **Using available data to generate additional metadata and semantic annotation for service descriptions.**

   As discussed earlier, more than 85% of published services are without proper natural language documentation, which hinders the correct identification of the domain of a service. This is addressed by incorporating effective mechanisms for capturing the functional semantics of a service using natural language processing techniques. The extracted functional semantics is used for service similarity computation and for representative metadata generation in the form of natural language tags, which is shown in Figure 3.2.

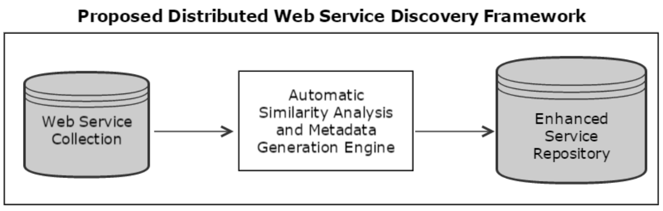

Proposed Distributed Web Service Discovery Framework
Web Service Collection
Automatic Similarity Analysis and Metadata Generation Engine
Enhanced Service Repository

Figure 3.2: Automatic metadata generation for services

3. **Incorporating semantics based categorization techniques for the large, dynamic service collection in the repository of the proposed distributed service discovery framework.**

   Most existing approaches for Web service discovery use small Web service datasets, which are essentially static. Hence, traditional clustering and classification algorithms are adequate. However, the number and diversity of services indexed within the proposed framework will keep changing constantly and effective categorization is required to deal with these changes. This is addressed by using a semantics based dynamic categorization algorithm (as seen in Figure 3.3) that can deal with multiple changes over time incrementally, without needing to start from scratch each time a change occurs.

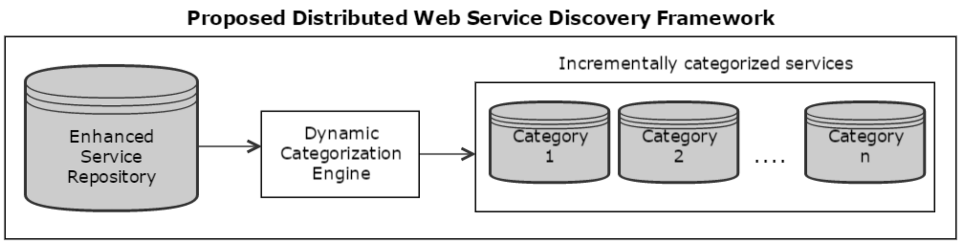

Proposed Distributed Web Service Discovery Framework
Enhanced Service Repository
Dynamic Categorization Engine
Incrementally categorized services
Category 1
Category 2
....
Category n

Figure 3.3: Process of dynamic categorization

4. **Adding context-sensitive querying features for the service discovery framework to support semantics based service discovery.**

   To compensate the service consumer's lack of complete domain term knowledge, processing the service request semantically, with emphasis on context will help in correctly understanding user requirement and generate relevant results. This problem is addressed by integrating a semantics based query processing mechanism designed to capture the correct context automatically (depicted in Figure 3.4).

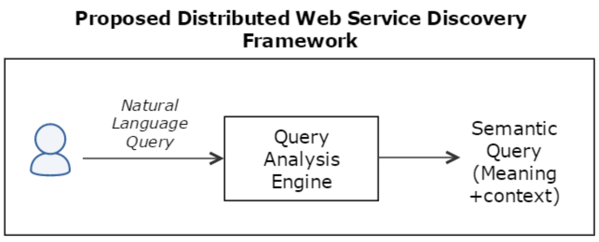

Proposed Distributed Web Service Discovery Framework
Natural Language Query
Query Analysis Engine
Semantic Query (Meaning +context)

Figure 3.4: Context-sensitive processing of user query

5. **Using the proposed framework and its semantic processing mechanisms for enabling basic and composite service discovery functionalities.**

The main requirement for simple service requests are identifying the domain terms and matching these to service metadata. For composite services, complex requests have to first correctly interpreted, and then the constituent services have to be identified. The sequence in which these services have to be chained also would have to be identified to get a correct and valid result, so user can perform composition using the identified services. This is done by using semantics based graph representations, for capturing the service dependencies between services, to identify candidates for serving complex requests, fast. This process is illustrated in Figure 3.5.

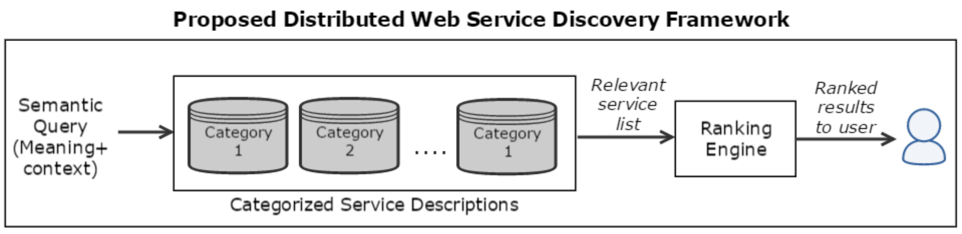

Proposed Distributed Web Service Discovery Framework
Semantic Query (Meaning+ context)
Category 1
Category 2
....
Category 1
Categorized Service Descriptions
Relevant service list
Ranking Engine
Ranked results to user

Figure 3.5: Basic and composite service discovery

With regards to the scope of the work presented, our main contributions, discussed in subsequent chapters of this dissertation are:

- Developing a framework for distributed Web service discovery, for collecting, retrieving, and indexing services already available on the Web.
- Automatic generation of representative metadata in the form of tagging and semantic annotations for indexed services using which the discovery process was enhanced and better precision and recall were observed.
- An automatic incremental algorithm for semantics based categorization of the dynamic service collection, which further enhanced the discovery process by reducing the time taken for generating matches.
- Query analysis algorithm for capturing the context and meaning of the user request.
- A service interface graph traversal algorithm for capturing service dependencies that helps in identifying the constituent services of a valid composition as well as the sequence in which they should be composed, for a given complex query.
- Demonstrating that the proposed framework and techniques are amenable for semantics based basic and composite service discovery.

3.3 $\mathcal{DWDS}$ - $\mathcal{D}$istributed $\mathcal{W}$eb Service $\mathcal{D}$iscovery Framework using $\mathcal{S}$emantics

The overall system architecture of the proposed $\mathcal{DWDS}$ framework is shown in Figure 3.6. It consists of two phases - the *pre-processing phase* and the *discovery phase.* In the pre-processing phase, the distributed Web service discovery and repository building tasks are carried out, while the discovery phase is where a user can articulate a natural language query and discover basic or composite services as per the query requirements. The processes during each of these phases are briefly described here.

Phase I: Preprocessing Related Activities.

1. **Retrieving services from the Web.** *(discussed in Chapter 4)*
   - The distributed discovery mechanism is employed for finding services available in various sources on the Web.
   - Services are added to a preliminary database called WSDL-Preprocessor after quality checks.
2. **Metadata Generation Engine.** *(discussed in Chapter 5)*
   - Here, the service WSDLs are semantically processed to capture the inherent functional semantics of the services. This is used to compute similarity between services and to generate a set of representative tags for each service.
   - Each service is indexed in the Enhanced Service repository using its metadata.
3. **Dynamic Categorization Engine.** *(discussed in Chapter 6)*
   - Using the similarity scores and the generated metadata, a dynamic categorization algorithm is employed to achieve domain-specific grouping.
   - Using the metadata of the member services of each group, tags that most represent the group's domain are generated for each category, to capture domain information.
   - As the framework tries to find new services continuously, the service collection keeps changing constantly. This is because, any new services found will be added and already indexed services ones that are no longer accessible will be removed from the service repository. The proposed dynamic algorithm aims to dynamically categorize these new dataitems without needing to restart the categorization process again.

Phase II: Service Discovery Related Activities.

1. **Query Analysis Engine.** *(discussed in Chapter 7)*

   - The natural language query submitted by the user does not have any structure to it. This free form query is processed using NLP techniques to identify the meaning and context of the query, to generate a structured, semantic query.
   - The process of identifying if the submitted query is simple or complex is also performed at this stage.
   - The semantic query (simple or complex) is submitted to the system, which invokes the discovery algorithm, and performs different actions based on the type of query.

2. **Basic Web Service Discovery.** *(discussed in Chapter 7)*

   - In case of a simple query, the task is to identify basic services that match the *meaning+context* of the query, and the matching is above a set threshold. A list of results generated is passed to the Ranking Engine.
   - The Ranking Engine determines the top-k services with similarity scores above a certain threshold, sorts them in descending order and displays the ranked list to the user.

3. **Composite Web Service Discovery.** *(discussed in Chapter 8)*

   - In case of a complex query, the task is to identify multiple basic services that can together satisfy the requirements of the complex query, when used in the correct sequence. A graph based approach that uses semantic techniques to capture the service dependencies is integrated with the complex query serving module, which is used to identify the constituent services for a valid composite service. All such found templates with a score above a predefined cut-off are submitted to the Ranking Engine.
   - Next, the Ranking Engine determines the top scoring complete composite service templates and generates a ranked list of these. After this, any partial templates identified that matched more than 75% of the user requirements are also considered and a ranked list of partial templates is generated. The complete and the partial template ranked lists are returned to the user.

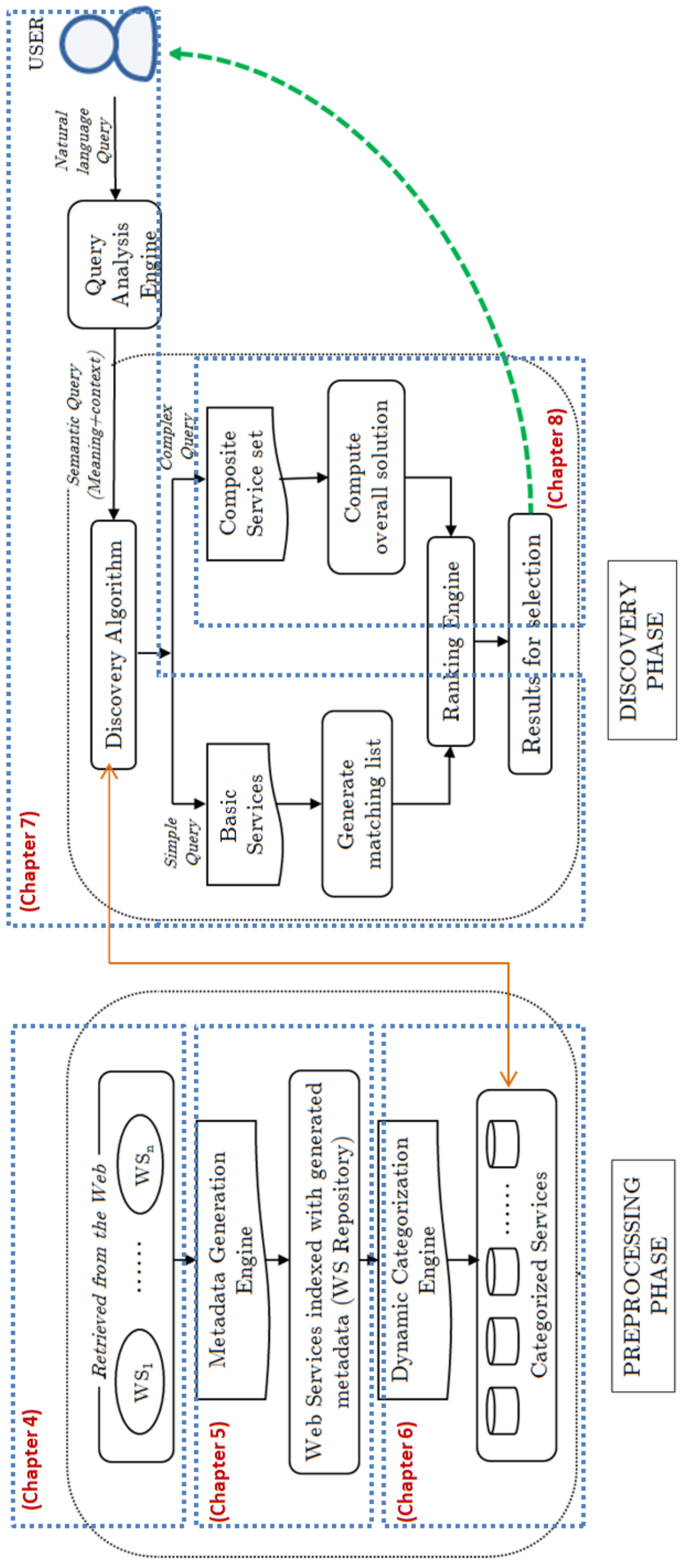


Figure 3.6: Proposed Methodology for $DWDS$ Framework

3.4 Summary

In this chapter, the scope of the research work and the identified problems that are addressed in this thesis are formally defined. Our contributions, presented in this thesis, address four important issues. First, we present a framework for efficiently finding, retrieving and indexing distributed Web services using automatically generated metadata, to build a Web service repository. Secondly, an efficient algorithm that can semantically categorize services as per their domain, even when there are constant changes in the service collections is presented. Thirdly, effective techniques for generating a structured, semantic query from a free-form, natural language service request to capture the meaning and context of a user is presented. Fourthly, a Web service discovery mechanism that can discover both basic and composite services based on user requirements is presented. The proposed system and techniques have been developed and experimentally validated. The observed results were found to be very good and the framework successfully supported semantics based distributed Web service discovery.

Publications

(based on the work presented in this chapter)

1. Sowmya Kamath S., and Ananthanarayana V. S. "A bottom-up approach towards achieving semantic web services.", In Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on, pp. 1317-1322. IEEE, 2013.

PART II

Building a Scalable Web Service Repository

Chapter 4

Finding Distributed Web Services

4.1 Introduction

In this chapter, we propose a distributed Web service discovery framework as a solution to the problems formally defined in the previous chapter. The main focus is on automating the tasks of finding, retrieval, categorisation and management of published services available on the Web, and also to minimize human involvement. Several researchers have underscored the need for automation in Web service life cycle tasks for sustaining the popularity of the Web services paradigm (Cheng et al. 2008; Issarny et al. 2011). They put forth the concept of enabling autonomous behaviour in service infrastructures for reducing management costs and effort. As the number of services on the Web keeps increasing, there arises a requirement for enabling automatic collection of such published services, to make them available for discovery and use. Hence, a scalable Web service repository is critical to supporting large-scale, distributed Web service discovery. In this chapter, we discuss the proposed techniques for developing such an infrastructure. Our main contributions, as reported in this chapter, are:

- Developing a framework for distributed Web service discovery to find and retrieve published services available in varied sources over the Web.
- Demonstrating that the framework successfully performs large scale service retrieval and enables a scalable service repository.
- Verifying that the framework minimizes human intervention by incorporating certain autonomic behaviour for quality and redundancy control.

The rest of this chapter is organized as follows. Section 4.2 describes the defined problem addressed in this chapter. Section 4.3 discusses the solution methodology used for finding and retrieving published services from their remote sources on the Web. Section 4.4 describes the mechanisms designed and incorporated for enabling autonomic features in the proposed framework. Section 4.5 presents the experimental results and analysis with reference to the framework presented and Section 4.6 summarizes the discussion presented in this chapter.

4.2 Problem Statement

The problem that is addressed in this chapter is defined here:

> *Given the large number of published services available from heterogeneous sources on the Web, find and retrieve these distributed services, to build a centralized service repository that incorporates autonomic features for system management, to minimize human intervention.*

The solution proposed for this problem, aims at building a Web service repository that uses - (i) a distributed approach to build the repository and, (ii) a centralized approach to maintain the service collection. Hence, it is a hybrid architecture that aims to mitigate the current lack of a central point of access to Web services. The solution also aims to reduce human intervention in repository management by incorporating self management behaviour into the framework. This eliminates the need for manual effort in the management and maintenance of the service collection, thus enhancing productivity. In Section 4.3, we describe the algorithms developed for finding and retrieving distributed Web services from varied sources on the Web.

4.3 The Proposed Framework - $\mathcal{DWDS}$

The proposed framework, $\mathcal{D}$istributed $\mathcal{W}$eb Service $\mathcal{D}$iscovery Framework using $\mathcal{S}$emantics ($\mathcal{DWDS}$), is intended to find services published in heterogeneous sources on the Web. Based on various published literature (Fan et al. 2005; Kim et al. 2004) and from our own study, public Web services are currently available from three main sources.

- Web servers of service providers, where the services are hosted.
- Embedded within Web pages and HTML forms.
- Third party service portals where services are often advertised by service providers.

Traditional IR techniques used by search engines use keyword based indexing to match webpages to user queries. The crawler of a search engine visits a web page, reads it, and then follows any links to other pages. The pages found are added to the search engine index, indexed by the keywords found in that webpage. Then, a ranking algorithm is used to search through millions of webpages to rank the results as per a submitted user query. Hence, in general, traditional web crawlers utilize the graph structure of the Web to move from one page to another.

When applying Web crawling techniques for finding services, several problems need to be addressed. Traditional Web crawlers are meant for Web pages and are not suitable for finding services as -

1. Webpages contain lot of textual information whereas a Web service often has a short textual description.
2. Webpages contain a lot of plain text while Web service description is a XML marked-up document. So IR methods like Term Frequency/Inverse Document Frequency (Tf/idf) are not very suitable, if applied directly.
3. Webpages are written in HTML with a predefined set of tags, whereas service descriptions are more abstract, requiring a knowledge about XML schemas, namespaces and document well-formedness.
4. Finally, there are no links in WSDL files that can connect one WSDL file with another. The WSDL is meant for describing the capabilities of a single Web service, hence such related service information is not available. Hence, the crawler cannot simply generate more WSDL pages out of several seeds WSDL files, as is the case with traditional IR.

For dealing with these distributed sources and the problems associated with applying traditional IR techniques, a modified IR based approach for a service-retrieval specific mechanism,as designed. The main requirement of such a method is, selectively finding only service descriptions among the billions of documents published on the Web and retrieving them whenever such services are available. To achieve this, a service-specific crawling mechanism called a *Specialized Service Crawler (SSC)* is proposed. We discuss the SSC mechanism and its associated processes and the way these are employed to build the service repository, in Section 4.3.1.

4.3.1 Building the Service Repository

Figure 4.1 shows the architecture of the service retrieval mechanism of the proposed $\mathcal{DWDS}$ framework. The Specialized Service Crawler forms a part of the Distributed Web Service Retrieval and Indexing Engine (DWSRIE). Its main components are - the *SSC*, the *Webpage URL Collector*, the *WSDL Existence Checker*, the *WSDL Parser & Downloader* and the *WSDL Duplicate Checker & Indexer*. All successfully retrieved WSDL files are finally indexed in the *WSDL Preprocessor* database. Each of these components are described next.

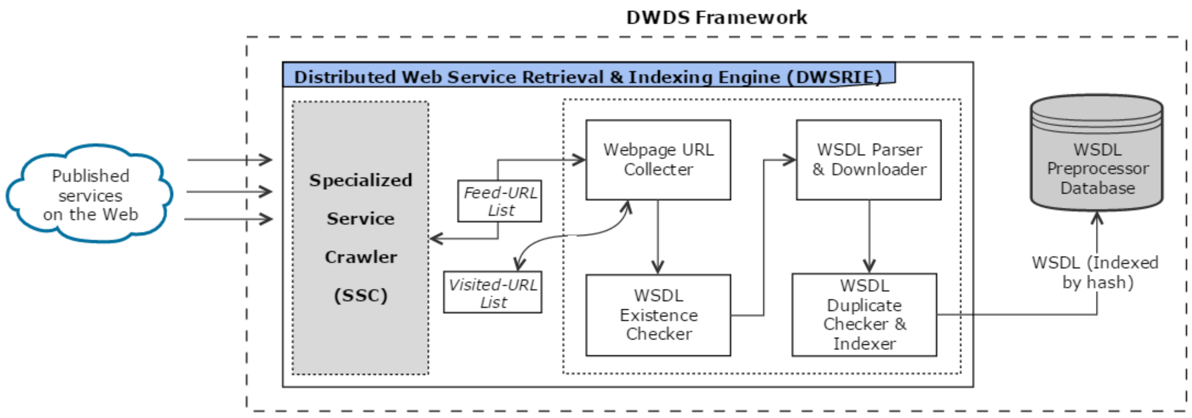

DWDS Framework
Distributed Web Service Retrieval & Indexing Engine (DWSRIE)
Published services on the Web
Specialized Service Crawler (SSC)
Feed-URL List
Visited-URL List
Webpage URL Collecter
WSDL Existence Checker
WSDL Parser & Downloader
WSDL Duplicate Checker & Indexer
WSDL Preprocessor Database
WSDL (Indexed by hash)

Figure 4.1: Distributed Web Service Retrieval and Indexing Process

(i) Specialized Service Crawler (SSC)

The SSC is based on the concept of a topical or focused crawler (Chakrabarti et al. 1999). Focused crawlers consider the fact that relevant pages tend to link to other relevant pages, either directly or through relatively small number of links. The $\mathcal{DWDS}$ SSC uses a Breadth-First-Search (BFS) algorithm for recursively collecting URLs, which are then examined for any potential service descriptions. During experimental evaluation, it was observed that BFS based focused crawl discovers the highest quality pages (links that can potentially yield valid service descriptions) during the early stages of the crawl, when compared to using depth-first-search (DFS). This is because; the probability that the neighbouring pages of a relevant link are also relevant is quite high. Hence, if such neighbouring links are examined early (as in BFS based crawl), more desired documents may be potentially found, as a faster rate. When DFS was used, the relevancy of the downloaded pages deteriorated as the depth from the original feed-URL increased. Based on these observations, a breadth-first search based focused crawl strategy was adopted.

The SSC consists of a set of distributed focused crawlers that start the crawling process using a set of initial starting points (*feed-URL list*). Initially, the feed-URL list consists of the URLs of some service portals currently in operation[1]. Each crawler thread is assigned a feed-URL from the initial feed-URL list and the focused crawl for WSDLs is performed. The newly discovered URLs are handled by the DWSRIE Webpage URL collector. Then, each crawled page p is categorised as either *relevant* or *irrelevant* based on certain criteria, and the neighbourhood of each relevant p is further explored up to depth d, looking for additional relevant pages. The new URLs discovered are collected by the next component.

[1]BioCatalogue, EMBL-EBI (ebi.ac.uk), ProgrammableWeb, ServicePlatform, Service-repository, WebServiceList, WebserviceX and XMethods

(ii) Webpage URL Collector

As discussed earlier, the initial feed-URL list consists of the links to the various service portals, and each crawler thread fetches the links from that domain's webpages. These newly fetched links are continuously added to the feed-URL list, which will be used as new feed-URLs for later crawl cycles. The function of the Webpage URL collector is to maintain the feed-URL list for use by the SSC. These links will be examined for potential WSDL files and also subjected to various checks during the subsequent phases, as performed by the next modules.

(iii) WSDL Existence Checker

To restrict the crawlers to webpages yielding service descriptions only, certain additional rules are incorporated in the SSC algorithm and these checks are performed by the WSDL Existence Checker module. To identify whether a retrieved web document is a web service description, three *WSDL Existence Checks* are incorporated. These include -

1. Verifying if the URL of retrieved webpage matches the WSDL regular expression (the webpage URL ends in either '`.wsdl`' or '`?WSDL`').
2. Checking the HTTP response header for the fetched webpage, for any occurrence of MIME types that may indicate Web service documents (the correct MIME type for WSDL documents is '`application/wsdl+xml`', however, sometimes the more generic MIME type, '`text/xml`' also is found).
3. Checking the source code of the webpage to determine if any web service description indicators are present (e.g. WSDL and XML namespaces, WSDL specific elements like portType, messages etc).

It was observed that out of these three tests, at least two have to be true to make a confident decision about the possible availability of a WSDL within the webpage. If only the `content-type` response header is considered, there is an increased possibility of false positives, specifically if the MIME type is '`text/xml`', as even a simple XML file can have the same MIME type. Hence, if at least two of the tests are true, then a WSDL may have been found and further processes are performed.

(iv) WSDL Parser & Downloader

Based on the results of the WSDL Existence Check, a WSDL has been found if at least two checks are true. It is now necessary to ensure that it is valid before adding it to the repository. To check the validity of the WSDL, its XML Document Object Model (DOM) structure has to be correct and valid. A *WSDL Parser* is used to parse the entire

WSDL file to check if it has a valid structure. If invalid, then the link is added to the *invalidWSDL-URL* list and finally removed from the feed-URL list, to avoid processing these unproductive links unnecessarily during future crawling rounds. If valid, then the WSDL file is downloaded from its location on the Web for further processing.

(v) WSDL Duplicate Checker & Indexer

Before the WSDL is added to the service collection, we need to ensure that only new and unique files have been downloaded. To represent each WSDL file uniquely, a simple hash generation function is run on the entire WSDL file to generate its unique hash. Then, the list of hashes of already indexed files is checked and only if the newly generated hash does not exist on the list, the new WSDL is added to the *WSDL Preprocessor* database.

4.3.2 SSC Crawling Modes

The SSC performs two types of crawling - *Active Crawling* (for repository building) and *Selective Crawling* (for repository management). These are referred to as *SSC Crawling Modes* and are described here.

1. *Active Crawling.* During this mode, the SSC set of crawlers peruse the links on the feed-URL list to perform the focused crawl with the objective of finding links matching the WSDL pattern. As new services are continuously published on the Web, the active crawling mode focuses on finding these new services for extending the $\mathcal{DWDS}$ repository further. In general, the active crawling process was carried out on a weekly basis. Algorithm 4.1 refers to the active crawling mode.

   Algorithm 4.1 illustrates the working of SSC and all its associated processes during the active crawling mode. Steps 1 through 4 are performed by the SSC and the Webpage URL Collector module. Based on the URLs found during these phases, the WSDL Evidence Checker module performs the task of identifying if a WSDL yielding link has been found (steps 6 to 8). If at least two check are positive, then the WSDL Parser Module applies a WSDL DOM tree check and if the file is found to be valid, then it is downloaded. Next, the WSDL Duplicate Checker component generates the hash of the newly found service document and checks if the hash exists. If yes, then this is a duplicate WSDL, which is discarded and the corresponding URL is added to the *duplicateWSDL-URL* list and the *visited-URL* list to optimize later crawling rounds. In case the new WSDL's hash does not exist, then it is added to the WSDL Preprocessor database with the status *'NewWSDL'*. Further processing like metadata generation and categorisation are carried out on all WSDL files in the WSDL preprocessor database, with the status *'NewWSDL'*.

Algorithm 4.1 Distributed Web Service Retrieval and Indexing Process

```
Input: Crawler Queue $Q$ with initial list of feed-URLs
Output: Valid WSDL files
 1: while crawler queue $Q$ with $feed\text{-}urls \neq \varnothing$ do
 2:     Perform $webpage.getURL()$ for each $feed\text{-}url$
 3:     Extract all hyperlinks from $webpage$ and add to $feed\text{-}url$ list      ▷ New feed URLs.
 4:     for each hyperlink on $feed\text{-}url$ list do
 5:         if hyperlink is not on $visited\text{-}URLs$ list then
 6:             apply WSDL evidence check rules (1) to (3)
 7:             if at least two checks are positive then      ▷ hyperlink contains WSDL.
 8:                 apply WSDL structure validity check
 9:                 if WSDL is valid then
10:                     download file and generate hash of service document
11:                     if hash exists then
12:                         WSDL already exists in repository
13:                         set URL status to ‘duplicateWSDL-URL’
14:                         add to $visited\text{-}URL$ list      ▷ to optimize later crawling rounds.
15:                     else
16:                         Set new service document status to ‘NewWSDL’
17:                     if WSDL is invalid then
18:                         add $webpage.url$ to $invalidWSDL\text{-}URL$ list
19:                         add to $visited\text{-}URL$ list      ▷ to optimize later crawling rounds.
20: Save new, valid WSDL file in WSDL Preprocessor database
```

2. *Selective Crawling.* Once the services are added to the $\mathcal{DWDS}$ repository, there needs to be process of checking the continued availability of an indexed service at its remote location on the Web. This is required to ensure that the repository is kept updated and unavailable services are automatically removed. To deal with this problem, the selective crawling mode is used. The selective crawling mode uses a special feed-URL list called the *validWSDL-URL* list, which consists of the service URIs of all successfully indexed services (shown in Figure 4.2). Using this list, the SSC performs a selective crawl of only those webpage links from which services were retrieved, to find if the service and its description is still available for download at the remote location.

If a service is not available, then the corresponding file has to be removed from the $\mathcal{DWDS}$ repository. However, this is not done immediately to compensate for any temporary unavailability problems. During the first selective crawling round, if the service is found to be unavailable, then it is flagged, but is left untouched in the service repository. If a second selective crawling round also finds that the service is still unavailable, only then the indexed service is deleted. The selective crawling process is performed periodically, on a fortnightly basis.

To enable efficient management of the service repository and the crawlers, certain autonomic processes were incorporated in the proposed $\mathcal{DWDS}$ framework. As the crawler based framework is inherently dynamic, due to periodic active and selective crawling processes, it is important to optimize the system performance and reduce human intervention. The mechanisms designed to support autonomous management of $\mathcal{DWDS}$ are described in Section 4.4.

4.4 Autonomic Repository Management

One of the important requirements of the $\mathcal{DWDS}$ framework is to automate critical maintenance tasks to reduce human effort and thus, promote self-management. Some of the issues observed during the process of building the service repository were -

1. Service descriptions that are already indexed within the repository may no longer be available at their remote locations any more. Hence, it becomes important to keep the index updated such that the status of already indexed documents can be automatically checked. This means periodic re-crawls have to be scheduled to maintain the index (handled by incorporating the *Selective Crawling Mode*).

2. Some URLs that match the WSDL containing regular expression, may not have a valid service URI, which cannot be added to the repository as the required service endpoint information would be missing. These also have to be kept track of, to avoid reprocessing (handled by WSDL Evidence Checker Module).

3. Some downloaded WSDLs may not have a valid DOM structure. These have to be identified and removed from the repository, and the URLs that resulted in these files have to be noted (handled by WSDL Parser Module).

4. A service description may be available from multiple sources, e.g. in more than one service portal and also from the service provider's website. In this case, only the first occurrence should be added to the repository and the others should be discarded as duplicates (handled by WSDL Duplicate Checker Module).

5. Very few URLs would match the WSDL regular expression (WSDL Evidence Check (1), discussed in Section 4.3.1). Most URLs processed by the SSC are actually those of normal webpages. These have to be kept track after the first time they are processed, and eliminated from the SSC feed-URL list to avoid unnecessary reprocessing during subsequent crawler rounds. Similarly, URLs that match the WSDL regular expression but do not contain a valid service URL and those yielding invalid/duplicate WSDLs also need to be kept track of, for performance optimization.

To tackle these issues, certain autonomic features were incorporated into the framework. As several thousand data entities are being handled (URLs, WSDL files, invalid WSDL files, invalid URLs etc), unnecessarily reprocessing these would be a waste of computing resources. Over time, this volume of data is subject to many (relatively) small changes each time new service descriptions are crawled and added. Also, as this data is processed in other ways – for example, generating metadata for each WSDL, categorising the services as per their domain and finally making them available for discovery, it is important optimize data handling. This challenge was addressed by designing and incorporating an effective change propagation technique based on the concept of event based state machines, which is described in more detail in Section 4.4.1.

4.4.1 The $\mathcal{DWDS}$ Change Propagation Strategy

As discussed in the previous section, the proposed $\mathcal{DWDS}$ framework is highly dynamic due its crawler-based mechanism. In this context, a major challenge to be addressed is automatic change propagation. *Change propagation* focuses on the problem of limiting each component's processing to only the newest changes, so that results are propagated down the line to the next component without having the redo the computations for the entire data (Lopes et al. 2007). The main issues to be handled would be -

1. how to propagate changes between the various components of the framework, so that consistency is maintained always.
2. how to process the continuous changes automatically without any need for manual intervention.

Keeping these concepts in mind, the $\mathcal{DWDS}$ change propagation strategy was designed based on the concept of a state machine. In $\mathcal{DWDS}$, each data entity has an associated "status", which is used to keep track of the various states that particular entity is in. Table 4.1 shows the list of states a particular data entity can be in, in the order corresponding to the various processing phases.

Table 4.1: The Proposed Incremental Strategy (States and the associated data entities)

No.	**State** ***(Associated Entity)***	**Meaning**	**Associated Action**
1	URLNoMatch *(URL)*	URL of crawled Webpage does not match the WSDL regular expression.	Add to *visited-URL* list and later, remove from *feed-URL* List.
2	noServiceURL *(URL)*	matched against the regular expression, but no valid service URL	Add to *visited-URL* list and later, remove from *feed-URL* List.
3	invalidWSDL-URL *(URL)*	Downloaded WSDL did not pass DOM tree verification test	Add to *visited-URL* list and later, remove from *feed-URL* list
4	duplicateWSDL-URL *(URL)*	Hash of service data already exists in WSDLPreprocessor database	Add to *visited-URL* list and later, remove from *feed-URL* list
5	validWSDL-URL *(URL)*	URL yielded a valid WSDL.	Add to *validWSDL-URL* list, which will be used by crawler to check status of downloaded WSDLs during periodic crawler runs.
6	NewWSDL *(WSDL)*	Valid WSDL file that is stored in WSDL Repository indexed by its hash	Initiate extraction of service elements.
7	WSDLProcessed *(WSDL)*	Elements extracted from WSDL & is ready for use.	Available for Metadata Generation Process
8	MetadataGenerated *(WSDL)*	Status of the WSDL for which metadata was successfully generated	The generated metadata and service are added to Service Repository.
9	ToBeCategorised *(WSDL)*	Status of a metadata-generated WSDL file when added to service repository.	Available for categorisation.
10	Active *(WSDL)*	Status of WSDL files that were successfully categorised.	Available for user querying.

As shown in the Table 4.1, a data entity in the system can be in the various states ranging from *URLNoMatch* to *Active* depending upon the various processes and transformations it has to undergo. Depending on which state a data entity is in, it is also possible to confidently say which data entity we are talking about. For example, in states *URLNoMatch* to *validWSDL-URL*, the data entity in question is an URL, which is important since keeping track of the state of an URL is useful in optimizing the functioning of the service crawler during subsequent crawls. In states *NewWSDL* through *MetadataGenerated*, the WSDL file is the data entity under consideration. Here, as the WSDL file's status changes from one to another, the change is propagated to the next data entity, the enriched WSDL files, in the phases *ToBeCategorised* and *Active*. Hence, only new changes are being processed, instead of all the data in the repository. So, automatic change propagation is achieved, that helps in efficient data handling in $\mathcal{DWDS}$.

Figure 4.2 depicts the various processes associated with the SSC and shows the workflow of the proposed change propagation based data handling strategy. The SSC's set of crawlers recursively fetch feed-URLs for a *feed-URL* list, extract links from each feed-URL and apply the processes described in Algorithm 4.1 to determine if a valid WSDL file exists. During this process, thousands of URLs need to be processed and most may not match the WSDL regular expression at all. For pruning the *feed-URL* list, these are assigned the status *URLNoMatch* and are added to the *visited-URL* list. In case a candidate URL matches the WSDL regular expression, then the next task is to check if it has a valid service URL, if yes, the WSDL is parsed for a DOM check and if valid, its hash is generated. The hash helps in eliminating duplicate WSDL entries in the repository.

At this point, those URLs that yield a WSDL without a valid service URL and URLs yielding invalid WSDLs have to be noted and dealt with, appropriately (added to *visited-URL* list with the status *NoServiceURL* and *InvalidWSDL-URL* respectively). After generating the hash of the service document, a check is performed to see if the hash exists, if yes, then the WSDL file is assigned a status *duplicateWSDL* and the corresponding URL is added to the *visited-URL* list. If hash does not exist, then the WSDL file is indexed in the WSDL Preprocessor database by its hash.Ultimately, the URLs on the *visited-URL* list are removed from the feed-URL list to avoid reprocessing of the same by another crawler thread.

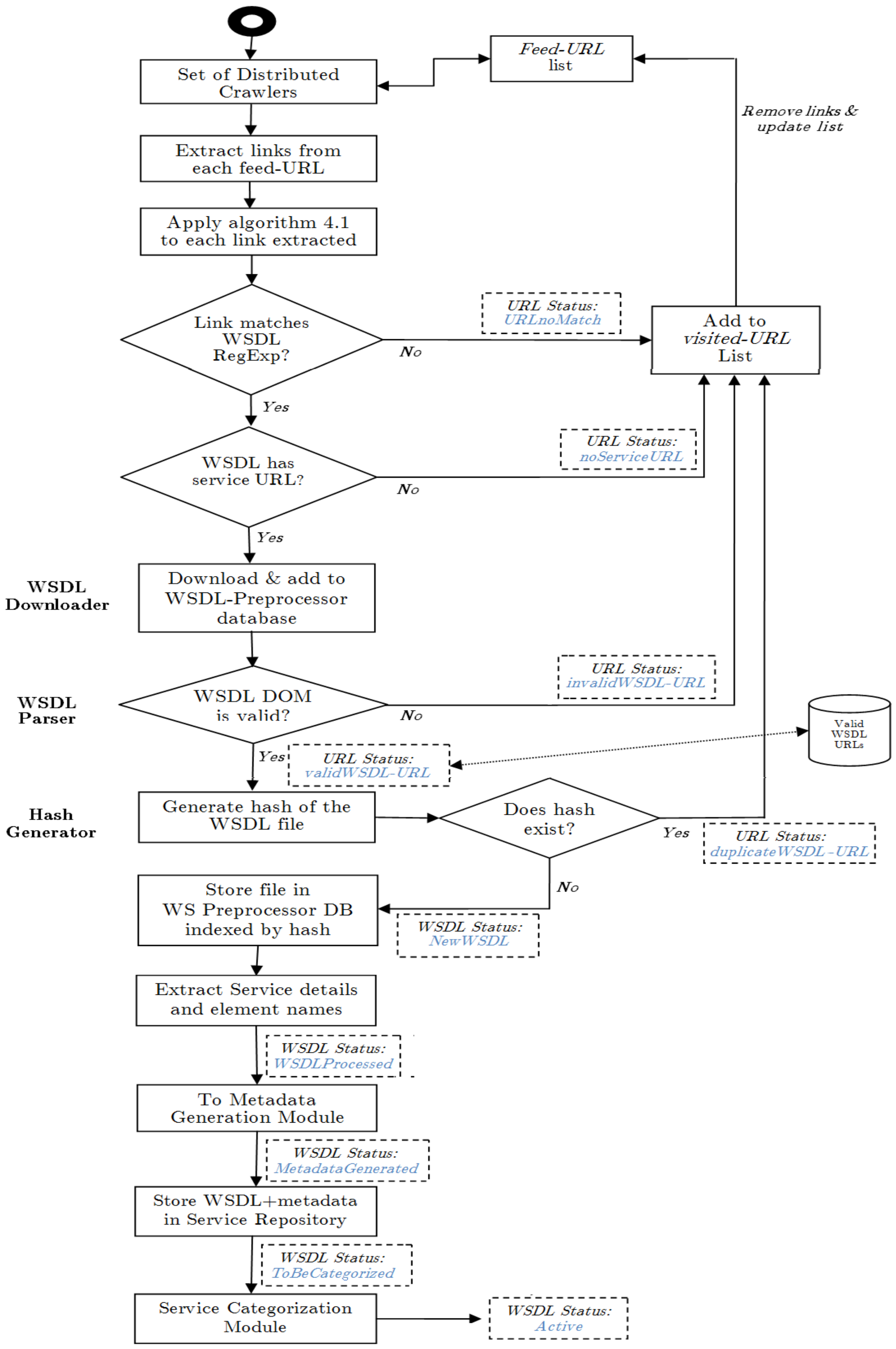

Set of Distributed Crawlers
Feed-URL list
Remove links & update list
Extract links from each feed-URL
Apply algorithm 4.1 to each link extracted
Link matches WSDL RegExp?
No
URL Status: URLnoMatch
Add to visited-URL List
Yes
WSDL has service URL?
No
URL Status: noServiceURL
Yes
WSDL Downloader
Download & add to WSDL-Preprocessor database
WSDL Parser
WSDL DOM is valid?
No
URL Status: invalidWSDL-URL
Valid WSDL URLs
Yes
URL Status: validWSDL-URL
Hash Generator
Generate hash of the WSDL file
Does hash exist?
Yes
URL Status: duplicateWSDL-URL
No
Store file in WS Preprocessor DB indexed by hash
WSDL Status: NewWSDL
Extract Service details and element names
WSDL Status: WSDLProcessed
To Metadata Generation Module
WSDL Status: MetadataGenerated
Store WSDL+metadata in Service Repository
WSDL Status: ToBeCategorized
Service Categorization Module
WSDL Status: Active

Figure 4.2: Workflow and associated processes of the SSC

4.4.2 Event Driven Change Propagation

Even with an automatic change propagation approach, it is inefficient to be continuously scanning the repository for the various state changes, for each small change and then performing the predefined actions. To overcome this issue, the concepts of event driven programming were used, for deciding when each of the modules should do their job. The proposed system's state machine is an event driven model, which keeps track of various events. When the number of events exceeds a fixed threshold value, the corresponding actions are automatically initiated and the change propagation happens automatically. Algorithm 4.2 illustrates the Event-driven Automatic Change Propagation Strategy.

Algorithm 4.2 Event-driven Automatic Change Propagation Algorithm

```
1: Initialize $event\text{-}count := 0$
2: for each event type do
3:     while $event\text{-}count \leq threshold$ do
4:         log each data entity's state change    ▸ keep track of all URL, WSDL and enriched WSDL events
5:         Increment count of that type of event
6:         if $event\text{-}count > threshold$ then
7:             Perform action associated with event type
8:             reset $event\text{-}count$
```

To illustrate this strategy, consider that a URL being examined turns out not to match the WSDL regular expression. As per Algorithm 4.2, the sequence of actions that are to be followed is begins when the URL is assigned a *URLNoMatch* status. As per the event-driven change propagation strategy, this is simply logged as an event on the event counter and the URL is added to the *URLNoMatch* list as per policy. After this, the system continues processing the next URL in the *feed-URL* list, continuously keeping track of subsequent events in an appropriate list. When a prefixed threshold is reached for URL related events (i.e., the number of URLs with *URLNoMatch* status is greater than fixed threshold value), then, the crawler's feed-URL list is automatically updated by deleting the URLs on the *URLNoMatch* list from it. Same procedure is followed for various other events like *noServiceURL*, *InvalidWSDL* and *duplicateWSDL-URL* etc, thus optimising the data handling process. In the next section, we present the experimental results as to the effectiveness of the proposed framework with the specialized service crawler mechanisms and its associated processes.

4.5 Experimental Results and Discussion

In this section, we present the experimental results and a discussion on the effectiveness of the proposed distributed discovery framework with reference to the contributions listed at the beginning of the chapter. We present the performance details of the SSC set of crawlers in Section 4.5.1. The statistics as to the number of services listed in the repository, the sources from which they were retrieved is discussed in Section 4.5.2. Some details about the quality and diversity of the service collection currently indexed in the framework's repository are presented in Section 4.5.3. We demonstrate that the framework is scalable using the statistics of the services indexed within the repository during various time periods and the changes over time in Section 4.5.4.

4.5.1 SSC Performance

To evaluate the performance of the service crawlers, the time required to collect URLs (URL harvest rate) was first measured. Also theoretical analysis as to the time and space complexity of the SSC is presented. Table 4.2 depicts the number of URLs collected by the multi-threaded crawlers over time. It is seen that collection time grows linearly with the number of URLs collected. The number of URLs collected by the crawler over time, number of URLs that matched the WSDL URL pattern and the number of valid WSDL files obtained are tabulated. It can be seen that the number of irrelevant URLs that have to be processed in order to obtain a few URLs matching the WSDL URL pattern that may or may not lead to valid WSDL files is quite high. This stresses the importance of optimizing the process of data handling to avoid unproductive, additional processing of data entities like irrelevant URLs and WSDL files after each crawler run.

Table 4.2: Crawl data specifics of the Multi-threaded crawlers of SSC

Total URLs fetched	**URL harvest rate (in minutes)**	**Links matching WSDL pattern**	**Links yielding valid WSDLs**
1000	6	87	39
5000	21	271	134
10000	39	507	298
15000	58	722	412
20000	82	1037	609

4.5.1.1 Theoretical Analysis of SSC Performance

The process of finding, retrieving services and indexing them in the repository after the requisite checks is the offline phase of the proposed framework; hence the processes are running in the background.

1. *Finding new links from feed-urls:*

   The algorithm takes as input an initial set of URLs, i.e the *feed-urls*, and the *depth* for the crawling. The algorithm begins with the *feed-urls* and explores all the neighbouring URLs (i.e. all the URLs that can be directly reached from the initial URL). Then, for each of the neighbouring URLs, it explores its unexplored neighbours, and so on until the given depth is reached. Hence, given a branching factor b and a depth d, the algorithmic time complexity is -

   $$|feed\text{-}urls| + |feed\text{-}urls|.b + |feed\text{-}urls|.b^2 + \ldots + |feed\text{-}urls|.b^d = \mathcal{O}(b^d)$$

   We found that, the average level at which the BFS based SSC crawler thread could not find any more URLs matching the WSDL pattern was about 4. Hence, we have set a maximum limit of 5 levels for the depth of the crawl, while the number of URLs crawled at each level was set to a maximum of 10000. Since all of the nodes of a level must be saved until their child nodes in the next level have been generated, the space complexity is proportional to the number of nodes at the deepest level. Thus, given a branching factor b and graph depth d, the asymptotic space complexity is also $\mathcal{O}(b^d)$.

2. *Feed-URL list Management:* The feed-URL list (Crawl frontier) is implemented as a FIFO queue using a dynamic array. Hence, new URLs are added to the end of the queue and crawler threads remove the URL at the beginning of the queue. Each queue/dequeue operation takes an average of $\mathcal{O}(1)$ time.

3. *Visited-URL list management:* As soon as a crawler thread dequeues an URL from the feed-URL list, this is a visited URL and needs to be kept track of, so avoid duplicates in the feed-URL list. The Visited-URL list is implemented as a separate hash-table (with URL as key) to store each of the frontier URLs for fast lookup. The hash-table is kept synchronized with the Feed-URL list. Hence, for each search and insert, the hash table has an average time complexity of $\mathcal{O}(1)$.

4. *Managing the lists containing URLNoMatch, InvalidWSDLURL, NoServiceURL and DuplicateWSDL-URL entities:* Each of the URLs with this status are added to the Visited-URL list as a key-value pair, to avoid crawling these links again. Hence, each insert operation has an average time complexity of $\mathcal{O}(1)$.

5. *ValidWSDL-URL List Management:* The selective crawling mode uses a special feed-URL list called the *validWSDL-URL* list, which consists of the service URIs of all successfully indexed services. Using this list, the SSC crawler threads perform a selective crawl of only those webpage links from which services were retrieved, to find if the service and its description is still available for download at the remote location. This list is also implemented as a FIFO queue using a dynamic array, hence each queue/dequeue operation takes an average of $\mathcal{O}(1)$ time.

Assuming that, the average number of links discovered per page visited is about 6 (as proven in the study published by Kumar et al. (2000) and also verified by Pant et al. (2004)), the number of URLs added to the feed-URL list per page visited, by each crawler thread is relatively small. Hence, overall time complexity of SSC can be said to be effectively proportional to the complexity of finding new links from existing feed-URLs and the number of links handled by $\mathcal{DWDS}$ (denoted as $\mathcal{L}$). Therefore, worst-case complexity = $\mathcal{O}(b^d).\mathcal{O}(\mathcal{L})$.

4.5.2 Volume Statistics of the $\mathcal{DWDS}$ Service Collection

In this section, statistics about the services available in the service repository currently, their sources, and their quality data is presented. We also discuss the performance of the SSC with reference to URL harvest rate. Table 4.3 shows the number of services available and also number of valid WSDLs found from different sources after the service crawling (all retrieved WSDLs) and WSDL indexing (all valid WSDLs) as of September 2015. To obtain the total valid WSDLs, each WSDL obtained from a source is subjected to a WSDL structure check to confirm that the file is a WSDL and not just an XML file. The Eclipse IDE provides the WSDL4J WSDL parser and validator, which is used for validation of a WSDL document tree. The total number of files obtained from each service portal with a valid WSDL DOM was taken as the count of the total valid WSDLs for each source.

As seen from Table 4.3, the SSC was successfully retrieved published services from various service portals as well as other remote locations of the Web. Of the retrieved services, a significant number were found to be without a valid service URI/invalid and hence were discarded. The quality of the service descriptions obtained from various sources is also tabulated in Table 4.3, and is computed as the ratio of valid to total service descriptions obtained. Of the multiple sources, the portal `ebi.ac.uk` yielded the best quality services, though the collection is currently quite small. Among larger portals, the services obtained from ProgrammableWeb resulted in the highest percentage of valid services, at 81.49%. This indicates that, service providers who advertised on this site

Table 4.3: Summary of retrieved WSDLs and their sources (as of Sep 2015)

Source	Total obtained	Total valid	Ratio of valid to total (%)
BioCatalogue	2315	1139	49.2%
Ebi.ac.uk	70	68	97.14%
ProgrammableWeb	1978	1612	81.49%
ServicePlatform	2183	972	44.52%
Service-repository	127	98	77.16%
Webservicelist	1134	286	25.22%
WebserviceX	71	21	29.58%
XMethods	411	338	82.23%
Open Web	11971	7697	64.29%
Total number	**20260**	**12231**	**61.2%**

were generally more active in maintaining their service data.

Figure 4.3 shows the relative service statistics for the various service portal sources from which WSDLs were retrieved. Among the services obtained from sources other than service portals (service providers' Web servers, from webpages and HTML forms etc), about 64.29% of the WSDLs obtained were valid. This is still quite good, given the large number of WSDLs obtained, as a majority of these come from service providers' websites and thus are relatively better maintained than those in service portals. A total of 12,231 service descriptions are currently indexed within the proposed framework's service repository and it can be said that the SSC based framework was effective in finding distributed services from varied sources over the Web.

4.5.3 Quality Statistics of the $\mathcal{DWDS}$ Service Collection

In this section, we present some details on the quality and diversity of the services indexed in the repository currently. Figure 4.4 presents the WSDL file size statistics of the service collection. An application designer has to first understand the interfaces offered by the service before using it. The file size is indicative of the complexity of a service. As the number of operations supported increases, the complexity increases, and also the size. Through this analysis, we learned that about 41% of the files were less than 10KB, about 65% were less than 20KB and about 85% of the total collection was less that 50KB in size. Only about 13% of the services were more than 50KB in size, out of which only 3% services were over 100KB and contained several operations. Through this analysis, we were able to find that the majority of the WSDL files were less than 50 KB, and the

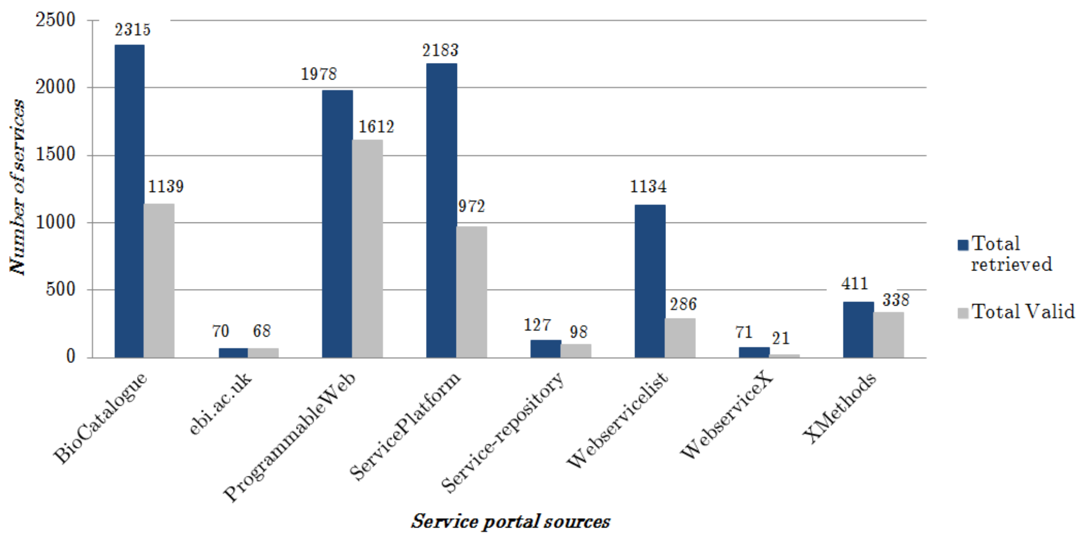


Figure 4.3: Service portals - Service quality statistics (Sep 2015)

available service descriptions were low in complexity.

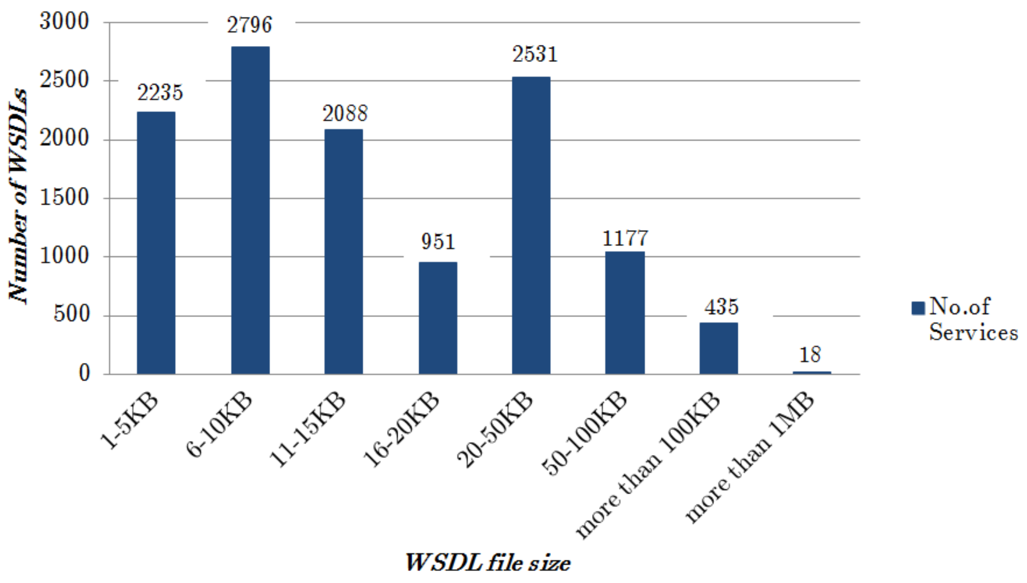


Figure 4.4: WSDL Complexity statistics (Sep 2015)

Next, we present some details as to the level of natural language documentation provided for these real world services. As reported by earlier studies (Fan et al. 2005; Li, Zhang, et al. 2007), we too found that several indexed services either did not have any documentation at all or had very small quantity of documentation that may not effectively capture the capabilities of the service. Figure 4.5 presents the statistics on the quality of the natural language documentation available for each WSDL within its <*documentation*> tag. Over 23% of indexed services did not have any natural language description at all. About 29% had less than 10 words of natural language documentation. This stresses the need for additional techniques to capture the functional semantics of the services to enable domain specific categorisation.

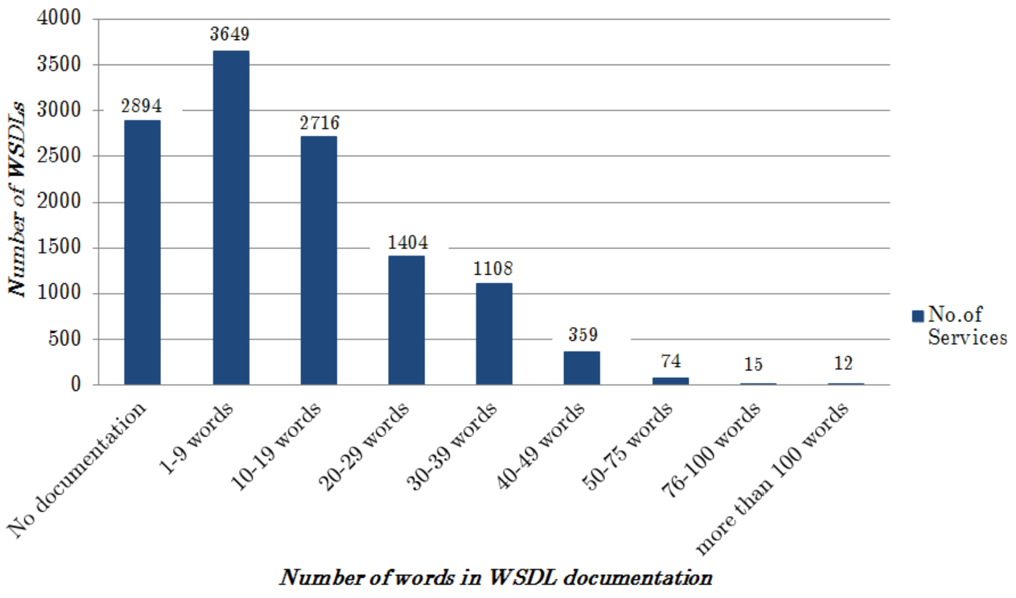


Figure 4.5: Quality of Natural Language Documentation (Sep 2015)

4.5.4 Temporal Statistics of the $\mathcal{DWDS}$ Service Collection

During the past three years of our work, periodic statistics were collected with reference to the changes in the $\mathcal{DWDS}$ service collection over time. These temporal statistics are discussed in this section. Table 4.4 presents the details of the number of services retrieved and indexed in the service repository as of September 2013, September 2014 and September 2015. It can be observed that, in some of the portals (ServicePlatform and WebserviceX), the number of services available has remained the same, indicating that there has been no maintenance/advertisement activity at all over the past three years, while in others (ProgrammableWeb and ebi.ac.uk), a number of REST[2] based Web APIs are also available in addition to WSDL based services. The ebi.ac.uk collection has reduced significantly since 2013 as only selected services are being offered currently, many of which are REST based only. This may be because REST is gaining popularity and service providers are also offering them as an option/alternative currently.

For understanding the changes in the service distribution and the rate of retrieval over time, we generated statistics on the growth rate of the indexed services on a yearly basis, starting from September 2013 to September 2015. As seen from Table 4.5, there

[2]Representational State Transfer (Fielding 2000)

Table 4.4: Temporal Statistics on indexed WSDLs and their sources

Source	**Sep 2015**		**Sep 2014**		**Sep 2013**	
	Total	*Valid*	*Total*	*Valid*	*Total*	*Valid*
BioCatalogue	2315	1139	1003	821	1892	981
Ebi.ac.uk	70	68	267	212	423	306
ProgrammableWeb	1978	1612	2231	1872	2062	1765
ServicePlatform	2183	972	2183	1289	2183	1437
Service-repository	127	98	104	69	83	74
Webservicelist	1134	286	1217	297	1051	173
WebserviceX	71	21	71	37	71	37
XMethods	411	338	341	301	358	291
Open Web	11971	7697	12300	6707	15272	8873
Total	**20260**	**12231**	**19717**	**11605**	**23395**	**13937**

was a significant change in the number of indexed services in September 2014, this was because a bigger number of services retrieved from other sources on the Web were found to be unreachable at certain points of time. Due to this, these files were removed from the repository during selective crawling rounds.

Table 4.5: Service availability - Growth Rate

Sources	**Sep 2013**	**Sep 2014**	**Sep 2015**	**% Change (2013-14)**	**% Change (2014-15)**	**Average Change**
Service Portals	5064	4898	4534	- 1.66 %	- 3.8%	- 3.68%
Other Sources on the Web	8373	6707	7697	- 11.05%	+ 6.87%	- 2.79%
Total indexed	13437	11605	12231	- 7.3%	+ 2.62%	- 3.12%

As of September 2015, the percentage change in the total number of services indexed in the $\mathcal{DWDS}$ repository over a period of last three years is about -3.12%. That is, the size of the current $\mathcal{DWDS}$ service repository is about 3% lesser than that of September 2013, but about 2.6% more when compared to the September 2014 service collection. Based on these temporal statistics, it can be concluded that the proposed distributed web service discovery and retrieval mechanisms were effective in enabling and supporting a scalable service repository for $\mathcal{DWDS}$.

4.6 Summary

In this chapter, a novel distributed Web service discovery framework for finding and retrieving services available in varied sources over the Web, in a scalable manner, was discussed. A Specialized Service Crawler mechanism was employed, for building the sizeable service repository, as evidenced by the collection statistics over the past three years. An event-based change propagation strategy was also incorporated to deal with the constantly changing data entities and to optimize the data handling processes. Experimental evaluation showed that the proposed techniques were effective. Analysis regarding the crawler performance; the number, quality and diversity of the service collection currently indexed within the service repository indicate that the framework is well-suited as a distributed discovery system. Currently, the developed system has a large service collection, which necessitates effective categorisation to deal with the functional diversity of the services, which will be addressed in the next chapter.

Publications

(based on the work presented in this chapter)

1. Sowmya Kamath S., and Ananthanarayana V.S., *"Change propagation based incremental data handling in a Web service discovery framework"* in Signal Processing and Information Technology (ISSPIT), 2014 IEEE International Symposium on, vol. 14, no., pp.000474-000479, 15-17 Dec. 2014.

2. Sowmya Kamath S., and Ananthanarayana V.S, *"A Service Crawler based Framework for Similarity based Web Service Discovery"*, at 11th IEEE India Conference on Emerging Trends and Innovation in Technology (IEEE INDICON 2014), 2014 Annual IEEE, pp. 1-6. IEEE, 2014.

PART III

Metadata Generation and Service Categorisation

Chapter 5

Generating Metadata for Web Services

5.1 Introduction

In the previous chapter, a novel distributed discovery framework for finding and retrieving Web services available in varied sources over the Web in a scalable manner, called $\mathcal{DWDS}$, was proposed. Currently, the developed system has a large service collection of 12,231 services, most of which do not have any associated information that can identify its domain or category. Dealing with this large collection would be a tedious and near-impossible task without intelligent mechanisms to organize them and to capture their domain information. Therefore, effective similarity analysis and categorization for grouping similar services is of utmost importance for supporting domain-specific Web service discovery in a user-friendly manner. In this chapter, we discuss the techniques proposed to address the problem of automatically generating metadata for services. Our main contributions, as reported in this chapter are listed here:

- Developing effective techniques for capturing the functional semantics of indexed Web services.
- Incorporating mechanisms for automatic metadata generation for each service, using the extracted functional semantics.
- Demonstrating that the proposed metadata generation mechanism was effective in capturing similarity of services for determining similar service groups.
- Verifying that the proposed mechanisms improve the Web service discovery process.

The rest of the chapter is organized as follows. Section 5.2 describes the defined problem addressed here and discusses some background information in this regard. Section 5.4 discusses the solution methodology developed for capturing the functional semantics of each service indexed in the $\mathcal{DWDS}$ repository. In Section 5.5, certain experiments that were performed to test the suitability of the generated metadata for service categorization are presented. Section 5.6 presents the experimental results and analysis with reference to the techniques discussed and Section 5.7 summarizes the work presented in this chapter.

5.2 Problem Statement

The problem that is addressed in this chapter is defined here:

> *Given the large number of indexed services in the framework's repository, to develop semantics based intelligent processing mechanisms for automatically generating metadata for each service, to enable efficient organization and management of the service collection.*

The aim is to incorporate automated processing mechanisms that can identify service functionalities from their service descriptions and use this to generate additional metadata. This metadata is also fundamental to the service categorization process, as the domain information of services is also captured. In Section 5.3, we discuss some background details on the proposed methodology and then discuss the algorithms developed for enhancing the service collections with semantics, in Section 5.4.

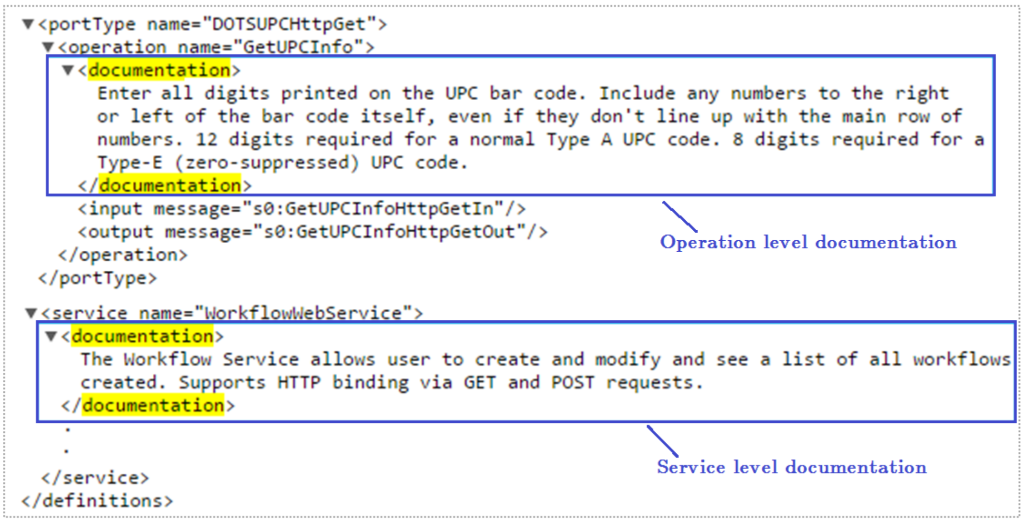
5.3 Background

In view of the defined objective, the first task is to determine potential avenues for analysing the information contained in the data at hand, i.e., the WSDL documents. Two possible options were considered - (1) The natural language documentation of a service, and, (2) The functional semantics of a service.

5.3.1 Natural Language Documentation of a Service

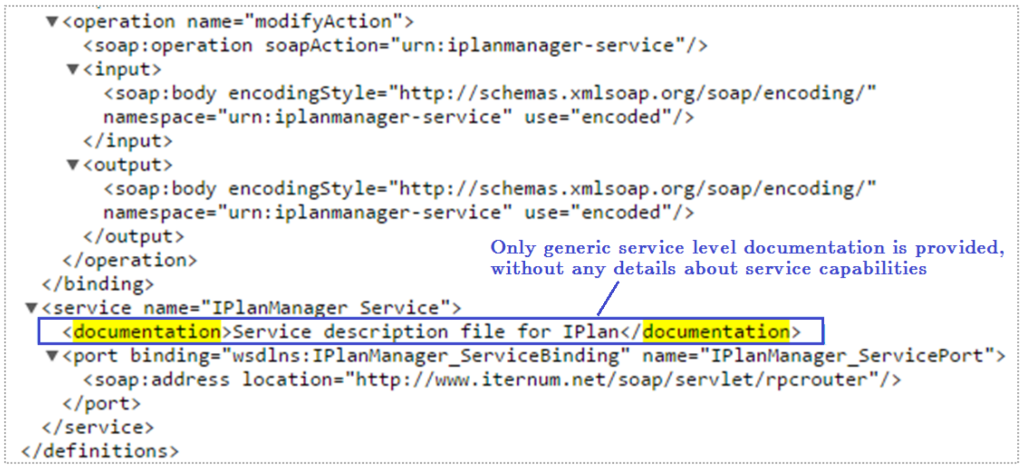
For the purpose of promoting easy discovery, WSDL provides a <documentation> tag, which is supposed to contain a detailed natural language explanation of the service functionalities. However, it has been reported that the quality of service documentation is quite often not rich enough for supporting efficient service discovery and selection (Li, Zhang, et al. 2007; Al-Masri et al. 2008). Fan et al. (2005) reported that this explanatory text is usually about 10-15 words in length, quite generic in several cases and often, even non-existent. Through our own analysis on the framework's service collection regarding this issue, we discovered that over 23% of indexed services did not have any natural language documentation at all, while more than 29% of the indexed services had less than 10 words of natural language documentation. In several cases, the provided text was as generic as *"Service description file for <service-name>"* or as useless as *"World's best service for free"*.

Ideally, a well-documented service description provides both service-level and operation-level documentation that helps in easily identifying the functionalities of a service. In Figure 5.1a, a snippet of such a well documented service description is shown, while Figure 5.1b illustrates a more common example as seen is many real world service descriptions. Hence, intelligent processing using semantic concepts and natural language processing methods are required for capturing the functional information that the service designers themselves failed to provide.



```
▼<portType name="DOTSUPCHttpGet">
  ▼<operation name="GetUPCInfo">
    ▼<documentation>
        Enter all digits printed on the UPC bar code. Include any numbers to the right
        or left of the bar code itself, even if they don't line up with the main row of
        numbers. 12 digits required for a normal Type A UPC code. 8 digits required for a
        Type-E (zero-suppressed) UPC code.
     </documentation>
     <input message="s0:GetUPCInfoHttpGetIn"/>
     <output message="s0:GetUPCInfoHttpGetOut"/>
   </operation>
 </portType>
▼<service name="WorkflowWebService">
  ▼<documentation>
      The Workflow Service allows user to create and modify and see a list of all workflows
      created. Supports HTTP binding via GET and POST requests.
   </documentation>
   .
   .
 </service>
</definitions>
```

(a) Well-documented service description



```
  ▼<operation name="modifyAction">
     <soap:operation soapAction="urn:iplanmanager-service"/>
    ▼<input>
       <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       namespace="urn:iplanmanager-service" use="encoded"/>
     </input>
    ▼<output>
       <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       namespace="urn:iplanmanager-service" use="encoded"/>
     </output>
   </operation>
 </binding>
▼<service name="IPlanManager Service">
   <documentation>Service description file for IPlan</documentation>
  ▼<port binding="wsdlns:IPlanManager_ServiceBinding" name="IPlanManager_ServicePort">
     <soap:address location="http://www.iternum.net/soap/servlet/rpcrouter"/>
   </port>
 </service>
</definitions>
```

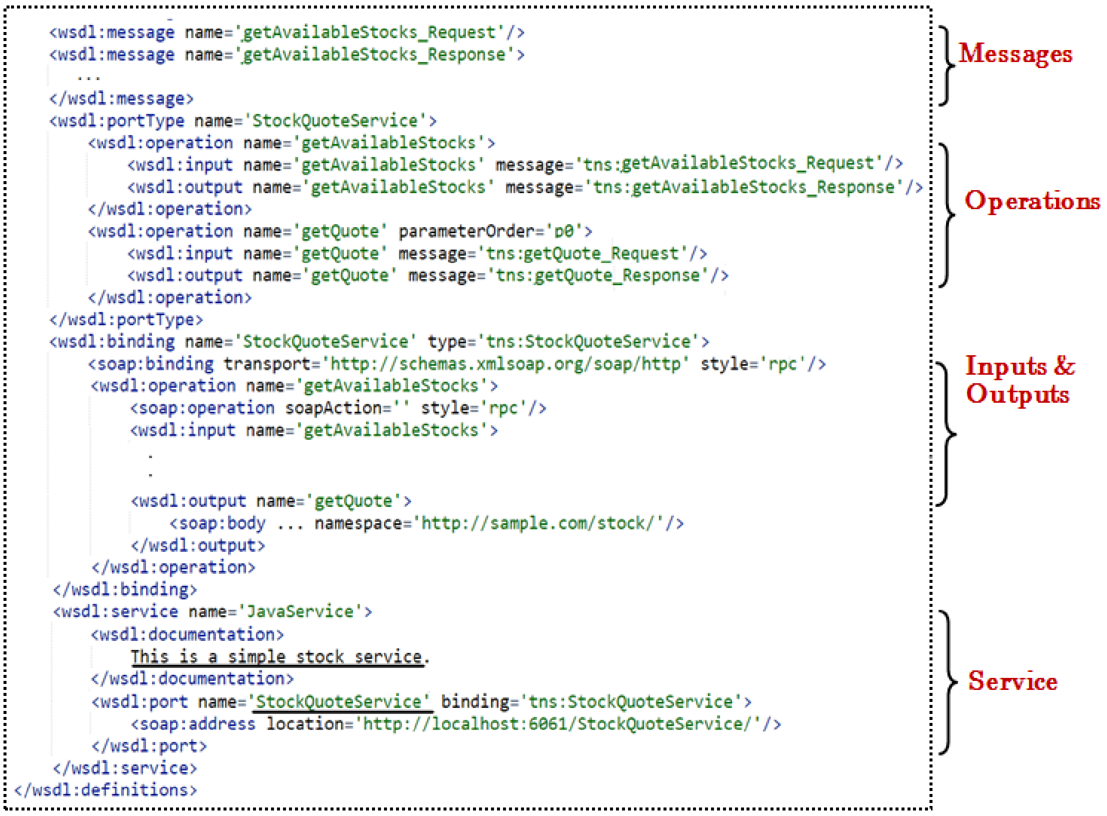
Only generic service level documentation is provided, without any details about service capabilities

(b) Poorly documented service description

Figure 5.1: Snapshot of well-documented and poorly documented WSDLs.

5.3.2 Functional Semantics of a Service

In view of the above limitation, an alternative is required to obtain the relevant information about a service's capabilities. The functional semantics of a service, that is, what a service can offer to its clients when it is invoked, can be helpful in this regard. The functionality of a service can be described in terms of what it does, and how it actually works. These aspects of a service's functional capability are captured to a certain extent by its service description (either WSDL or semantic descriptions like OWL-S and WSMO) and is referred to as the IOPE (Inputs, Outputs, Preconditions and Effects) of the service. However, due to limited availability of semantic service descriptions, the WSDL of the service can be a potential source of functional information.



```
    <wsdl:message name='getAvailableStocks_Request'/>
    <wsdl:message name='getAvailableStocks_Response'>
       ...
    </wsdl:message>
    <wsdl:portType name='StockQuoteService'>
        <wsdl:operation name='getAvailableStocks'>
            <wsdl:input name='getAvailableStocks' message='tns:getAvailableStocks_Request'/>
            <wsdl:output name='getAvailableStocks' message='tns:getAvailableStocks_Response'/>
        </wsdl:operation>
        <wsdl:operation name='getQuote' parameterOrder='p0'>
            <wsdl:input name='getQuote' message='tns:getQuote_Request'/>
            <wsdl:output name='getQuote' message='tns:getQuote_Response'/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name='StockQuoteService' type='tns:StockQuoteService'>
        <soap:binding transport='http://schemas.xmlsoap.org/soap/http' style='rpc'/>
        <wsdl:operation name='getAvailableStocks'>
            <soap:operation soapAction='' style='rpc'/>
            <wsdl:input name='getAvailableStocks'>
              .
              .

            <wsdl:output name='getQuote'>
                <soap:body ... namespace='http://sample.com/stock/'/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name='JavaService'>
        <wsdl:documentation>
            This is a simple stock service.
        </wsdl:documentation>
        <wsdl:port name='StockQuoteService' binding='tns:StockQuoteService'>
            <soap:address location='http://localhost:6061/StockQuoteService/'/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Figure 5.2: A sample WSDL and its functional elements

In WSDL, the functional information is contained within the natural language name phrases of its elements - *<service>*, *<documentation>*, *<input>*/*<output>* messages, *<operation>* etc. Figure 5.2 shows a sample WSDL and its constituent elements. The name of each WSDL element is specified in natural language, generally as a combination of two or more words to completely represent its functionality. If processed intelligently and extracted correctly, the functionality and domain of a service could be derived effectively, thus overcoming the problem of unavailability of semantic Web service descriptions.

In summary, using the functional semantics of a service, some form of metadata has to be generated, so that indexed services can be represented intelligently with the $\mathcal{DWDS}$ repository. A detailed study of available metadata formats was conducted and the Web 2.0 style of tagging was found to be well-suited. However, Web 2.0 applications depend on users who manually tag the data (for e.g, tags used for blog posts on Blogger/Wordpress, photos on Flickr, Facebook etc). In case of the $\mathcal{DWDS}$ service repository, the focus is on automatic functional semantics extraction and service tag generation, without requiring user intervention. The generated tags are used as representative metadata for each service, based on which they can be categorised and efficiently retrieved. To this end, the techniques employed in $\mathcal{DWDS}$ to extract a service's functional and domain information from its WSDL are discussed in Section 5.4.

5.4 Capturing the Functional Semantics of Services

In $\mathcal{DWDS}$, services retrieved from the Web are indexed in the WSDL Preprocessor database after performing the various checks (discussed in Chapter 4). The next processes of metadata generation and categorisation are to be performed for this service collection, to enable effective management. Firstly, the WSDL is to be parsed and processed using NLP techniques for extracting the functional information contained within it, and this will be used for generating representative metadata for each service. This metadata is intended to capture the domain-specific terms associated with each service, so that they can be grouped as per their domain.

For metadata generation, the WSDL description of the indexed services is considered. From figure 5.2, it can be observed that the WSDL elements have natural language names, typically a combination of two or more words, to indicate their functional aspects. Standard programming conventions dictate that good naming practices should be followed to ensure understandability while naming such service elements. Here, an operation *'getAvailableStocks'* that serves a request for returning all available stock information is named using *pascal-casing*. Some other naming conventions that can be used - *camel-casing* (*'GetAvailableStocks'*), underscores as term separators (*'get_ available_ stocks'*) etc. A combination of one or more these techniques is also used often. Extracting these element names from the WSDL is the starting point of the metadata generation process.

Figure 5.3 depicts the process of metadata generation for the services indexed in the WSDL Preprocessor database. It consists of three stages - *Feature Extraction*, *Service Tagging* and *Service Similarity Computation*. Each of the stages and their sub-tasks are described in Section 5.4.1.

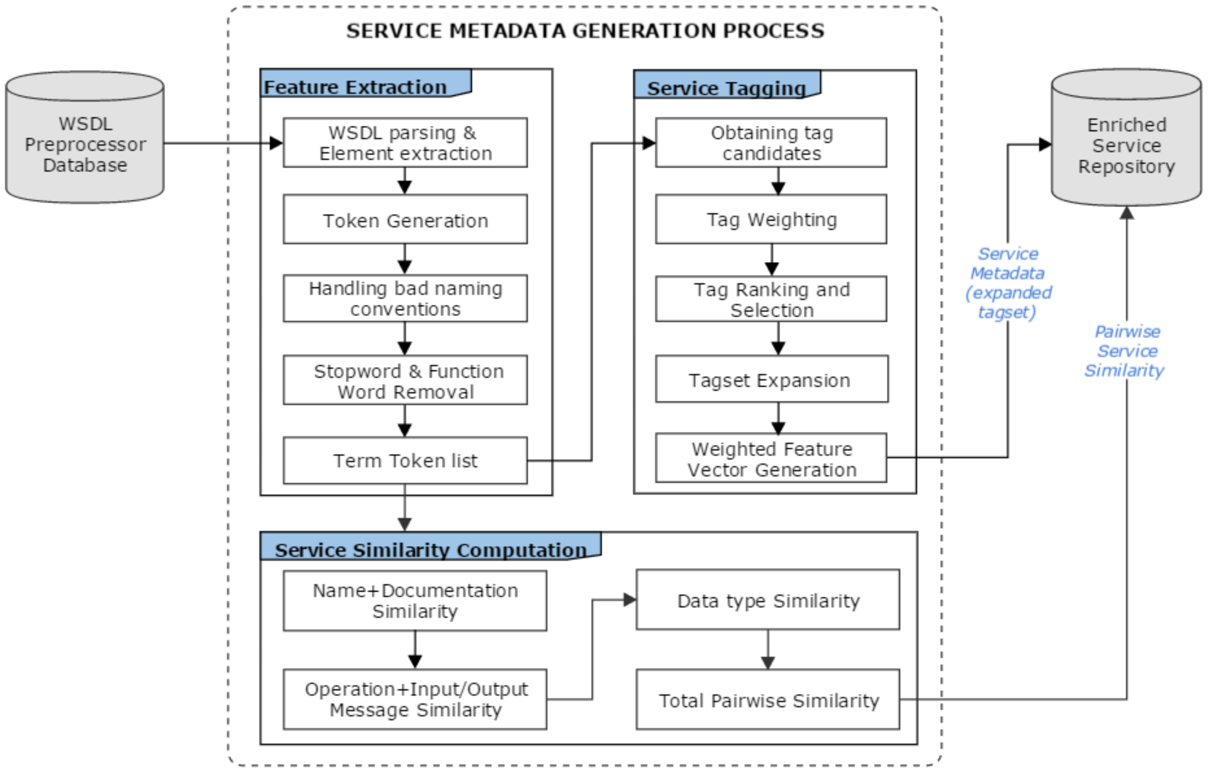

SERVICE METADATA GENERATION PROCESS
WSDL Preprocessor Database
Feature Extraction
WSDL parsing & Element extraction
Token Generation
Handling bad naming conventions
Stopword & Function Word Removal
Term Token list
Service Tagging
Obtaining tag candidates
Tag Weighting
Tag Ranking and Selection
Tagset Expansion
Weighted Feature Vector Generation
Service Metadata (expanded tagset)
Enriched Service Repository
Pairwise Service Similarity
Service Similarity Computation
Name+Documentation Similarity
Operation+Input/Output Message Similarity
Data type Similarity
Total Pairwise Similarity

Figure 5.3: Process of metadata generation for services in $\mathcal{DWDS}$ repository

5.4.1 Feature Extraction

The purpose of the functional semantics extraction process is for generating its representation features. A feature is defined as the *"distinctive attribute or characteristic of an object"*. In case of WSDLs, the distinctive features are their elements that indicate the service functionalities, that is *<service>*, *<documentation>*, *<portType>*, *<operation>*, *<input>/<output> messages* and *<types>*. These can be used for discovering useful information about a service's domain and function. Hence, a *feature extraction* process is used to parse the WSDL and extract all the useful elements required for further analysis. The process is performed as follows.

(i) WSDL parsing & Element Extraction. For every service indexed in the repository, the WSDL documents are parsed and various elements like service name, documentation, operation names, input/output messages and types are extracted and are used as *features*. As seen from Figure 5.2, each feature's name is a combination of natural language words, and 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. The name of each such element extracted is split into tokens. For example, an operation of the sample service shown in the example is named *'getAvailableStocks'*. This phrase cannot be considered as one token as it is not a valid English word, so three *term tokens* given by {*'get', 'available', 'stocks'*} are generated. To maintain consistency during token generation, certain rules are followed, which are explained next.

(ii) Token Generation. To ensure that the name of each service element is properly captured, certain name splitting rules are followed -

Case 1: Service element names are split into term tokens by considering *capital letters* as the start of a new token and *special characters* as token separators.

Case 2: Contiguous capital letters are considered as a single token as these mostly represent acronyms (e.g. SMS, SSN, ISBN etc).

Case 3: If no capital letters or special characters mark the beginning of a new word in the element name, proper splitting positions cannot be found. In this case, these element names need to be processed specially to identify tokens correctly.

Algorithm 5.1 illustrates the process of term token generation. Table 5.1 summarizes the rules followed in each case and the term vector obtained after feature extraction. In the first two cases, splitting positions can be easily determined and terms can be obtained. For example, consider an example where a feature is the name of the service *'Academic-degreeScholarship'*, which results in the term tokens *'academic', 'degree'* and *'scholarship'*. Similarly, the service's operations are also considered as a feature, so after splitting, an operation named *'getAcademic-degreeType'*, will result in *'get', 'academic', 'degree'* and *'type'*, which are the token list for that feature of a service.

Table 5.1: Term vector generation process - Examples

Extracted Feature	**Splitting Rule(s)**	**Term tokens**	**Feature vector**
filmAction	Pascal-casing	{*film, action*}	{*film, action*}
BookAuthor	Camel-casing	{*book, author*}	{*book, author*}
latitude1	Suffix number elimination	{*latitude*}	{*latitude*}
Book_Price	Underscore operator	{*book, price*}	{*book, price*}
getFamily-roomPrice	Pascal-casing; Dash operator	{*get, family, room, price*}	{*family, room, price*}
SMStoIndiaRequest	Contiguous capital letters; Pascal casing	{*sms, to, india, request*}	{*sms, india*}
findluxuryhotel-service	Dash operator; standard naming conventions not used	{*findluxuryhotel, service*}	Additional processing required.

Algorithm 5.1 WSDL Element Extraction & Term Token Generation process

```
1: procedure TokenGeneration(listofWSDLs)
2:     Parse WSDL of service $S$ and extract element names.
3:     $listofElementsofS \leftarrow element\ names$
4:     $listofTokens \leftarrow \emptyset$
5:     for each name_phrase $np$ in $listofElementsofS$ do
6:         if camel-casing/pascal-casing etc then
7:             Split into tokens as per rule (1)
8:             if special characters then
9:                 Split into tokens as per rule (2)
10:                if contiguous capital letters then
11:                    Split into tokens as per rule (3)
12:        else
13:            Further process name_phrase                ▷ as per Algorithm 2
14:        return list of generated tokens.
```

(iii) Handling Bad Naming Conventions. As seen from the example in last row of Table 5.1, we observed that some real world services in the service collection had quite unkempt element names. For example, consider a service that does not follow standard naming conventions - *'citycountrySkilledoccupation_Service'*. One of its operations is named as *'get_skilledoccupation'*, which if considered as a single token will result in incorrect tokens, that can adversely affect the accuracy of metadata generation. To correctly generate term tokens for this particular name, it has to be split into *'city'*, *'country'*, *'skilled'*, *'occupation'* and *'service'* tokens. For extracting tokens automatically from such bad names, we used a dynamic programming approach to select the splitting positions, that exploits relative word frequencies in English as per Zipf's law (Newman 2005). Zipf's law states that,

> *"The probability of encountering the r^{th} most common word in the English language is given roughly by $P(r) = \frac{0.1}{r}$ for r up to approximately 1000".*

Zipf also noted that the law does not hold good for less frequent words due to the divergence of the harmonic series. Hence, assuming the relative word frequencies follow Zipf's law, one can obtain the probability of a word by $\frac{1}{r \cdot log\ N}$ where r is the rank of the word in a dictionary of N words sorted by their relative word frequencies. This concept is used to resolve badly named service elements such as in the example discussed previously. Algorithm 5.2 illustrates this approach.

Algorithm 5.2 Term token generation from badly named WSDL elements

```
Input:
    $L$ = list of badly named WSDL elements
    $D$ = custom dictionary consisting of words, successfully extracted from other indexed
        services, sorted by their relative frequency of occurrence.
Output: List of correct term tokens, $term\text{-}list$
    $term\text{-}list = \varnothing$
    for each badly named element in the list $L$ do
        string $str = bad\text{-}element\text{-}name$
        $start\text{-}index = 0$
        $end\text{-}index = 0$
        for $i = (str.length - 1)$ till $i$=0 do
            $end\text{-}index = i$                                ▸ Start from the end of the bad-element-name string
            $term$ = splice($str$, $start\text{-}index$, $end\text{-}index$)        ▸ check if spliced text exists in D
            Check in dictionary $D$ for occurrence of $term$
            if $found$ = true then                                  ▸ Dictionary contains term
                Add $term$ to $term\text{-}list$
                $start\text{-}index = i$                          ▸ Reset current position to end of extracted term
                $i = (str.length - 1)$                            ▸ Find current length of bad-element-name string
            Decrement $i$ by 1                                       ▸ Start checking for next valid term
    return $term\text{-}list$
```

A custom dictionary D modelled specifically for our service dataset, containing words successfully extracted from correctly named services, was used. The dictionary consists of the words extracted successfully from well-named service features, which is then used for inferring correct splitting positions for the badly named elements. Assuming that all words in the corpus are independently distributed, the words in D are sorted as per their relative frequency of occurrence. Then, a word with rank r in the D will have a probability of occurrence as given by Zipf's Law. Using this approach, the splitting positions could be determined in near linear time ($\mathcal{O}(n + k)$), where n is the number of characters in the bad WSDL element name and k is the size of the word to be found; both n and k being small values). As the exact domain was modelled, i.e., service dataset specific terms were used in the custom dictionary D, this method achieved nearly 94% accuracy. However, if some generic corpus of English language words was used, then the results of this approach would not be as effective due to inaccurate domain modelling.

(iv) Stop word & Function Word Removal. Once all the term tokens for all features of a service are generated, the stop words (like *'get', 'by'* etc., that do not contribute to the semantics of a service) are filtered out and a term vector is obtained for each feature. In addition to this, words used commonly in Web services domain, referred to as *function words* (for e.g. *'service', 'input', 'output', 'request', 'response', 'SOAP', 'parameter'* etc) are also filtered as they contribute very little to the context of a service.

(v) Term Token list. The resulting list after stopword & function word removal is considered the final term token list for the service. The final term token list represents the functional information of each service and is considered the initial feature vector for that particular service.

Using these feature vectors, some metadata has to be generated for the services, so that, their functional and domain information is captured. Using this tag candidate list, the next stage of automatically generating tags for each service is carried out, as described in Section 5.4.2.

5.4.2 Automatic Service Tagging

During the previous stage, a set of terms are generated and a feature vector is obtained from each WSDL element name. Each of these terms represent the service's functional information in some way, but all such terms cannot be tags for the service. To determine the most relevant tags for a given service that would capture its functional semantics best, automatic tag weighting and ranking techniques were designed, and these are discussed in this section.

(i) Obtaining Tag Candidates. For this process, the terms extracted from the service's WSDL are considered as potential tag candidates. For example, the natural language documentation for the *ZipCodeLookup* service, given as *"Returns latitude and longitude for a given US city zipcode"* contains the noun phrases *'latitude', 'longitude', 'US city zipcode'*. Similarly, from one of its operations *'getZipcodeforCityState'*, the noun phrases *'zipcode', 'city'* and *'state'* are obtained. These, together with all the other noun phrases discovered from the element names of a particular WSDL are potential tag candidates.

Generally, the tag candidates may contain the same word in different forms. Lemmatization is a process of identifying and grouping together the different forms of a English language word so they can be analysed as a single item. Examples are words like *'books', 'booking', 'booked'* which are different forms of the word 'book'. This ensures that different variants of a term are reduced to only one. This also reduces the vector size, as the unique terms that make up the document are reduced. We used the WordNet Lemmatizer available in Python's NLTK for performing lemmatization.

(ii) Tag Weighting. The set of potential tag candidates of a service may contain several terms, all of which cannot be used as tags. To find the most relevant terms, the importance of each term to the service functionality should be considered. For this, a term weighting method called Tf-idf (term frequency-inverse document frequency) is used to compute the importance of a particular tag candidate to a given service when compared to its importance in other services in the dataset. Tf-idf values are computed as follows.

$$weight_t = (0.5 + \frac{tf}{tf_{max}}) \cdot log(0.5 + \frac{D}{df}) \tag{5.1}$$

where, tf is the term frequency of a tag candidate t in the given WSDL document d; tf_{max} is the highest frequency of any term in d, df is the frequency of occurrence of the term t in other WSDL documents and D is the number of WSDL documents in the dataset.

Next, the *Tf-idf matrix* is calculated and is stored in `.csv` files. In order to calculate the semantic relatedness term matrix for terms obtained in the previous step, WordNet is used. For each pair of tags, a list of associated words is constructed. Semantic similarity is calculated between each pair of terms in the lists and a weighted average is produced. This *semantic relatedness matrix* is also stored in a `.csv` file.

(iii) Tag Ranking and Selection. The tf-idf matrix will have a very high dimensionality as it captures the relative importance of all tag candidates generated for a particular dataset, with respect to each service. However, very few tags will be relevant for a given service, hence it is important to identify these terms only. In order to obtain a feature vector made up of the most significant terms, we used Singular Value Decomposition (SVD) (Klema et al. 1980), a popular feature reduction method. SVD is a Linear Algebra matrix decomposition technique, which is used to obtain a factorization of the two complex matrices - tf-idf matrix (U) and semantic-relatedness matrix (V). A factorization (M) over a given domain K is given by -

$$\mathbf{M} = \mathbf{U}\ \mathbf{\Sigma}\ \mathbf{V}^* \tag{5.2}$$

where, Σ is a $m \times n$ diagonal matrix with non-negative real numbers on the diagonal, and U is an $m \times m$, and V^* is an $n \times n$, orthogonal matrix over K. The diagonal entries, σ_i, of Σ are known as the singular values of M, listed in descending order.

The SVD of M, denoted by Σ, is a diagonal matrix, whose diagonal values are called singular values. These singular values represent the feature vector of each service, where, a value for a tag is a combination of syntactic and semantic importance of that tag in the service. This helps in effectively capturing the importance of a term in a service's feature vector. Hence, SVD helps in factorizing the Tf-idf matrix to identify that set of features which are most important for a particular service.

We used the Rapidminer Studio (Ertek 2013) for Tf-idf matrix generation and also for the SVD process. SVD has a complexity of $O(mn^2)$, where, m and n is the order of the U and V matrices to be multiplied. SVD was very effective in reducing the initial tag set and in identifying the ideal tag candidates that represented the functionality of a given service to the maximum extent. In our experiments, we initially obtained several thousands of tag terms. Applying SVD with a variance limit 0.95 (the allowed range is between 0 and 1), resulted in more than 4000 attributes. With further experiments, it was found that a variance value of 0.4 gave the most optimal results and 10 attributes were selected after the SVD process. Then, the tags were ranked as per their relative importance and based on their rank, the top 5 weighted terms were taken as the tag set and also the feature vector of each service.

(iv) Tagset Expansion. Once the set of 5 tags for each service is determined, that service is uniquely indexed by its tags. Hence, homonyms of these tags will also need to be considered for complete domain representation. For example, a service *'QuickCarRental'* service is tagged with {*car, quote, rent, location, date*}. The word *'car'* can be referred to as *'automobile'* or *'motorcar'* depending on the popular usage in different locations. If these additional terms are not considered as related terms, then the service may never be included in the results, even though it is relevant. To overcome this issue, a process of tag expansion is performed after obtaining the tag set.

For each tag in the tagset, the WordNet synsets are fetched. The tagset is then expanded by using these cognitive synsets, in order to capture synonyms of the words that were used as tags for each service. Finally, expanded tagset is used as the feature vector for each indexed service and is added to the service entry in the repository. This feature vector will be used to improve the service retrieval process for enabling fast matchmaking with the given user query.

Since all the services are now represented by their weighted tag set, a mechanism to identify similar services, so that domain specific categorization can be performed is required next. In order to do this, the similarity between different services has to be captured, so the most representative terms for a particular domain can be identified. Using the computed similarity values, similar services can be grouped, thus helping in reducing the search space for a particular query, as search can be directed towards the query-relevant domains and irrelevant groups can be neglected. The stage of service similarity computation is discussed in Section 5.4.3.

5.4.3 Service Similarity Computation

To capture the domain information of a service, the similarity between it and every other service in the repository is required.

To compute the similarity $sim(t_i, t_j)$ between any two terms t_i and t_j belonging to service s_i and s_j, we used WordNet (Miller et al. 1990). WordNet is an online lexical database that can be used to compute semantic relatedness between two concepts. It is a large lexical database of English, where nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (called *synsets*), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations and the resulting network of meaningfully related words and concepts makes it a useful tool for computational linguistics and natural language processing tasks.

WS4J (Shima 2013) is the Java implementation of WordNet::Similarity (Pedersen et al. 2004) that allows the calculation of synset similarity based on the path distance between two words in the WordNet taxonomy. This can be then used for specifically computing semantic similarity and semantic relatedness using the WordNet taxonomy. The tool returns a similarity score for any two given WordNet synsets. The maximum similarity as given by WS4J between the pair-wise combinations of synsets of the two considered terms, extracted from each service is taken as the similarity value between them. Terms from the service name, documentation, input/output messages and operations are processed separately and the similarity between each term pair is recursively computed. Finally, the similarity between 2 feature vectors f_{s_1} of service s_1 and f_{s_2} of service s_2 of a feature f is given by Equation (5.3).

$$sim_f(s_1, s_2) = \frac{\sum_{i=1}^{i=|f_{s_1}|\cdot|f_{s_2}|} \{\alpha \cdot (1-\alpha)^{i-1} \times sim^i(t_1, t_2)\}}{\sum_{i=1}^{i=|f_{s_1}|\cdot|f_{s_2}|} \{\alpha \cdot (1-\alpha)^{i-1}\}} \tag{5.3}$$

where, term $t_1 \in f_{s_1}, t_2 \in f_{s_2}$ and α is a constant less than 1. $|f_{s_1}|$ and $|f_{s_2}|$ represents the number of terms in f_{s_1} and f_{s_2} respectively. $sim^i(t_1, t_2)$ is the similarity corresponding to the i^{th} index in an ordered list of decreasing values from the set $S = \{sim(t_p, t_q) | t_p \in f_{s_1}, t_q \in f_{s_2}\}$. This is done to emphasize similar pairs of tokens in the similarity computations.

We illustrate the need for an emphasis on similar pairs of tokens in the similarity computations, instead of taking the average value of S as was the case in of Elgazzar et al. (2010) with an example. Consider three term vectors, T_1 = [*book, price*], T_2 = [*horror, book, recommended, price*] and T_3 = [*car, price*] of services s_1,s_2 and s_3 respectively. Intuitively, T_1, T_2 are more similar in comparison to T_1, T_3. However, when similarities were computed by Elgazzar et al's approach, it was found that $sim_T(s_1, s_2) = 0.42 <$

$sim_T(s_1, s_3) = 0.55$. Similarities computed by Equation (5.3) gave correct results i.e., $sim_T(s_1, s_2) = 0.83 > sim_T(s_1, s_3) = 0.74$. Also, over a large set of similarity values computed, the values got by Equation (5.3) were more distributed in the range of 0 to 1.

Finally, the total similarity between any two service descriptions s_1 and s_2 can be computed by Equation (5.4) below.

$$\begin{aligned} sim_{wsdl}(s_1, s_2) = vw_1 \cdot sim_{servicename}(s_1, s_2) + vw_2 \cdot sim_{documentation}(s_1, s_2) + \\ vw_3 \cdot sim_{operations}(s_1, s_2) + vw_4 \cdot sim_{messages}(s_1, s_2) + vw_5 \cdot sim_{types}(s_1, s_2) \end{aligned} \tag{5.4}$$

where, vw_1, vw_2, vw_3, vw_4 and vw_5 are variable weights whose summation should be equal to 1. The values of $sim_{service}(s_1, s_2)$ and $sim_{documentation}(s_1, s_2)$ are calculated by equation 5.3 and $sim_{operation}(s_1, s_2)$ is calculated by equation 5.5.

$$sim_{operation}(s_1, s_2) = \frac{\sum_{o_i \in O_{s_1} \cup O_{s_2}, o_i \notin O_{s_j}, j \in \{1,2\}} sim_{oo}(o_i, O_{s_j})}{|O_{s_1}| + |O_{s_2}|} \tag{5.5}$$

Here, similarity between o_i and the operation set of s_j, O_{s_j}, is given by -

$$\begin{aligned} sim_{oo}(o_i, O_{s_j}) = Max_{o_k \in O_{s_j}}(0.4 \cdot sim_{o\ name}(o_i, o_k) + 0.3 \cdot sim_{o\ input}(o_i, o_k) \\ + \ 0.3 \cdot sim_{o\ output}(o_i, o_k)) \end{aligned} \tag{5.6}$$

where, $sim_{o\ name}(o_i, o_k)$ is calculated using Equation (5.3). Similarity $sim_{o\ i/o}(o_i, o_k)$ between input/output messages with operations o_i, o_j is computed as the average of $sim_{i/o\ name}(o_i, o_j)$ (calculated by Equation (5.3)) and $sim_{i/o\ type}(o_i, o_j)$ is given by

$$sim_{o\ i/o\ type}(o_i, o_j) = \frac{2 \cdot |type_{o_i} \ \cap \ type_{o_j}|}{|type_{o_i}| + |type_{o_j}|} \tag{5.7}$$

Here, $type_{o_j}$ is the set of data types of input or output used in operation o_j. Using these equations, the total similarity between two services is given by a summation of the values of each element's similarity and is calculated using Equation (5.4). This similarity value can be later used for enabling service domain-specific categorization.

5.5 Service Categorization using Similarity

The process of generating metadata in the form of tags is helpful for capturing the inherent semantics of a service to create its representative tags, using which the services can be better indexed. Another advantage is that services with same/similar tags can be considered to be belonging to the same domain and hence grouped, thus enabling service categorization. In this section, we discuss two different techniques used to test the suitability of the generated tags for automatic categorization. We applied both unsupervised techniques (*clustering*) and supervised techniques (*classification*) for this purpose. In Section 5.5.1, we discuss the tag based clustering process and in Section 5.5.2, the classification process is presented.

5.5.1 Similarity based Clustering

In this section, a method to cluster services using the extracted features, computed similarity and generated tags is described. Using the computed similarity, the services are clustered using Hierarchical Agglomerative Clustering (HAC) (Hastie et al. 2009). HAC uses *"a bottom up approach, where each entity in the considered dataset starts in its own cluster, and pairs of clusters are merged as the algorithm considers entities further up the hierarchy, based on their similarity"*.

Algorithm 5.3 Hierarchical Agglomerative Clustering

```
Input: Set of WSDLs
Output: Service clusters
1: Consider each service as a separate cluster
2: for every service in individual clusters do
3:     Calculate similarities between clusters
4: for each service description do
5:     while similarity value ≥ $t_{merge}$ do        ▸ $t_{merge}$ = largest Dunn's Index value
6:         Select a pair of clusters that are most similar
7:         Remove pair from matrix and merge them
8:         Evaluate similarity of new cluster with all other clusters & update matrix
9:         Repeat for all services in matrix
```

Algorithm 5.3 illustrates the process of hierarchically clustering services using the computed similarity. HAC was applied a small dataset of 500 services selected from the $\mathcal{DWDS}$ service collection. In the first iteration, the pair of services with the maximum similarity value are merged to form a cluster. This process is continued and the pair of clusters with maximum similarity are merged in each iteration until the similarity value is lesser than some threshold. The merging of clusters stops when the similarity value falls below a given threshold. Instead of setting this to a pre-defined value, we provided a method to automatically set the threshold value based on the service similarity values. We used Dunn's Index (DI) (Dunn 1973), which is a cluster validity measure that identifies well separated clusters (calculated using Equation (5.9). DI is the ratio of the smallest distance between two objects from different clusters to the largest distance of two objects in the same cluster. Hence, the maximum DI value is the optimal value. As per this, the HAC merging threshold, t_{merge}, is set to the largest DI value.

Here, the similarity of a pair of clusters c_1 and c_2 is computed using Equation (5.8), based on which they are either merged (if value is more than t_{merge}), or left separate.

$$sim_{cluster}(c_1, c_2) = Max_{s_i \in c_1, s_j \in c_2}(sim_{wsdl}(s_i, s_j)) \tag{5.8}$$

The value of the Dunn's Index is calculated using Equation (5.9).

$$DI(C) = min_{i \in C}\Big\{min_{j \in C, j \neq i}\Big\{\frac{\alpha(c_i, c_j)}{max_{k \in C}\{\alpha(c_k)\}}\Big\}\Big\} \tag{5.9}$$

where, C represents the total set of clusters and $\alpha(c_i, c_j)$ is the smallest distance between two services belonging to different clusters c_i and c_j, and is given by Equation (5.10). $\alpha(c_k)$ is the largest distance between two services in the same cluster c_k, and is given by Equation (5.11).

$$\alpha(c_i, c_j) = min\{1 - sim_{wsdl}(s_i, s_j) | s_i \in c_i, s_j \in c_j\} \tag{5.10}$$

$$\alpha(c_k) = max\{1 - sim_{wsdl}(s_i, s_j) | s_i, s_j \in c_k\} \tag{5.11}$$

The complexity of the HAC algorithm is $O(n^2 \ log \ n)$ since the similarity values have to be computed for each service pair and then checked with the computed DI value before deciding if two services belong to the same cluster. Since the highest DI value is used to set the merging threshold, the algorithm continues merging clusters while the similarity value is above t_{merge} When this condition is not met, the process stops and this indicates the formation of well defined, well separated clusters.

5.5.1.1 Cluster Tagging

After HAC, a set of service clusters are obtained. Now the task is to identify the domain of the services within each cluster and tag the clusters appropriately, for improving the efficiency of the retrieval process. To generate tags for each cluster, initially, we consider all service tags of services within a cluster as potential tag candidates. Then, a ranking process is applied to these tag candidates by considering two factors:

- how well a tag g represents service s is equal to the $Tf\text{-}idf$ value of g in s. (represented as $sim_{tag}(g, s)$).

$$sim_{tag}(g, s) = Tf\text{-}idf(g, s) \tag{5.12}$$

- how well the service s whose tag is under consideration represents its cluster c (represented as $csim_{service}(s,c)$) and is computed as follows:

$$csim_{service}(s,c) = \sum_{i=1, s_i \neq s}^{i=|c|} \frac{sim_{wsdl}(s,s_i)}{|c|-1} \tag{5.13}$$

where $|c|$ represents number of services in cluster c.

Then, the level of similarity between a cluster tag g and its cluster c (denoted by $sim_{clusterTag}(g,c)$) is given by -

$$sim_{clusterTag}(g,c) = \sum_{g \in \tau(s), s \in c} (sim_{tag}(g,s) \times csim_{service}(s,c)) \tag{5.14}$$

where, $\tau(s)$ is the set of tags for service s. Based on the resultant ranking, the top-5 tags are chosen as cluster tags. In addition to this, a cluster tag expansion process is performed, by adding synsets provided by WordNet for each tag in the cluster tag set. These together form the feature vector of each cluster, which will be utilized during the query-to-service matching process. The results of the suitability of using service similarity and service tags to generate cluster tags, is presented in Section 5.6.2.

5.5.2 Similarity based Classification

Similar to the way tag based clustering was performed; we discuss a method to classify services using the extracted features, computed similarity and generated tags. Firstly, using the features extracted, the similarity between Web services is computed as discussed in Section 5.4.2 and the service tags are generated as described in Section 5.4.3. Then, we apply various machine learning techniques to the service tags to classify services into relevant categories and finally generate class tags for each class. We used six different classifiers available in the Weka Data Mining Suite (*Weka 3 - Data Mining Software in Java* n.d.) for classification of the services into domain specific categories.

1. *Naïve-Bayes*: Naïve-Bayes is a simple probabilistic classifier based on Bayes' theorem, that assumes independence between the considered features (Murphy 2006). Here, the assumption is that each feature has an independent contribution in the probability. For example, consider that an animal can be called as a dog if it has four legs and it barks. In Naïve Bayes, there is no correlation between the *number of legs* and *barking* features, which often leads to false results.

2. *Decision tree*: This classifier uses a decision tree for building a predictive model

(Swain et al. 1977). Here, numerous test questions and conditions are modelled as a tree structure, where each internal node and the root of the tree denote a testing attribute while each leaf corresponds to a class label. When some input variables are given, the class label can be predicted by traversing the tree such that the values of the given input variables represent the path from the root to leaf.

3. *Random forests*: Random forests is an ensemble classifier which involves growing many classification trees (Breiman 2001). A service is classified by assigning its input vector to each of the trees. Each tree assigns a class label to the service. This process is called as voting as each tree votes for a class label. The forest then chooses the class label that obtained the highest number of votes.

4. *Bagging*: Bagging is another ensemble classifier that applies base classifiers on random subsets of the original dataset (Skurichina et al. 2002). The subsets are made by drawing random samples and replacing them at later stages of the classification. Each base classifier gives its individual predictions and these are then aggregated either by averaging or voting.

5. *Regression*: Regression is a probabilistic statistical model which uses the relationship between the categorical dependent variable (which needs to be predicted) and various other independent variables (Hosmer et al. 2000). Regression analysis helps in understanding the variation in the value of a dependent variable when any one of the independent variables are varied, keeping all other independent variables fixed. Hence, the focus is on the relationship between a dependent variable and one or more independent variables.

6. *Artificial Neural Network (ANN)*: ANNs (Bishop 1995), also called Multilayer Perceptrons, are inspired by biological nervous systems (i.e., the brain), and are modelled as a layered network comprising of highly interconnected processing elements called *neurons*. Neurons work together in order to approximate a specific transformation function. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process during which the weights of the inputs in each neuron are updated. The weights are updated by a training algorithm, such as *back-propagation*, based on previous iterations, to reduce the value of an error function.

The classification accuracy was determined using a 10-fold cross validation method (Kohavi 1995) method. Cross validation is a method of accuracy estimation of a model when applied to a dataset and is very useful in case of limited data availability. In such a situation, if enough data is held for training to construct a good model then testing would not yield reliable results. The most common cross validation technique is k-fold cross validation, where, the dataset is divided into k parts with a homogeneous distribution of classes. Out of the k parts, one is retained for testing while $(k-1)$ parts are used for training. This process is performed k times with different testing partitions each time. The final accuracy is calculated by taking the average of k accuracy scores for each fold. Using the classes generated by the most accurate classifier, we then applied Equations (5.12), (5.13) and (5.14) to generate class tags, based on each class's member services (by substituting 'class' for 'cluster' in the equations). The results of the classification and accuracy achieved with various classifiers is discussed in Section 5.6.3.

5.6 Experimental Results and Discussion

In this section, we present the experimental evaluation of the proposed approaches for functional semantics extraction, metadata generation and for the service categorization using the service tags. First, an evaluation of the quality of service tagging based on the extracted functional semantics is presented in Section 5.6.1. Next, the suitability of the proposed similarity computation and service tagging approach for service categorization was evaluated. The results of the evaluation for service clustering is presented in Section 5.6.2 and results of service classification is presented in Section 5.6.3. Finally, a simple querying mechanism was used to measure the improvement in service retrieval process due to the addition of tags to the services, when compared to services indexed without generated tags. The performance of web service retrieval using the untagged services and tagged services, in the form of precision and recall is discussed in Section 5.6.4.

5.6.1 Quality of Service Tagging

As stated earlier, the Zipf's Law based approach for determining splitting positions for token generation from badly named service elements achieved more than 94% accuracy. To evaluate the efficiency and quality of tagging services and classes, we conducted several experiments. Since tags are natural language text, we used manual inspection techniques for a sample random subset. Several services were selected randomly from the service repository and the service tag quality was subjected to human validation, to determine relevancy of tags generated for each service. The results of the check are presented here.

Table 5.2 lists the tags generated for one such sample service *CustomerSupportService* available in our repository, provided by *carrentals.com*, using which customers can request for email support from customer service team. The table also shows the top-5 tags for the service and their associated tag weights (discussed in Section 5.4.3). The tags are ranked by their weights and the top-5 tags for this service are {*'support', 'customer', 'email', 'car 'rent'*}.

Considering another service, *calculateDistanceInMiles*, we list the service's generated tags and the corresponding tag weights in Table 5.3. It can be seen that the system generated meaningful tags that effectively represented the functionalities of individual web services. As these tags represent the functional semantics of a service, it can be concluded that the metadata generation process was effective.

Table 5.2: Tags and associated tag-weight for a service *CustomerSupportService*

Tags	**Weight**
support	2.26948500217
customer	2.0170629812
email	1.59481500924
car	1.34948500217
rent	1.34948500217

Table 5.3: Tags and associated tag-weight for the service *calculateDistanceInMiles*

Tags	**Weight**
mile	2.48886180264
calculate	2.32660625689
distance	1.5484550065
geographic	1.14360118508
latitude	0.715121673171
longitude	0.715121673171

5.6.2 Service Clustering Performance

Using the tags generated by the system and the similarity computation mechanism (discussed in Section 5.4.2), the services were clustered using HAC. A subset of 500 services was chosen from the service repository for this purpose. The HAC algorithm was applied to these services and the clustering obtained was observed.

It was seen that best results were obtained for $vw_1 = 0.25$, $vw_2 = 0.15$, $vw_3 = vw_4 = vw_5 = 0.2$ and $\alpha = 0.4$. The maximum Dunn's Index value was 0.010042, for which

Table 5.4: Some generated clusters and corresponding cluster tags

Cluster No.	**Services in Cluster (WSDLs)**	**Cluster Tags**
1 (7*)	AcademicAddress; researcher-postal-address-service; AcademicPostal-address; professor-in-academia-,service; researcher-address-service; Educational-employeeAddress; EmployeePostal-address1	*address, post, academic, research, employee*
2(10*)	ProvideMedicalFlightInfo-service; Booknonmedicalflight ProvideMedicalTransportInfo-service; GetNonMedicalFlightAccount-service; GetNonMedicalTransportAccount-service; GetMedicalFlightAccount-service;Bookmedicaltransport GetMedicalTransportAccount-service; Booknonmedicaltransport; Bookmedicalflight	*medical, book, account, flight, transport*
3(7*)	BookPrice; EncyclopediaPrice; BookAuthorprice1; EncyclopediaAuthor; BookAuthor1; Author1; publication-author-service	*author, book, price, encyclopedia, publication*
4(9*)	getAltitudeOfLocation; getElevation-FromLocation; getAltitudeAboveSeaLevel; addressDistanceCalculator getDistanceBetweenCities-Worldwide; getDistanceBetweenPlaces; calculateDistanceInMiles; getDistanceBetweenLocations; calculateDistance	*distance, geographic, location, longitude, latitude, altitude*
5(3*)	retailstore-preparedfoodquantity-service; retailstore-foodquantity-service.wsdl; retailstore-foodquality-service.wsdl	*store, retail, food, quality, quantity*

** Number of services in each cluster*

the corresponding t$_{merge}$ value was calculated to be 0.65. This yielded 87 clusters with average cluster size 5.632184. About 87.96% of the web services in the considered dataset were in top 32% of the clusters formed. Next, cluster tags were generated for each of the clusters using the method described in Section 5.5.1. A subset of the clustering results obtained and the corresponding cluster tags for some selected clusters are tabulated in Table 5.4. As seen from the tabulated results, the clustering accuracy was good and also the tags generated for each cluster from its member services tags were meaningful.

5.6.3 Service Classification Performance

For classification, a training dataset is required for which the category information is already known. For this experiment, a standard dataset called the ASSAM dataset[1] was used, which contains 448 service descriptions manually categorized into 25 broad categories, each subdivided into several subcategories. For example, *postal* services are categorized into '*Mail*' category and '*Mail*' is a type of '*Communication*' etc, so the dataset is hierarchically organized. The values $vw_1 = 0.3$, $vw_2 = 0.25$, $vw_3 = vw_4 = vw_5 = 0.15$ and $\alpha = 0.4$ were used for computing WSDL similarity and generate feature vectors. SVD was performed on the full feature vectors and the reduced feature set of each service was obtained. Next, the classification process was performed and the accuracy was determined using 10-fold cross validation (Kohavi 1995).

The Weka classifiers used for experimental evaluation of service classification accuracy using the feature vectors generated before SVD were - Bagging, Näive Bayes, Decision Tree, Random Forest and Classification by Regression (CbR). The ANN was used only after dimensionality reduction due to its complex nature. It was seen that Näive Bayes, Decision tree and Bagging achieved almost similar accuracy, in the range of 68-70%, while Regression achieved an accuracy of around 74%. Since regression is based on assignment of weights to features, it can be inferred that following a similar approach before classification may result in higher accuracy. For the Random Forests classifier, the observed accuracy is the highest and it is around 78%. It is an ensemble approach and works with a randomly selected set of features in each iteration. So, it can be inferred that some subsets of features tended to have higher accuracy than other subsets. If a feature selection method is applied on the vectors, it might achieve better accuracy.

The accuracy obtained for service classification using each classifier for the dataset before and after dimensionality reduction is shown in Figure 5.4. After applying dimensionality reduction, the accuracy of each of the classifiers improved by at least 7-8% in the case of most classifiers. We applied ANN as well in the second case, i.e. after dimension reduction. ANN takes a lot of time for training over a dataset with a large number of attributes, so it was not applied in the first case, when the dimensionality of the features was very high. In the case of the reduced feature set, ANN achieved the best accuracy of 88%. Hence, it can be concluded that similarity computation methods are suited for service categorization using classification.

[1]Available at http://www.andreas-hess.info/projects/annotator/ws2003.html

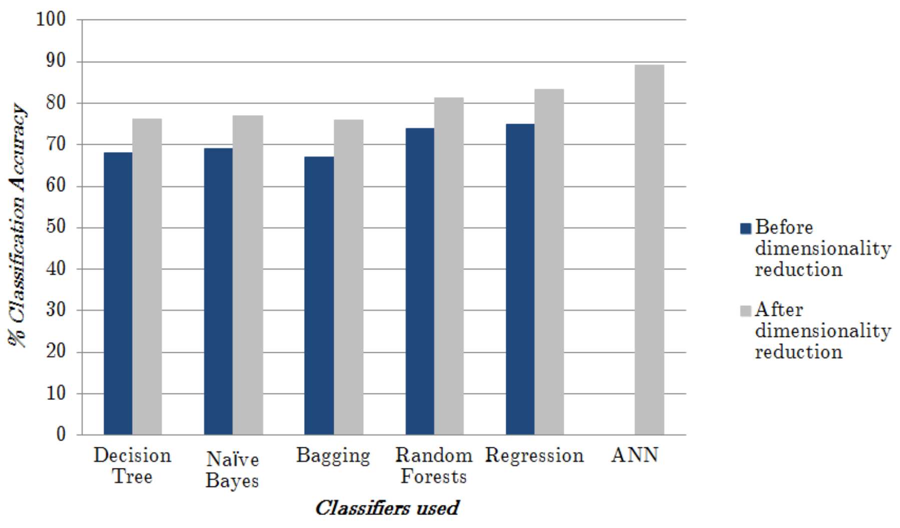


Figure 5.4: Service Classification Accuracy

5.6.4 Web Service Retrieval Performance

In this section, a simple querying mechanism is used to search for services based on keywords. The main objective of this is to verify that the tagging and similarity based categorization improves Web service discovery. For this purpose, we use HAC again to cluster the set of 500 services considered in Section 5.6.2, by using two different approaches as listed below -

i. Using WSDL-level similarity only (obtained from Equation (5.4)).

ii. Using both the computed WSDL-level similarity and the generated tag similarity.

Since the proposed system is modelled as per traditional IR methods, 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 successfully retrieved for the given query.

$$Precision = \frac{|relevant\ services \cap retrieved\ services|}{|retrieved\ services|} \tag{5.15}$$

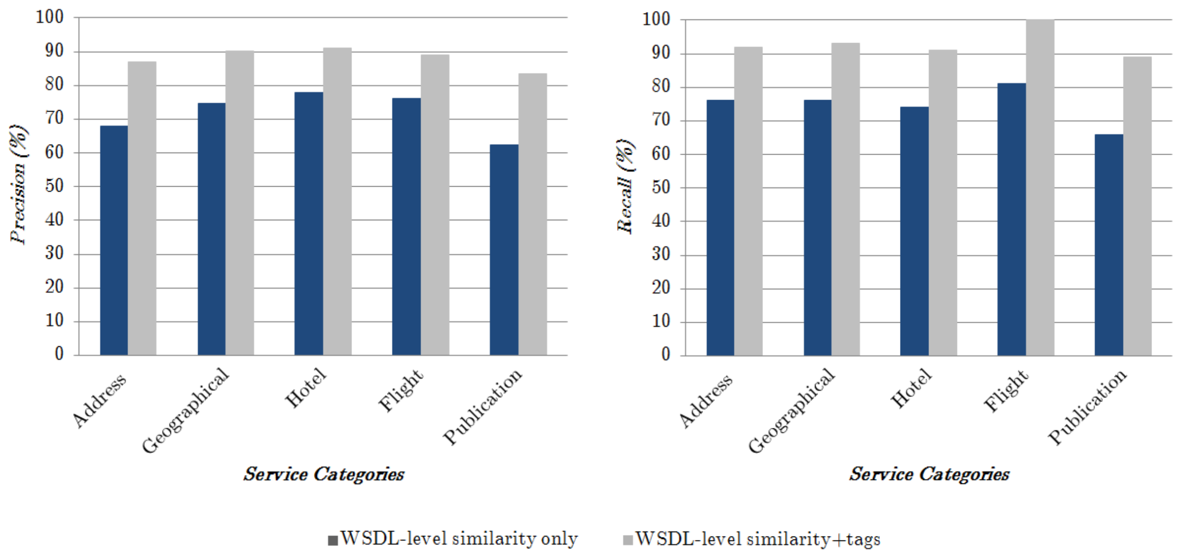
$$Recall = \frac{|relevant\ services \cap retrieved\ services|}{|relevant\ services|} \tag{5.16}$$

User input was taken through a simple form based GUI through which the users can specify their service requirement. We illustrate the service discovery process with a simple example as below.

Search terms: book price

Effective user request: "find a service that returns the price of a given book"

Based on the above service request, out of the 500 services in the dataset, 57 services were considered relevant and others were discarded as irrelevant, hence the search domain was reduced by 88.6%. Out of these 57 services, the system finally identified three services that were best possible candidates for serving the user request. The candidate set discovered consisted of the following web services - the services *BookPrice*, *BookAuthorPrice* and *EncyclopediaPrice* were returned as most relevant for serving the user query. Similarly, we used different queries to evaluate the overall performance. The average precision and recall values for each, over the different domains considered are shown in Figure 5.5. It was seen that the second approach which used both WSDL-level similarities and the generated tags to discover services achieved 14% improvement in precision and 16% improvement in recall over the first approach.



(a) Observed precision values (b) Observed recall values

Figure 5.5: Comparison between *WSDL-level similarity only* and *WSDL-level Similarity+tags* approaches

5.7 Summary

In this chapter, we presented an automated technique to generate representative metadata for services and to compute their similarity to enable service categorization. These techniques apply semantics based natural language processing techniques to the extracted elements of the service description to generate tags that capture the functional semantics of the service. The quality of the tagging was validated by manual inspection and was found to be meaningful. To further evaluate the effectiveness of these mechanisms, the computed service similarity and generated tags were used to categorize services using both clustering and classification. Then, tags were generated for the generated clusters/classes using weighting and ranking techniques on the tags of their member services. Experimental evaluation of this categorization validated that the categorization accuracy was quite good and the cluster/class tags generated, were able to effectively capture the domain information of a cluster/class. Further, we performed simple Web service discovery experiment and found that the computed similarity and tagging resulted in a 14% increase in precision and 16% increase in recall.

Publications

(based on work presented in this chapter)

1. Sowmya Kamath, S., and V. S. Ananthanarayana. "Similarity analysis of service descriptions for efficient Web service discovery", In Data Science and Advanced Analytics (DSAA), 2014 IEEE/ACM International Conference on, Shanghai, China., pp. 142-148. IEEE, 2014.

2. Sowmya Kamath S., and Ananthanarayana V.S., *"Towards Semantic Web Services: An Empirical Evaluation of Service Ontology Generation Tools"*, 12th IEEE India Conference on Electronics, Energy, Environment, Communication, Computer, Control $E^3 - C^3$, (IEEE INDICON 2015), pp.1-6, 2015.

3. Sowmya Kamath S and Ananthanarayana V.S, *"Semantics based Web Service Classification using Morphological Analysis and Ensemble Learning"*, International Journal of Data Science and Analytics, Springer, ISSN: 2364-4168 (Scopus and EI Indexed)

   Status: Accepted

Chapter 6

Dynamic Clustering of Web Services

6.1 Introduction

The problem of building a scalable Web service discovery framework and enabling automated metadata generation and categorisation techniques have been discussed in the previous chapters. The service repository of the framework is scalable, because it is based on the concept of a Specialized Service Crawler. This means that the number of services in the service repository can change over time, as demonstrated by the temporal statistics presented of the service collection (discussed earlier in Section 4.4.4). As a result, the service collection is inherently dynamic and to deal with this problem efficiently, an incremental change propagation based approach was proposed for keeping track of the different data entities.

The same problem is observed while the process of service categorisation is to be carried out. As the service collection (i.e., the dataset) may change after each scheduled periodic crawler run, services that are already categorised should effectively remain the same and only the new changes must be processed, thus reducing unnecessary rework. Supervised approaches like machine learning based classification is not suited for our service collection, as the domains of most indexed services are unknown, and hence a training dataset is difficult to provide. Unsupervised approaches include several traditional clustering approaches like K-Means, Hierarchical Clustering etc. The clustering process has to restart after any change in the underlying dataset, when these algorithms are used; hence these clustering approaches are not equipped to deal with a dynamic data collection.

Hence, there is a need to design an effective clustering algorithm that is capable of dealing with the dynamic nature of the $\mathcal{DWDS}$ framework to achieve domain-specific grouping, incrementally, as and when the changes happen, without any need for manual intervention. In this chapter, we discuss such an algorithm.

Our main contributions, as reported in this chapter are listed below:

- Developing a mechanism for large-scale, domain-specific categorisation of services using their functional semantics.
- Designing the algorithm to be able to deal with the dynamic nature of the proposed framework.
- Demonstrating that the proposed clustering approach performs better when compared to traditional algorithms and was effective in dealing with dataset changes incrementally.
- Evaluating the quality of clustering of the proposed clustering approach.

The rest of this chapter is organised as follows. Section 6.2 describes the defined problem addressed in this chapter and Section 6.3 discusses some background information in this regard. Section 6.4 discusses the solution methodology developed for categorising services using their functional semantics, while incorporating an incremental strategy to deal with the dynamic nature of the service collection. In Section 6.5, we present the experimental results on the performance of the proposed algorithm in comparison with traditional algorithms and evaluate the goodness of clustering achieved by it. Section 6.6 summarizes the work presented in this chapter.

6.2 Problem Statement

The problem that is addressed in this chapter is defined here:

> *Given the constant changes in the indexed service collection, to develop an efficient algorithm that can deal with the changes incrementally, to enable domain-specific service grouping.*

The solution proposed for this problem, aims at improving the process of managing the repository by automating the process of categorising services. Using the tags generated as metadata for each service, the service categorisation process has to discover the domain information of services, so domain-specific grouping can be performed. In addition to this, the categorisation has to be performed in an optimized manner, such that only the new changes in the service collection must be processed, instead of reprocessing the entire collection. In Section 6.3, we discuss some background information regarding the proposed methodology and then, describe the algorithms developed for performing dynamic service categorisation in Section 6.4.

6.3 Background

The problem of clustering a tagged dataset can be effectively handled by using traditional clustering algorithms like hierarchical clustering. However, a major limitation of such algorithms is that they are efficient only for small sized, static datasets. The $\mathcal{DWDS}$ service collection is inherently dynamic in nature as WSDLs are being continuously processed and added/deleted during periodic active and selective crawler modes. During each scheduled active crawling cycle, new service descriptions may be added. When selective crawling is performed, the availability status of service descriptions already added to the WSDL repository is checked and the service may be removed if continuously inaccessible. Hence, every small change in the WSDL dataset will mean that the clustering will have to restart if traditional clustering methods are used. Therefore, an efficient clustering strategy is needed to process the dynamic data in a cost-effective and time-aware manner.

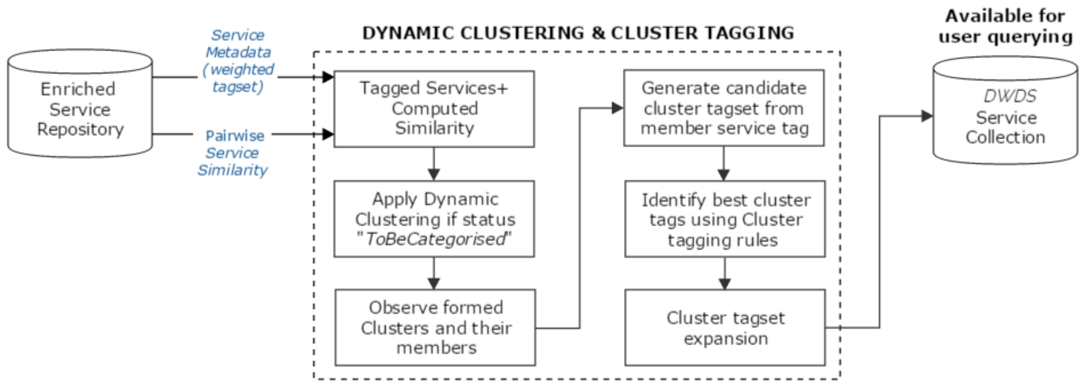
To mitigate this problem, the concept of incremental clustering (Can 1993) was adopted, which focuses on dealing with multiple (small) changes incrementally, rather than re-clustering the entire dataset. Currently, there exist incremental versions of some traditional algorithms like Incremental K-Means (Wagstaff et al. 2001) and Incremental DBSCAN (Ester et al. 1998). However, incremental K-Means assumes that the data distribution fits into a pre-specified number of clusters k. Similarly, for incremental DBSCAN, the best values of ϵ and *minpts* are determined by a trial and error method, and are also highly dependent on the data being clustered. Also, another main drawback of both incremental algorithms is the execution time required when dealing with large data sets of high dimensionality (Jain 2009).

Most existing work in Web service clustering deal with small, closed service datasets and have not been applied to larger collections. In real-world scenarios, service collections are dynamic as new services may be published and older ones may no longer be accessible, due to which, traditional clustering algorithms are not very suitable. Recently, algorithms inspired by animal behaviour in nature have been applied towards clustering dynamic datasets like streaming data and for dynamic information retrieval problems. These have achieved good results and are well suited for such optimization problems. The main principle of these algorithms is the tasks accomplished by the interactions of a large number of simple agents that can adapt to their environment and work towards a common goal. These agents can be *ants* as used in ant colony optimization algorithm (ACO), *particles* in particle swarm optimization (PSO), *bees* in artificial bee colony algorithm (ABC) or *populations of individuals* in genetic algorithm, which collectively exhibit intelligent behaviour.

Wang, Shen, et al. (2012) performed experiments to measure the adaptability of bio-inspired algorithms like ant colony optimization (ACO), particle swarm optimization (PSO), genetic algorithms etc., for web service composition and found them to be very well suited. They reported that these techniques when applied to the problem of web service candidate selection for composition were efficient and resulted in a significantly reduced search space. Since clustering for service discovery also has a similar objective of reducing the search space to effectively serve a user query, these findings are relevant. Since the proposed framework is inherently dynamic, there is a need for a dynamic clustering mechanism that can cope with the continuous changes in the dataset and process new data incrementally. Also, the clustering would be efficient as the functional capabilities of the services are best represented by its tag set, based on which they can be categorised most efficiently. In this context, bio-inspired approaches like Swarm Intelligence algorithms are best suited for high density and dynamic data. To this end, a novel bio-inspired incremental clustering algorithm for web service clustering is proposed and discussed in detail in Section 6.4.

6.4 Proposed Dynamic Clustering Approach

Figure 6.1 depicts the dynamic clustering methodology for the developed framework's service collection. The process of automatic service categorisation is dependent on two main processes - *Automatic Service Tagging* (described in section 5.4.2 of Chapter 5) and *Service similarity computation* (described in section 5.4.3 of Chapter 5). The *Dynamic Clustering & Cluster Tagging* phase is performed based on the computed similarity values and the generated service tags and is discussed in Section 6.4.1.



DYNAMIC CLUSTERING & CLUSTER TAGGING
Available for user querying
Enriched Service Repository
Service Metadata (weighted tagset)
Pairwise Service Similarity
Tagged Services+ Computed Similarity
Apply Dynamic Clustering if status "ToBeCategorised"
Observe formed Clusters and their members
Generate candidate cluster tagset from member service tag
Identify best cluster tags using Cluster tagging rules
Cluster tagset expansion
DWDS Service Collection

Figure 6.1: Dynamic Service Clustering process

The process of service tagging, similarity computation and service categorisation is performed based on the status of each service entity, as per the change propagation strategy described in section 4.3.2 of chapter 4. This is required to reduce the amount of work required every time a new service is added to the system, after periodic crawler runs. According to this, each service entity is assigned a corresponding status as soon it successfully completes a certain process. For instance, when a newly crawled service S has passed all checks and is added to the repository, it is assigned the status *'NewWSDL'*, which means that it is now ready to be preprocessed. Next, the WSDL of the service S is analysed for tag generation and similarity computation (using the techniques described in section 5.4.1 and 5.4.2). If successful, the status of service S is set to *'MetadataGenerated'*, and the service tags are saved in the repository, after which, the service status is changed to *'ToBeCategorised'*. Thus, clustering is to be performed on all services with this status, using the service pair-wise similarities. The proposed dynamic clustering algorithm described in Section 6.4.1.

6.4.1 Bird Intelligence based Incremental Clustering (BI^2C)

The proposed dynamic clustering algorithm, named BI^2C Algorithm, is inspired by the migratory behaviour of birds in nature. Birds display very complex and diverse grouping rituals, called *flocking*, which differs from species to species. Generally, some species display very rigid patterns and groupings during seasonal migration, while others form complex flocks for safety and self-preservation. During migration, many species of birds follow a 'V' shaped formation, with the leader at the leading edge of the formation. It is quite extraordinary that among hundreds of birds migrating, same species always form a flock very fast and in an organised manner, even in full flight (Freiderici 2009). Even when the individuals have to leave the flock for food or rest, they always return to the exact same flock and resume their position. Figure 6.2 illustrates such bird flocking behaviour during flight and migration.

Figure 6.2: Bird flocking behaviour during migration

An interesting phenomenon that has often captured the interest of bird watchers and researchers alike, is that, birds in formation are always in alignment with each other and maintain the same velocity as their nearest neighbours. This effectively helps them in keep in formation. Researchers proved that birds watch and imitate their nearest neighbours, hence making this an effective local optimization strategy. All birds inherently display this behaviour, thus, helping them achieve a global optimization by maintaining a constant relative speed and alignment. Also, birds can make sudden changes in direction and still maintain flock formation, in the case of danger or direction change, due to this strategy. Thus, birds can almost always maintain their formation and individual position within their species' flock, even if thousands of birds belonging to different species are flying all around them. Hence, the main properties of flocking behaviour can be summarized as -

i. *Homogeneity :* Every bird in the flock follows the same behavioural model.

ii. *Locality :* Only the movements of the nearest flock-mates influence the motion of a particular bird.

iii. *Collision Avoidance :* Each bird actively avoids colliding with its nearest neighbours.

iv. *Velocity matching :* Each bird attempts to match the velocity of its neighbours.

v. *Flock centering :* Each bird attempts to stay close to its nearest neighbours.

The bird flocking behaviour was formally described and mathematically formulated by Reynolds (1987), who used the *flocking model* for computer simulations of groups of flying objects in animation movies. In a novel approach to clustering Web services, we adapt this model and use it to enable domain-specific categorisation of services incrementally and dynamically, to deal with the dynamic nature of the proposed framework. The fundamental concepts and the complete working of the proposed algorithm is discussed in Section 6.4.2.

6.4.2 The BI^2C Algorithm

6.4.2.1 The Basic Flocking Model

In BI^2C, the $\mathcal{DWDS}$ services are modelled as a flock of independent agents that interact with each other in virtual space, to cluster by a bottom up, decentralized and self-organizing method. The flocking model is described by three important properties based on which the behaviour of birds can be emulated as a data clustering strategy. They are -

1. *Alignment* is a behaviour that causes a particular agent to line up with agents that are close by i.e., within a defined radius (helps in encouraging clustering behaviour).

2. *Cohesion* is a behaviour that causes agents to steer towards the "centre of mass", i.e., the average position of the agents within a certain radius. This helps in achieving well-formed, compact clusters (minimising intra-cluster distance).

3. *Separation* is the behaviour that causes an agent to steer away from all of its neighbours, within a defined radius. This helps in maintaining well-separated clusters (maximising inter-cluster distance).

The basic flocking model follows these three properties and defines three different velocities that conform to each property. Hence, each agent *(service)* is influenced and guided by three steering velocities that operate parallely on each individual at every instance of time. These are described below.

1. *Alignment Velocity* ($\overrightarrow{v_{al}}$) - The velocity which enables agents *(services)* to keep moving in accordance to the average heading and velocity of the neighbours, to promote clustering behaviour. Each boid in a flock tries to head in the same direction as the rest of the flock. In every iteration, each agent looks at the direction in which it is travelling in comparison to the directions of all its neighbours (within a defined radius), and realigns itself to match their heading. The velocity vectors of each boid within the S_r are averaged and the resulting vector points in the average direction of the flock, which the boid then tries to head in. It is given by Equation (6.1).

$$\textit{Alignment velocity} \quad \overrightarrow{v_{al}} = \sum_{i}^{n} (\overrightarrow{v_i}) \tag{6.1}$$

2. *Cohesion Velocity* ($\overrightarrow{v_{ch}}$) - The velocity that guides an agent *(service)* to keep moving along with its neighbours. The cohesion component of the algorithm is mainly responsible for the togetherness of a group of similar agents. During every iterations, each agent checks at the position of each other agent to see if it is within a specified sensor range S_r, that is, it checks to see which other agents are close enough to be considered flockmates. The positions of the qualifying neighbours are averaged and the agent steers towards that position. This way, each agent is trying to steer towards the centre of the flock, resulting in them all staying close together. It is given by Equation (6.2).

$$\textit{Cohesion velocity} \quad \overrightarrow{v_{ch}} = \sum_{i}^{n} (p_i - p_c) \tag{6.2}$$

3. *Separation Velocity* ($\overrightarrow{v_{sp}}$) - The velocity that keeps an agent *(service)* from colliding with its neighbours by moving away from them. While in a flock, each agent tries not to run into another in the flock. They try to remain separate by keeping a specified amount of space in between themselves. Each agent checks all the other

agents on the canvas to see if the distance between them is too small, and if so, adds an inversely proportional amount to its velocity in the opposite direction. It is given by Equation (6.3).

$$Separation\ velocity \quad \overrightarrow{v_{sp}} = \sum_{i}^{n} \left(\frac{\overline{\overrightarrow{v_i} + \overrightarrow{v_c}}}{d(n_i, a_c)} \right) \tag{6.3}$$

where,

a_c = agent under current consideration.

$\overrightarrow{v_c}$ = the velocity of the current agent a_c.

p_c = the position of a_c in virtual space.

n_i = i^{th} neighbour of a_c

$\overrightarrow{v_i}$ = the velocity (in this case, similarity) of the i^{th} neighbour, n_i, within the defined locality of the current agent (given by the *attraction* or *repulsion* velocities as computed by equations (6.4) and (6.5)).

p_i = the position of n_i in virtual space.

$d(n_i, a_c)$ = the Euclidean distance between the current agent a_c and the neighbour n_i in virtual space.

6.4.2.2 The Extended Flocking Model

Apart from the above three velocities defined by the Basic Flocking Model (Reynolds 1987), two other velocities *attraction* and *repulsion* are defined. These are based on the computed service similarity values $sim(n_i,\ a_c)$, and are defined to govern the clustering behaviour. These are calculated based on the similarity between every service pair (computed using the technique described in section 5.4.3 of Chapter 5).

4. *Attraction* ($\overrightarrow{v_{sim}}$): keeps two similar agents together and is given by -

$$Attraction\ v_{sim} = \sum_{i}^{n} \{sim(n_i, a_c) \times d(p_i, p_c)\} \tag{6.4}$$

   where, $sim(n_i, a_c)$ is the pair-wise similarity computed using the feature vectors of agents a_c & n_i; and $d(p_i, p_c)$ is the Euclidean distance between p_i and p_c in virtual space.

5. *Repulsion* ($\overrightarrow{v_{dsim}}$)*:* allows dissimilar agents to repel each other to refrain from clustering, and is given by:

$$Repulsion\ velocity\ v_{dsim} = \sum_{i}^{n} \frac{1}{\{sim(n_i, a_c) \times d(p_i, p_c)\}} \tag{6.5}$$

6.4.2.3 Model Constraints

To conform to the *locality* property of the flocking model, a locality perimeter called the *sensor range* (S_r) is defined. It is a variable value and is the measure of the radius around a particular agent, using which a circle is drawn around it. Within this circle, the current agent a_c interacts with the other neighbouring agents and shows clustering behaviour based on the similarity values and computed velocities.

To limit the search for a neighbour to this predefined radius, an efficient optimization technique called *Region Quadtree* were used, which makes BI^2C an efficient local-search and optimization algorithm (discussed in detail in Section 6.4.2.6). Since similarity is also considered, this particular flocking model becomes a *Multi-species Flocking Model*. This means that, even though there are services belonging to multiple domains, only similar services should form a cluster and maintain some degree of separation from the services belonging to other clusters. Hence, a certain threshold value is pref-defined, and only if the similarity between two services is larger than the defined value, they can form a cluster.

6.4.2.4 Flocking Methodology

Each agent *(service)* is governed by the values of the computed velocities of alignment, separation and cohesion; while the direction of movement is governed by the attraction/repulsion velocities. The procedure is carried out as follows:

1. Initially, the agents are placed at random positions on the 'canvas'. The values of *maxIterations*, *sensor range* S_r and *threshold* are defined.

2. During the first iteration, a random agent is chosen as the current agent.

3. The neighbours of a_c within the defined S_r are retrieved by querying the region quadtree.

4. Now, the values of the velocities $\overrightarrow{v_{al}}$, $\overrightarrow{v_{sp}}$ and $\overrightarrow{v_{ch}}$ are computed for the first time, using the velocity of the i^{th} neighbour and the Euclidean distance between current agent a_c and the i^{th} neighbour, within S_r. Each of these velocities work independently, so, they may have positive or negative values.

5. Next, these three velocity vectors are added to the agent's current velocity to obtain its new velocity. Interpreting the velocity as how far the boid moves per iteration, it is simply added to the current position.

6. The direction in which the current agent should move is given by the similarity/dissimilarity between it and the i^{th} neighbour.

- If similarity $\geq$ *threshold*, then current agent is moved towards the i^{th} neighbour by a value equal to the new velocity.
- If similarity $<$ *threshold*, then current agent is moved away from the i^{th} neighbour by a value equal to the new velocity.

7. During subsequent iterations, agents can form clusters by moving towards similar services (or away from dissimilar services to interact with other possibly similar services). The status of each cluster and its members can be observed after the pre-defined number of iterations (maxIterations).

To model this intelligent behaviour in each agent, certain requirements are to be met. Each individual service has to be modelled as an social entity, that interacts with other entities and takes active decisions, regarding its position and movement, during each iteration. The process of defining services as intelligent agents and the framework used to implement BI^C is discussed in Section 6.4.2.5.

6.4.2.5 Web Services as a Multi-agent Framework

To be able to follow the flocking rules, each service has to be modelled as an intelligent entity that is capable of distinct, simple functionalities. To emulate the independent agent whose clustering behaviour can be governed by the rules of the flocking model, we used the Java Agent DEvelopment Framework (JADE) (Bellifemine et al. 1999; Bellifemine et al. 2001). JADE is a multi-agent framework which follows the FIPA (Foundation of Intelligent Physical Agents) rules for implementing agents. This framework can be used to implement a distributed system of agents controlled by a central administrative platform in which each agent can effectively display autonomous quality, pro-activeness and social behaviour. These properties are important in simulating the flocking behaviour and are described below -

1. *Autonomous* - Agents can control their own actions to a certain degree and, sometimes, are also able to take decisions.
2. *Proactive* - Agents do not only react in response to external events (i.e. a remote method call) but they also exhibit a goal-directed behaviour and, whenever required, are able to take initiative.
3. *Social behaviour* - Agents are able to, and need to, interact with other agents in order to accomplish their task and achieve the defined system goal.

To ensure that each service entity in the system, modelled as an agent, follows the rules required for the enactment of the flocking model, two types of agents are created that display the above three properties -

1. *Manager agent* - takes care of initializing the agents, communicating with them, calculating similarities when new agents are added to the dataset, destroying agents when their corresponding services are to be deleted from the repository. The Manager agent holds references to all basic agents created in a container and allows communication between the agents.

2. *Basic agent* - are actual services that are modelled to interact in 2D space, emulating birds in the sky. Each basic agent has a certain position, a velocity and a reference to the Manager agent for communicating with other agents. Each agent can query for all its neighbouring agents through an optimized data structure called the *quadtree*. After obtaining a list of neighbouring agents within its sensor range S_r, each agent calculates the resulting change in velocity (due to similarity or dissimilarity) by applying the flocking rules. Using this, each updates its own velocity & direction of movement.

JADE conforms to the specifications defined by the Foundation of Intelligent Physical Agents (FIPA) (O'Brien et al. 1998). One of the major principles defined by FIPA for agent based systems is asynchronism. According to this principle, agents cannot directly interact with each other. They have to interact through message queues. This principle poses a problem when an agent needs to query other agents within its sensor range only. In order to overcome this problem, we store the changes in the data values of each agent in a NoSQL database so that each agent can query the database to get information about the neighbouring agents within its sensor range. To improve the process of querying for an agent's neighbours only in the vicinity of the current agent, the proposed BI^2C uses certain efficient data structures to optimize the clustering performance. We discuss this optimising mechanism in Section 6.4.2.6.

6.4.2.6 Optimizing the Clustering Process

In BI^2C Algorithm, there is a need to optimise the performance to support local search and optimise the process of finding neighbours within the sensor range. As each agent needs to perform a continuous local search to find agents within its sensor range, the algorithm is of complexity $O(n^2)$. In order to optimize the querying process, we need an efficient data structure to optimise the process. Such a data structure should facilitate quick access to neighbouring nodes for optimal local search.

For this purpose, the quadtree (Finkel et al. 1974), a class of hierarchical data structures that are efficient for large-scale spatial indexing problems is used. Owing to its "Divide and Conquer" strategy, it is most often used to partition a two dimensional space by recursively subdividing it into four quadrants or regions. It is used extensively in the fields of computer graphics and image processing for encoding data to reduce storage requirements. Some common types of quadtrees are – region quadtrees, point quadtrees, and edge quadtrees. We used the *region quadtree* to optimize the querying process during the running of the BI^2C algorithm.

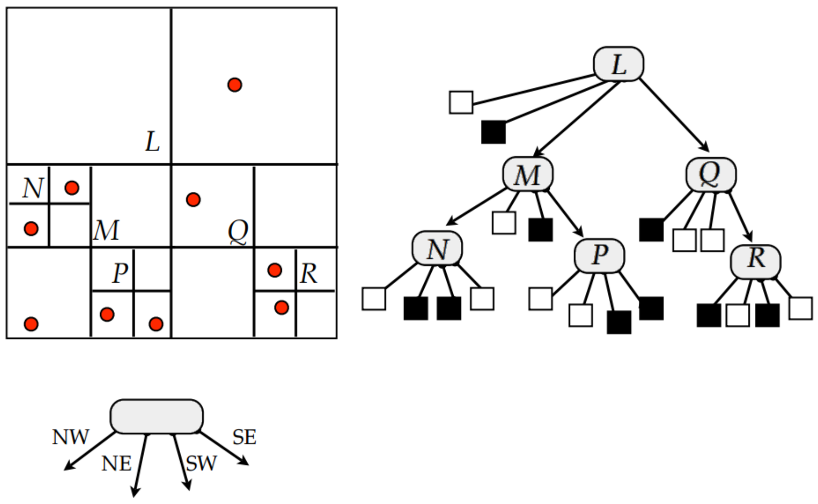


Figure 6.3: Given region with agents and the corresponding quadtree

Region Quadtrees are based on a recursive subdivision of a given region into four equal sized quadrants at each level. The region is recursively decomposed into four rectangular blocks until all blocks are occupied by an agent or are empty as shown in Figure 6.3. Every node of the tree corresponds to a quadrant in the dataset. Each inner (non-leaf) node has four child-nodes where each one represents a quarter of its parent node (referred to using geographic direction: NW, NE, SW, SE) and all quadrants in the same level have the same size. The leaf nodes correspond to the blocks, which do not require further subdivision. Each descendant n_d of a node n represents a quadrant in the plane whose origin is n and which is bounded by the quadrant boundaries of the previous step.

Region quadtrees were used in the BI^2C to manage the agents in 2D space and optimize the process of searching for neighbours within an agent's sensor range S_r. During quadtree construction, each agent is assigned to a leaf node in the quadtree. Then, the quadtree is traversed to get all the agents within the sensor range of the agent under consideration. The similarity between the agent and all the retrieved agents within the sensor range is then computed, so that the flocking model based interaction can commence. This greatly decreases the computation overhead as the total number of computationally intensive similarity comparisons are limited to only those between the retrieved agents in the vicinity of the current agent and not the entire set. The detailed working of the algorithm is explained in Section 6.4.2.7.

6.4.2.7 BI^{2}C– Clustering Phases

The working of the proposed BI^2C algorithm can be viewed as consisting of two separate phases - *Initial Clustering* and *Incremental Cluster Maintenance*. Each is discussed in detail below, with reference to the respective stages.

A. Initial Clustering Phase. In the first phase, all the services in the repository are to be clustered for the first time. Algorithm 6.1 illustrates the initial clustering phase in BI^2C.

Algorithm 6.1 BI^2C Initial Clustering Phase

```
Input:
    Set of services modelled as autonomous agents
    Computed similarity values
    Defined sensor range $S_r$ of each service
    Number of iterations $maxIterations$
Output: Clustered services
    Create agents and deploy in 2D virtual space with random velocities
    for j = 1 $\cdots$ $maxIterations$ do                ▸ maxIterations set to 400 iterations
        for each current agent $a_c$ do
            Retrieve neighbours within $S_r$ by querying the Region Quadtree
            Compute velocities $\overrightarrow{v_{al}}$, $\overrightarrow{v_{sp}}$ and $\overrightarrow{v_{ch}}$ using velocity of the $i^{th}$ neighbour and their
    Euclidean distance
            Add three velocity vectors to get new velocity $\overrightarrow{v_{new}}$    ▸ Current agent must move
    by a value = $\overrightarrow{v_{new}}$
            for each neighbour $n_i$ within sensor range $S_r$ do
                if attraction velocity $v_{sim} \geqslant$ threshold then
                    Move towards $n_i$ by a value = $\overrightarrow{v_{new}}$   ▸ $n_i$ & $a_c$ are similar enough to cluster
                else
                    Move away from $n_i$ by a value = $\overrightarrow{v_{new}}$          ▸ services are dissimilar
                    Update $S_r$
                Interact with other agents as per flocking rules.
    return cluster details
```

As seen in Algorithm 6.1, a set of services are modelled as agents and are added to the virtual space for the first time, after computing each service's pair-wise similarity with every other service in the repository. The agents should follow the defined rules as per the flocking model and their movement is governed by the various velocities as described earlier, so that they can interact with their neighbours within a defined sensor range S_r.

For preventing an exhaustive search of the entire service set to determine neighbours, a small sensor range is defined, so that the search for neighbours can be treated as a local search problem. Agents can then form clusters by moving towards similar services (or away from dissimilar services) when similarity is above a certain threshold value. The status of each cluster and its members can be observed after the pre-defined number of iterations (during experiments, it was seen that the maximum number of iterations, after which there were no more changes in the formed cluster information was about 360, hence, the $maxIterations$ value was set to 400).

B. Incremental Cluster Maintenance Phase. The next phase is triggered on the introduction of a new set of services after a periodic (active or selective) crawler run. The *Incremental Cluster Maintenance* phase is triggered when the *event-count* of *'ToBeCategorised'* events is above some defined number, say 100. This means that 100 new metadata generated WSDLs with the status 'ToBeCategorised' are available in the *Enriched Service Repository*. Hence, only these 100 WSDLs have to clustered incrementally, by either joining existing clusters, or by forming a new cluster, if sufficiently dissimilar from existing services.

Consider that active crawling was performed, and that several new WSDLs have been found, which have been added to WSDL Preprocessor database first and then to the Enriched Service Repository after metadata generation. The pairwise similarity computation with all other services has to be computed now. Here, instead of computing the similarity between every service pair in the repository, a small fraction equivalent to $\frac{1}{10}^{th}$ of the total services is selected, randomly from each existing cluster. The similarity between the each newly added service and those from the selected $\frac{1}{10}^{th}$ services from each cluster is computed.

Next, the new service is also modelled as an agent along with these $\frac{1}{10}^{th}$ services and deployed in 2D space. After enactment of the flocking model, it can join the cluster containing the service with which the maximum attraction velocity (v_{sim}) was obtained (if it is higher than the threshold value of 0.75). If the maximum similarity value is lesser than threshold value, after 400 iterations, then the new service is sufficiently dissimilar, so a new cluster can be formed. Hence, this phase clusters new additions incrementally without having to recompute all the earlier clusters.

During the selective crawling process, the SSC rechecks the status of all the indexed WSDL files in the repository to keep it up-to-date. The crawlers use the links on the *validWSDL-URL* list as feed-URLs and systematically visit each for checking the status of indexed WSDLs (described in detail in section 4.3, Chapter 4). At the end of this process, any indexed services that were unavailable have to be removed from the repository. Hence, the corresponding hashes are located and the services are removed from the repository, at the same time, the cluster member list is also updated. Algorithm 6.2 illustrates this process.

Algorithm 6.2 BI^2C Incremental Cluster Maintenance Phase

```
Input: Clustered set of services
Output: Updated clusters with new services
 1: Find pair-wise similarity with $\frac{1}{10}^{th}$ random services from each existing cluster
 2: Create agent to represent each new service $S_{new}$.
 3: Deploy $S_{new}$ as agent along with $\frac{1}{10}^{th}$ random services from each existing cluster in
    2D virtual space
 4: for i = 1 ··· maxIterations do
 5:     for each new service $S_{new}$ added do
 6:         find list of neighbouring agents using quadtree    ▸ As explained in Section 6.4.2.7
 7:         Compute various velocities w.r.t. each neighbour using Euclidean distance    ▸
    As per basic and extended flocking model
 8:         if $max(\vec{v}_{sim}) \geqslant$ threshold then    ▸ threshold is set to 0.75
 9:             Add $S_{new}$ to the cluster of the neighbour for which $\vec{v}_{sim}$ value was highest
    ▸ incrementally clustered
10:         else
11:             Move $S_{new}$ away by an amount equal to total computed velocity
12: if $max(\vec{v}_{sim}) \leq$ threshold then
13:     Create a new cluster    ▸ $S_{new}$ does not belong to any existing clusters
14: return updated cluster details
```

After the completion of each clustering phase, each cluster is labelled with an unique identifier. The member services of each cluster, as identified by their hash-id are also noted. Next, the tags of the member services have to be processed further to capture enough domain information so that representative cluster tags can be generated. This process is described in Section 6.4.3.

6.4.3 Automatic Cluster Tagging

After the incremental clustering process, the clusters formed represent the functional diversity of the services in the collection. To capture the domain-specific terms of each cluster, additional metadata in the form of cluster tags is required. Given a query, cluster metadata can be used to efficiently eliminate irrelevant domains, thus reducing search space and optimising the process of service discovery. To capture the most appropriate domain-specific terms for generating the tagset of a cluster, the tags of each member service of that cluster are considered as potential tag candidates. However, all such tag terms cannot be used as cluster tags, so to find the relative importance/weight of each, a ranking process is applied to these tag candidates. two factors are considered while performing this ranking -

- How well a tag g represents service s. This value is equal to the *Tf-idf* score obtained for g in s (represented as $sim_{tag}(g, s)$).

$$sim_{tag}(g, s) = Tf\text{-}idf(g, s) \tag{6.6}$$

- How well the service s, whose tag is under consideration, represents its cluster c (denoted as $csim_{service}(s, c)$).

$$csim_{service}(s, c) = \sum_{i=1, s_i \neq s}^{i=|c|} \frac{sim_{wsdl}(s, s_i)}{|c|-1} \tag{6.7}$$

where $|c|$ represents number of services in cluster c.

Then, the level of similarity between a cluster tag g and its cluster c (denoted by $sim_{clusterTag}(g, c)$) is given by -

$$sim_{clusterTag}(g, c) = \sum_{g \in \tau(s), s \in c} (sim_{tag}(g, s) \times csim_{service}(s, c)) \tag{6.8}$$

where $\tau(s)$ is the set of tags for service s.

Using the values obtained for these two factors, all candidate tags are ranked. Based on the resultant ranking, the top-5 tags are chosen as cluster tags. In addition to this, a cluster tag expansion process is performed, by adding synsets provided by WordNet for each tag in the cluster tag set. These together form the weighted feature vector of each cluster, which will be utilized during the query-to-service matching process.

6.5 Experimental Results & Discussion

In this section, the experimental evaluation of the proposed approaches for functional semantics and similarity based incremental clustering is discussed. Firstly, we evaluate the performance of the proposed BI^2C algorithm in comparison to traditional clustering algorithms to verify the speed-up achieved by its incremental nature in Section 6.5.1.

The evaluation of the clustering goodness of the proposed BI^2C is discussed in 6.5.2. After the clusters are formed, the process of tagging these cluster is carried out, and the quality of tagging is discussed in Section 6.5.3. A simple visualization of the incremental clustering process was developed which is presented in Section 6.5.4.

6.5.1 Incremental Clustering Performance

The main advantage of the incremental clustering algorithm over traditional clustering algorithms is the time and cost savings by not having to recompute the clusters after every small dataset change. To evaluate the performance of BI^2C, a traditional algorithm, Hierarchical Agglomerative Clustering (HAC) is also applied on the service collection. We also used two other service datasets, the ASSAM dataset[1], which is a publicly available collection of categorised service descriptions containing 448 manually classified service descriptions. We chose 70% of the services for initial clustering *(round 1)*, then 15%, 10% and 5% for three subsequent clustering rounds *(round 2, round 3 and round 4)*, for testing incremental clustering performance. The clustering performance in terms of time taken and clustering goodness was compared with that of Hierarchical Agglomerative Clustering. Similar experiments were conducted on the OWL-S TC dataset[2], which contains 1076 WSDL files belonging to various domains like food, communication, military, medical etc. Finally, the proposed BI^2C and HAC were also applied to the $\mathcal{DWDS}$ service collection (currently contains 12,231 WSDL files). Again, four different testcases were considered, with number of services equal to 70%, 15%, 10% and 5% of the dataset size.

Table 6.1: Experimental Setup - BI^2C vs. HAC

Test	**Dataset**	**Initial size**	**Number of New Additions***		
		Round 1(70%)	*Round 2(15%)*	*Round 3(10%)*	*Round 4(5%)*
1	ASSAM (448 WSDLs)	314	67	45	22
2	OWL-S TC (1076 WSDLs)	752	162	108	54
3	$\mathcal{DWDS}$ (12231 WSDLs)	8562	1835	1222	612

**to mimic the actual changes in the repository after a periodic active/selective crawl*

The experiments were carried out on a Intel® Core i7™ Quad-Core Workstation with 16GB DDR3 SDRAM and 1TB hard drive. The experimental setup with reference to

[1] Available at http://www.andreas-hess.info/
[2] Available at http://semwebcentral.org/projects/owls-tc/

each round, for the various datasets used, is provided in Table 6.1. The results obtained with reference to clustering time of the two algorithms, HAC and the proposed BI^2C, is summarised in Table 6.2.

As can be seen from Table 6.2, the proposed incremental algorithm was very effective in handling the various datasets with a significant saving in time. For the ASSAM dataset, the HAC required a total of 12.52 minutes for the initial dataset (round 1) and for the reclustering process (after round 2, round 3 and round 4), while the BI^2C algorithm performed incremental clustering in about 5.1 minutes. Similarly, for OWL-S TC dataset, the HAC's total clustering time was 21.79 minutes, while that of BI^2C was 9.71 minutes. Finally, for the $\mathcal{DWDS}$ service collection of 12,231 WSDLs, the hierarchical algorithm failed to complete, while the BI^2C algorithm completed the clustering in about 1 hour, 28 minutes.

Table 6.2: Comparative Cluster-time Evaluation - BI^2C vs. HAC

Test	Method	Clustering Time (in minutes)				Speedup Factor	Speedup (%)
		Round 1	*Round 2*	*Round 3*	*Round 4*		
1	HAC	2.54	3.09	3.36	3.52		
	BI^2C	3.19	1.01	0.56	0.34	**2.69x**	**62.8%**
2	HAC	4.23	5.11	6.02	6.43		
	BI^2C	4.41	2.27	1.58	1.45	**2.24x**	**55.4%**
3	HAC	*failed to complete*	-	-	-		
	BI^2C	58 .39	14.27	9.19	6.21	-	-

Figure 6.4 provides the clustering time performance of HAC and the proposed BI^2C for the first dataset considered, the ASSAM dataset. It can be seen that the HAC algorithm performs well for the initial size of 314 WSDLs considered, and has a clustering time lower than that of BI^2C. This is because, for BI^2C, the initial clustering phase requires repeated interaction with the manager agent for querying the relative positions of the neighbouring agents and for computing the velocities for beginning the clustering process. However, during round 2, 3 and 4, the BI^2C just performs the *incremental cluster maintenance* process, due to which the time taken for clustering the new services is lesser. In contrast, HAC has to restart the clustering process after new services are introduced during round 2-4, hence, the clustering time is dependent on the number of services to be clustered. Similar results were observed for the OWL-S dataset (refer Figure 6.5). For the $\mathcal{DWDS}$ repository, the HAC failed to compute the clusters during the initial phase itself, while the BI^2C algorithm took about an hour's time for initial clustering, completing the incremental clustering of the entire dataset in 88 minutes.

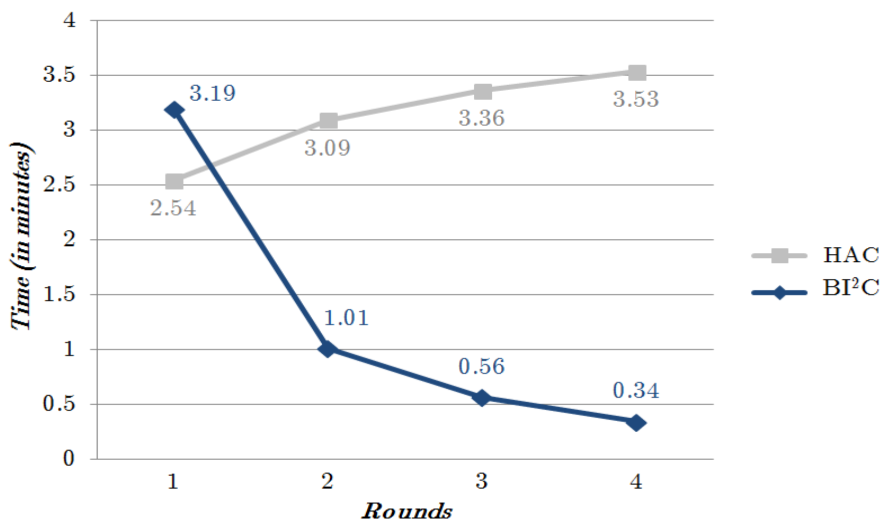


Figure 6.4: BI^2C vs. HAC - Clustering Time for ASSAM dataset

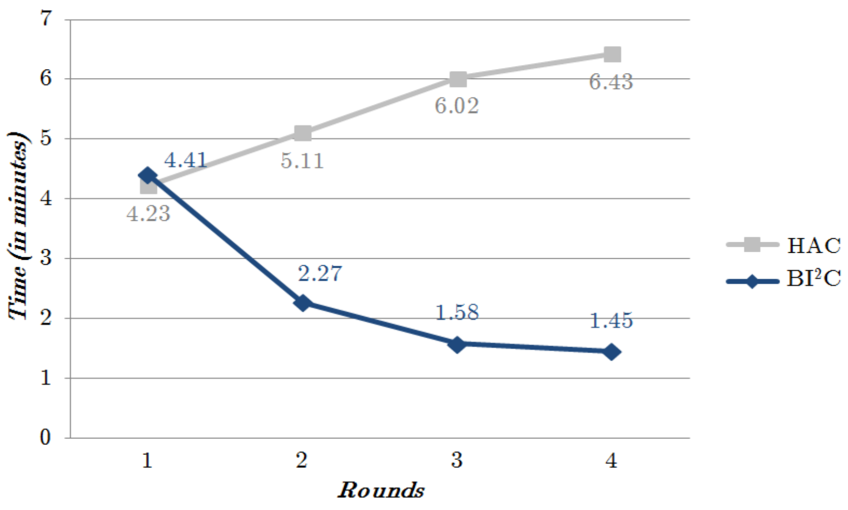


Figure 6.5: BI^2C vs. HAC - Clustering Time for OWL-S TC dataset

Based on these observations, it can be said that there was a significant improvement in clustering time for each dataset, when the BI^2C was used. The average speedup achieved by BI^2C when compared to the HAC approach was calculated as per equation (6.9) and (6.10).

$$Speed\text{-}up\ Factor\ SF = \frac{t_{HAC}}{t_{BI^2C}} \tag{6.9}$$

$$\%Speed\text{-}up = (1 - \frac{1}{SF}) \times 100 \tag{6.10}$$

where, t_{HAC} is the total clustering time required by HAC to achieve clustering for the initial size considered, and rounds 2, 3 and 4. Similarly, t_{BI^2C} represents the total clustering time for the proposed algorithm and SF is the computed speed-up factor.

As per the clustering time comparison, BI^2C was 2.69 times faster than HAC for the ASSAM dataset, while for the OWL-S TC dataset, it was 2.24 times faster. It was not

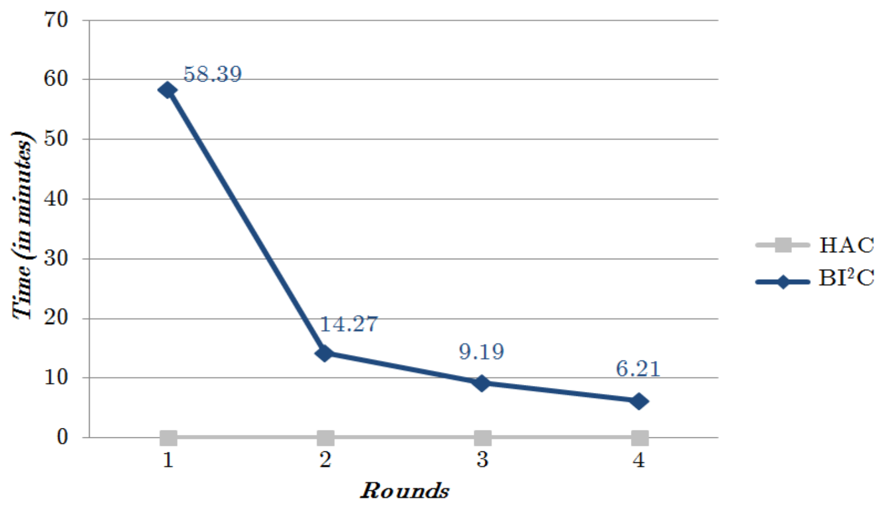


Figure 6.6: BI^2C vs. HAC - Clustering Time for services in $\mathcal{DWDS}$ Repository

possible to compare the two for the $\mathcal{DWDS}$ service collection as the HAC failed to cluster the large collection completely. Hence, considering the first two testcases, the proposed BI^2C algorithm achieved an average speed-up of about 57.35% over the HAC approach.

6.5.2 Goodness of Clustering

The validations performed for assessing the clustering goodness of the proposed incremental algorithm is presented in this section. Table 6.3 summarises the details about the number of clusters formed for the proposed BI^2C and the HAC approaches, for the considered testcases (as shown in Table 6.1). We check the goodness of the clustering using standard cluster validation measures, which is discussed below.

Table 6.3: Number of clusters formed per dataset - BI^2C vs. HAC

Case	Algorithm	Number of clusters formed			
		Round 1	*Round 2*	*Round 2*	*Round 4*
1	HAC	37	39	39	39
	BI^2C	36	38	39	39
2	HAC	48	50	54	54
	BI^2C	51	54	56	56
3	HAC	*failed to complete*	-	-	-
	BI^2C	321	334	339	343

In general, clustering validation can be categorised into two classes, external clustering validation and internal clustering validation. External clustering validation measures use external information not present in the data for performing validation, while internal validation measures rely on the data information itself to measure the goodness of the clustering (Liu, Li, et al. 2010). As such external information for the service description data considered is not available, we used two internal validation methods - *Jaccard's Measure* and *Davies Bouldin Index*.

Jaccard's Measure. In general, a good clustering method should construct high quality clusters with low inter-cluster similarity. The Jaccard's Measure (JM) (Jaccard 1912; Real et al. 1996), also known as the Jaccard Similarity Co-efficient, is a statistic used for comparing the similarity and diversity of sample sets. Hence, it can be used to observe the similarity or degree of overlap between generated clusters. Since the proposed algorithm uses natural language tags that are generated from the WSDL files, the resultant clusters may have some degree of overlap and hence the Jaccard's Measure is used to evaluate cluster overlap. JM between any two clusters C_i and C_j is calculated using Equation (6.11).

$$JM(C_i, C_j) = \frac{|C_i \cap C_j|}{|C_i \cup C_j|} \tag{6.11}$$

where, $|C_i \cap C_j|$ is the number of common services in the two clusters C_i and C_j. $|C_i \cup C_j|$ gives the total number of services in both clusters C_i and C_j. If $JM(C_i, C_j)$ is equal to 1, it indicates that the clusters C_i and C_j are exactly similar while a value of 0 indicates that the clusters are totally different. Therefore, for good clustering, the computed JM should be low, as it indicates higher inter-cluster distance and low level of overlapping.

Davies Bouldin Index. The Davies and Bouldin index (Davies et al. 1979), commonly called DB-Index, is a well-known cluster validity measure, defined as the ratio of the sum of within-cluster scatter to between-cluster separation. DB-Index is computed using the Equation (6.12) as below.

$$DB\text{-}Index(C_i, C_j) = \frac{1}{k}\sum_{i=1}^{k} max_{i \neq j}\left\{\frac{S_k(C_i) + S_k(C_j)}{S(C_i, C_j)}\right\} \tag{6.12}$$

where k is the number of clusters, $S_k(C_i)$ is the average distance of all services in a cluster to their cluster centre, and $S(C_i, C_j)$ is the distance between two different cluster centres C_i and C_j.

To compute DB-Index, the information about clusters formed is collected. Then, the average distance of all services in a particular cluster to their cluster centre is calculated.

The, for each cluster, the DB index with other clusters is calculated using equation 6.12, and the largest DB-Index for each cluster is determined. Finally, the average DB-Index for the clustering algorithm is obtained by equation 6.13.

$$DB\text{-}Index_{avg} = \frac{\sum_{i=1}^{k} DB\text{-}Index_{max}}{Number\ of\ clusters\ k} \tag{6.13}$$

In general, a clustering algorithm that produces a collection of clusters with the smallest DB–Index is considered the best algorithm. This is because, the computed ratio will be small only if the clusters are compact and far from each other, thus indicating a well-defined clustering. Consequently, the average DB-Index value was considered for both HAC and BI^2C.

Evaluation. The average value of the computed Jaccard Measure and the least Davies Bouldin Index obtained for both HAC and BI^2C, for the considered datasets, are tabulated in Table 6.4. After the clusters were generated, Jaccard's Measure value between the various cluster pairs for all clusters formed was computed and JM_{avg} was found for each dataset. The JM_{avg} was found to be nearly the same for HAC and BI^2C for both the ASSAM and the OWL-S TC dataset, indicating similar cluster goodness performance in case of small datasets. In the case of a large dataset like that of the $\mathcal{DWDS}$ repository, HAC failed to cluster at all, while JM_{avg} for BI^2C was about 0.219. In view of these reasonably low value of Jaccard's Measure, it can be concluded that the quality of clusters generated by BI^2C was satisfactory.

Table 6.4: BI^2C vs. HAC - Clustering Goodness Evaluation

Dataset	**Algorithm**	**Total Clusters** k	**Jaccard Measure** JM_{avg}	**DB Index** $DB\text{-}Index_{avg}$
ASSAM	HAC	39	0.169	1.26
	BI^2C	39	0.171	1.32
OWL-S TC	HAC	54	0.198	1.47
	BI^2C	56	0.201	1.49
$\mathcal{DWDS}$	HAC	-	-	-
	BI^2C	343	0.219	1.95

When least $DB\text{-}Index_{avg}$ values were considered, HAC performed slightly better than BI^2C for both ASSAM and OWL-S TC datasets. However, HAC fails for the $\mathcal{DWDS}$

dataset, while the $DB\text{-}Index_{avg}$ value for BI^2C was 1.95. This indicates good inter-cluster separation, based on which we conclude that the clustering goodness of BI^2C is satisfactory.

6.5.3 Evaluating the Cluster Tagging Process

To evaluate the quality of the automatic tagging of service clusters, we present the tagging results for the ASSAM dataset used earlier. It consists of 448 service descriptions categorised by a human into 25 broad main categories, each subdivided into several subcategories. For example, *mail* services were categorised into 'Mail' category and 'Mail' is a type of 'Communication', so the dataset is hierarchically organised.

Table 6.5: Some generated classes and their corresponding class tags

Cluster No.	**Service Domain**	**Some member services**	**Cluster Tags**
1 (42*)	conversion	ConvertAcceleration, ConvertDensity, C-Temperature, HtmlToxml	*convert, unit, measure, translate, metric*
2 (12*)	weather	FastWeather, WeatherByZipCode, ForecastByICAO, weatherWarningsByState	*weather, zipcode, forecast, station, country*
3 (5*)	flight	AdvanceFlightService, AeroflotServiceSchedule, AirportInformationWebService	*flight, airport, status, schedule, city*
4 (11*)	currency	CurrencyConverter, EuroService, CurrencyExchangeService	*currency, convert, exchange, rate, country*
5 (22*)	news	DotnetDailyFact, eSynapsFeed, LiveNewsService, SlashDotHeadlinesProvider	*news, article, live, daily, feed*
6 (16*)	email	EmailValidate, EmailVerifier, ISpamCheck, HTMLEmail	*email, address, domain, validate, verify*

** Number of services in cluster generated*

Table 6.5 shows the tags generated for some generated clusters after the incremental clustering process for the ASSAM dataset. The quality of tags generated for both the services and the clusters was good after conducting a manual human validation. It can be seen that the term weighting applied to the tags of the member services of each class resulted in good quality cluster tags. The system generated meaningful tags that effectively represented the functionalities of the individual Web services and of the generated clusters.

6.5.4 Visualizing the Clustering Process

A simple visualisation was generated to illustrate the working of the BI^2C algorithm. A small sample dataset consisting of 25 services was considered for the visualisation and the interaction between the services was visualised in a 2D bounded space. For the purpose of visualization, we used the Visualizer class of the JADE framework, where an object of Visualizer class is run as a separate thread along with the JADE framework for visualizing the data and the behaviour of the agents. The Visualizer contains a reference to the manager agent, through which it can obtain data about the agents. It generates a simple two dimensional GUI to display the visualization, through which the user can interact with the visualization. A paint function queries for the current position of all the agents through the manager and draws them on the panel. This is an asynchronous thread that is unrelated to the JADE framework, which enables it to act as the *View* part of a MVC framework.

The visualization consists of a canvas referred to as the *sky*, where each of the services, represented by agents, are visualized as discs. A special function queries the database for the computed similarity values between the services for rendering the positions of each of the agents, after which their movement is controlled by the various computed velocities. A sensor range S_r (visualized as a red circle) is defined, within which the agents start interacting with their neighbours. Based on this interaction, the agents can either attract (or repel) based on their similarity values to begin the clustering process. The blue lines represent similarity and yellow lines represent dissimilarity. Figures 6.7, 6.8, 6.9, 6.10 and 6.11 depict the initial and various in-progress views of the clustering process' visualisation.

As seen from Figure 6.7, the services are visualized as agents and are placed at random positions in the 2D space. The manager agent queries the database for the similarity values between the different services using which the agents start interacting with each other. In Figure 6.8, the sensor range (defined as 0.2 currently) is visualised, within which the agents interact and begin the clustering process based on the computed similarity values. The attraction velocity is depicted as the blue line and the repulsion velocity is the yellow line, as can be seen in Figure 6.9. Figure 6.9 also shows a snapshot of the cluster formation process, where a few clusters have started forming.

To demonstrate the incremental clustering process, two new services are introduced onto the sky, which is shown in Figure 6.10. Also, the representative tagset of each service

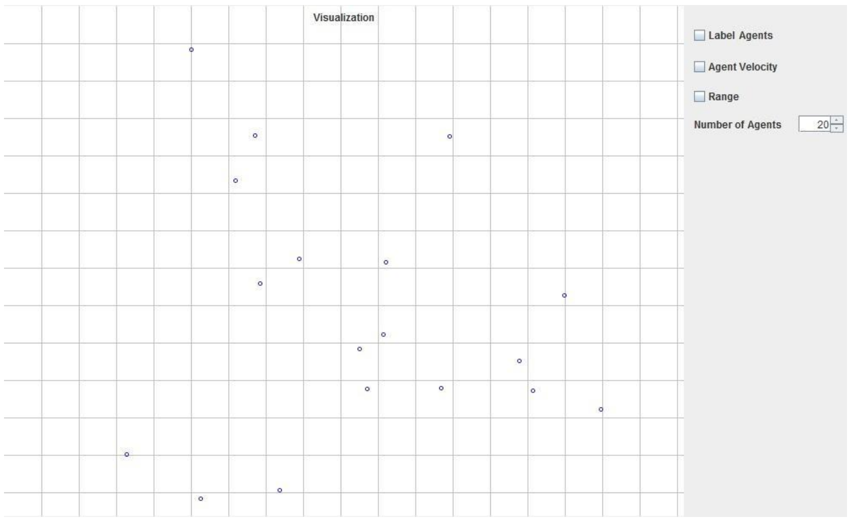


Figure 6.7: Services visualised as agents on 2D canvas

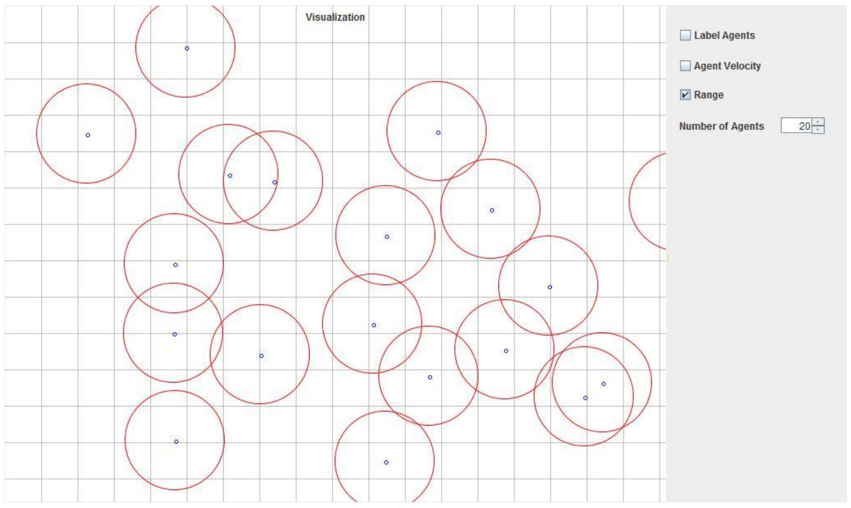


Figure 6.8: Sensor range of visualised agents

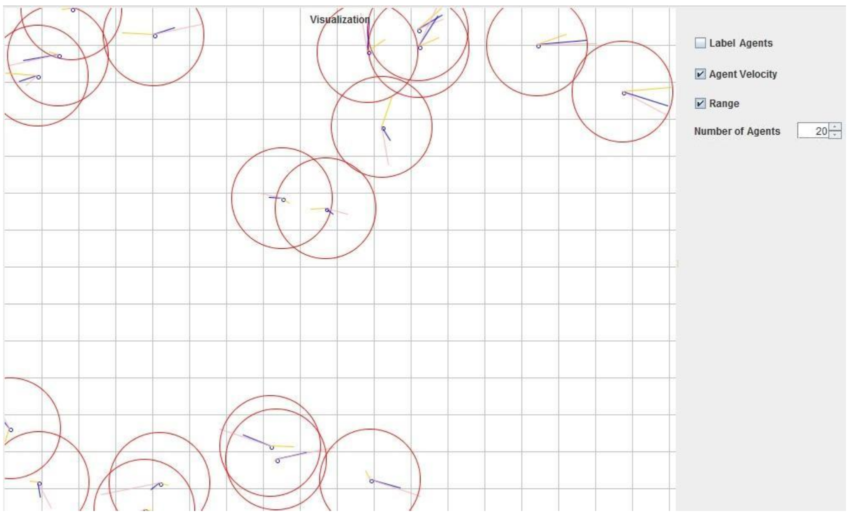


Figure 6.9: A view of cluster formation at some iteration

is also used as a label (the first tag of the tagset is only shown), using which it is easy to identify some formed clusters. Figure 6.11 shows the generated clusters at the end of the predefined number of iterations. It can be seen that the newly introduced services have clustered with existing clusters based on their similarity values. Already formed clusters also may have changed based on the new services interaction. As depicted in Figure 6.11, all services were effectively managed and a satisfactory domain-specific clustering was achieved.

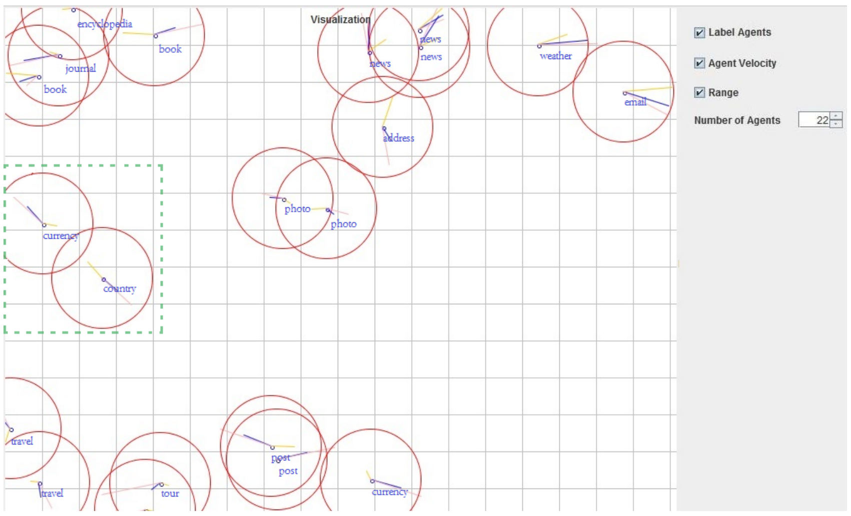


Figure 6.10: Two new services are introduced, to imitate repository changes after active crawling

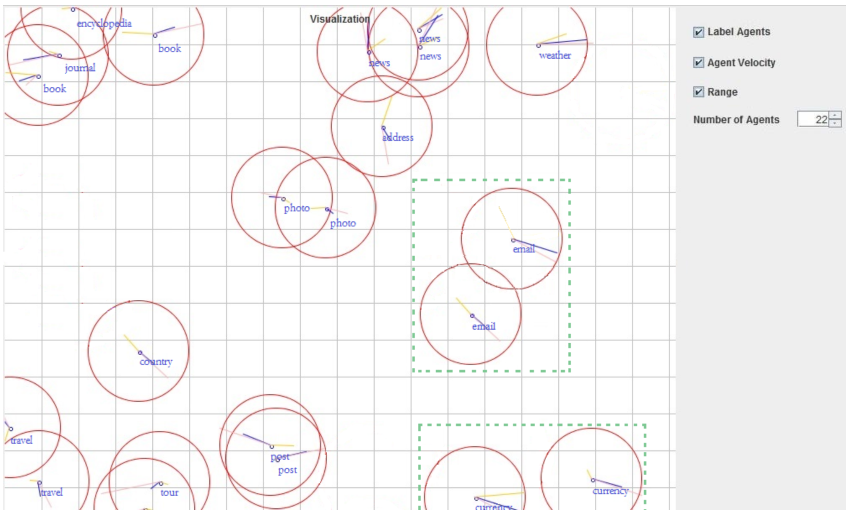


Figure 6.11: Incremental clustering process for the newly added services

6.6 Summary

In this chapter, a novel clustering algorithm called Bird Intelligence based Incremental Clustering (BI^2C) Algorithm was discussed. The BI^2C Algorithm is intended to effectively cluster the constantly changing service collection in the $\mathcal{DWDS}$ framework, as a result of the repeated runs of the service crawler. The proposed clustering algorithm uses the computed semantic similarity between services to perform incremental clustering of the service dataset, thus overcoming the limitation of traditional clustering algorithms. When compared to Hierarchical Agglomerative Clustering, the proposed BI^2C Algorithm achieved an average speed-up of over 57.35%, during experimental evaluation using the $\mathcal{DWDS}$ service collection and two other standard datasets. Two internal cluster validity measures, Jaccard Measure and Davies Bouldin Index were used for evaluating the goodness of the clustering of the BI^2C algorithm, which was found to be good. The quality of the tags generated for each cluster generated by BI^2C was subjected to human validation and were found to be meaningful. These cluster tags would be used during the process of Web service discovery, to eliminate irrelevant clusters for a given query, so that search space reduction can be achieved.

Publications

(based on work presented in this chapter)

1. Sowmya Kamath S and Ananthanarayana V.S, *"A Bio-inspired, Incremental Clustering Algorithm for Semantics based Web Service Discovery"*, International Journal on Reasoning-based Intelligent Systems (IJRIS), Inderscience Publishers, Volume 7, Issue 3-4, 2015. ISSN: 1755-0564 (Scopus and EI Indexed)

   Status: Online.

PART IV

Context-aware Web Service Discovery

Chapter 7

Semanticising the User Query

7.1 Introduction

As discussed in the previous chapters, the proposed techniques for automatic service metadata generation and dynamic categorisation are intended for extending domain-specific service discovery over the $\mathcal{DWDS}$ service collection. To support semantics based Web service discovery in $\mathcal{DWDS}$, all services and clusters in the large collection are indexed using their representative tags. Hence, a query can be served by performing a query-service matching. However, such a matching is highly dependent on the exact terms used by the user while querying, due to which potentially relevant services may not be matched.

This problem arises when information is retrieved by literally matching terms in documents with those of the query (syntactic matching). Since there are usually many ways to express a given concept (synonymy), the literal terms in a user's query may not match those of a relevant document (e.g. *"convert"* -> *"change"*, *"transform"*, *"translate"* etc). Most words also have multiple meanings, called polysemy (e.g. the word *chip* is polysemic as it can be used in multiple contexts, as in "*getChipDetails*" in Silicon chip specification *vs.* "*getChipPrice*" used in online casino services.). So, terms in a user's query may often be literally matched to those contained in documents, which may result in imprecise results. This problem is further compounded by the occurrence of homonyms (e.g., "*tire*" as in a car tyre *vs.* "*tire*" after exercising), hypernyms (e.g., *"vehicle"* is a hypernym of *"car"*) and hyponyms (e.g., *"car"* is a hyponym of *"vehicle"*), which deteriorates the result quality further, if not specifically identified. Hence, due to the diversity with which users can express their requests and service providers can describe their services, simple syntactic matching will be completely inadequate.

To address these problems and to enhance the Web service discovery process, an intelligent mechanism for understanding user context and requirements is crucial. As the user query is unstructured, i.e., in natural language, techniques for natural language processing (NLP) need to be employed, for effectively capturing user context. The resultant semantic query is well-structured, which can be much easily processed and executed by the system, thus enabling it to produce best matching results for a given user query.

In this chapter, we discuss the techniques proposed for understanding the user requirements using semantics and natural language processing techniques. Our main contributions, as reported in this chapter are listed below:

- Incorporating techniques for semantics based query analysis in $\mathcal{DWDS}$ framework to capture user requirements.
- Automatic identification and processing of simple and complex queries for relevant Web service discovery.
- Demonstrating that the semanticised query improves the performance during Web service discovery process.

The rest of this chapter is organised as follows. Section 7.2 describes the defined problem addressed in this chapter. In Section 7.3, we discuss the methodology of semantic analysis of user queries for capturing the correct context and the methods used for identifying simple and complex queries for enabling simple and composite service discovery. Section 7.4 presents the experimental results on the performance of the proposed techniques in comparison with natural language keyword based matching approach. Section 7.5 summarizes the work presented and the contributions of the work presented in this chapter.

7.2 Problem Statement

The problem that is addressed in this chapter is defined here:

> *Given an unstructured user query for discovery Web services, to design semantics based query analysis techniques for capturing user context and to identify simple & complex queries.*

We propose a solution that aims to incorporate a natural language querying interface to the proposed $\mathcal{DWDS}$ framework, to maximise ease-of-use. The developed techniques should be able to cope with the diversity in natural language terms used by a user. It must also be able to identify specific situations where simple service discovery is to be conducted or a composite service discovery process is required, for a given user query. In Section 7.3, we discuss the proposed methodology and describe the techniques developed for processing and semantic analysis of query.

7.3 Proposed Methodology

The detailed methodology followed for identifying the type of service discovery required (simple or composite), and for efficiently capturing user requirements, when specified in unstructured natural language is described in this section. Figure 7.1 depicts the overall view of the complete service discovery process for a given natural language query. In Section 7.3.1, we describe the working of the Query Analysis Engine in detail, followed by the Service Matching process in Section 7.3.2.

7.3.1 Query Analysis Engine

The process of query processing and generating a structured query is performed by the Query analysis Engine is shown in Figure 7.1. When a natural language query is submitted to the system, the processes depicted abstractly in Figure 7.1, and in detail in Figure 7.2 are performed in order. Each of these processes is described below.

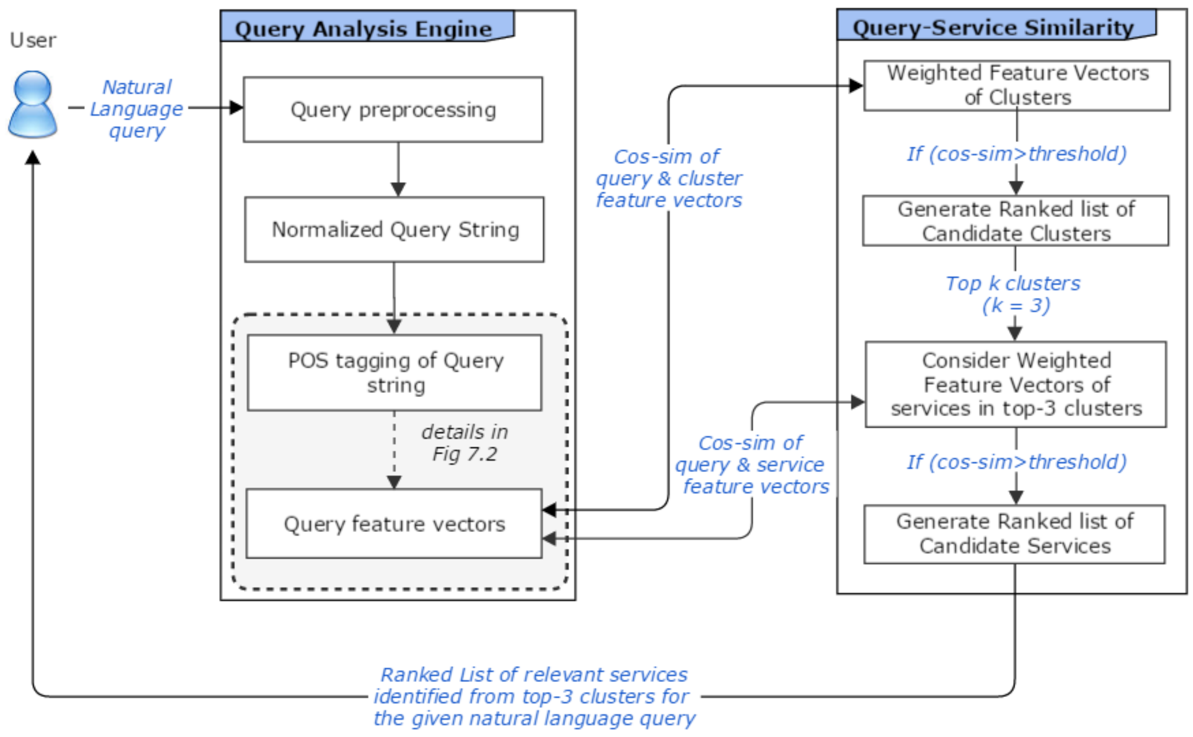

User
Natural Language query
Query Analysis Engine
Query preprocessing
Normalized Query String
POS tagging of Query string
details in Fig 7.2
Query feature vectors
Cos-sim of query & cluster feature vectors
Cos-sim of query & service feature vectors
Query-Service Similarity
Weighted Feature Vectors of Clusters
If (cos-sim>threshold)
Generate Ranked list of Candidate Clusters
Top k clusters (k = 3)
Consider Weighted Feature Vectors of services in top-3 clusters
If (cos-sim>threshold)
Generate Ranked list of Candidate Services
Ranked List of relevant services identified from top-3 clusters for the given natural language query

Figure 7.1: Web Service Discovery - Overall View

1. Query Preprocessing. The first step in handling the query is to perform certain cleaning tasks to preprocess the query. Extra white spaces introduced accidentally by the user while entering the query is trimmed down to one and the query string is subjected to a normalization process to take care of other irregularities.

2. Normalized Query String. After preprocessing, the resultant query string is subjected to a process of normalization. Firstly, the occurrence of the short form '&' is mapped to its English language equivalent of 'and'. After this, all other special characters are removed. Another common problem that occurs during natural language querying, is the varied forms in which users can give input for the same document. For example, *'NITK'* may be typed as *'N.I.T.K'*, or *'Nitk'* or *'nitk'*. If not addressed, each of these will be considered as different terms. Hence, a process of normalization is performed and, finally all words are converted to lower-case, for a uniform representation. The next series of tasks performed on the normalized query are illustrated in Figure 7.2.

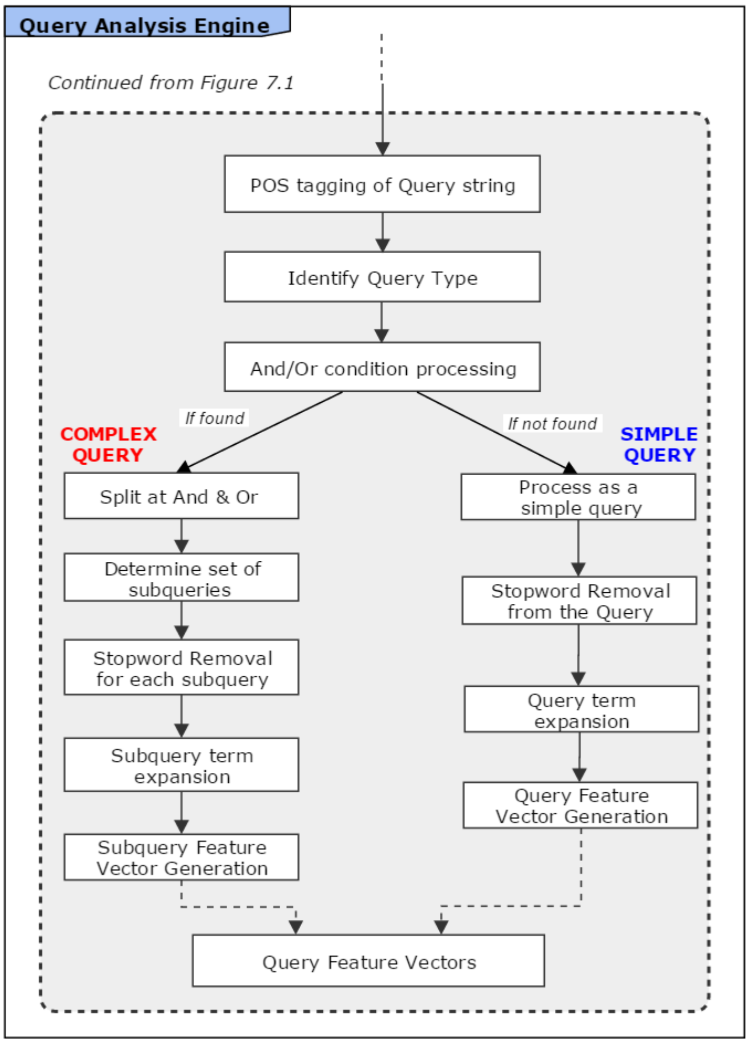

Query Analysis Engine
Continued from Figure 7.1
POS tagging of Query string
Identify Query Type
And/Or condition processing
If found
If not found
COMPLEX QUERY
SIMPLE QUERY
Split at And & Or
Determine set of subqueries
Stopword Removal for each subquery
Subquery term expansion
Subquery Feature Vector Generation
Process as a simple query
Stopword Removal from the Query
Query term expansion
Query Feature Vector Generation
Query Feature Vectors

Figure 7.2: Query Analysis Engine - Detailed View

3. POS Tagging of Query String. For the normalized query, the next task is to determine its type - *simple* or *complex*. Depending on the type of query, the method of processing further differs. Firstly, the normalized query is taken as a string and is tagged using a Part of Speech (POS) tagger. We used the Stanford POS Tagger (Toutanova et al. 2000), which follows the Penn Treebank tagset (Miltsakaki et al. 2004). After POS tagging, the query is represented as a string of *tokens*.

4. Identify Query Type. In the POS tagged query, we specifically look for all terms tagged as */CC*, which represent the *coordinating conjunctions*, as per Penn Treebank terminology. In English, coordinating conjunctions are those parts of speech that are used in a sentence to connect words, phrases & clauses. Seven such coordinating conjunctions exist – *'and', 'or', 'but', 'for', 'nor', 'yet'* and *'so'*. Based on these, the query type can be easily determined - their existence indicates a complex query, while their absence means the query is simple.

5. And/or condition processing. At present, only the first two coordinating conjunctions, *'and'* & *'or'*, have been considered. This means that a query is simple, if no 'and'/'or' are present. If the *'and'/'or'* coordinating conjunctions are found, then the query is treated as a complex query and has to be processed accordingly, as below.

- ***Complex Query Processing***
  - (i) *Splitting at 'And/Or':* In the case one or more *'and'/'or'* are found, the query is to be considered a complex query. To process it further, it is first split at each word tagged /CC (i.e., at each coordinating conjunction *'and', 'or'*. The resultant subqueries are each processed as an independent query, which helps in determining the candidate 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
  - (ii) *Determining set of Subqueries:* For the given example, the input query is composed of three subqueries, and the user desires to find services for hotel *or* resort booking 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, then the results are put together as per the logical representation of $((SQ1 \vee SQ2) \wedge SQ3)$.
  - (iii) *Stopword Removal:* Stopword removal focuses on eliminating those words that add little or no meaning to the search. This is helpful as the search

space can be reduced. In general, to determine the stop words, the stopword list provided by Python's NLTK was adapted, to exclude any coordinating conjunctions (if they exist).

(iv) *Subquery Expansion:* Each subquery is processed independently and parallely. To address synonymy, WordNet synsets of each final subquery term are retrieved to capture the synonyms and are added to the subquery feature vector. Hypernym/homonym relationships can also be captured through the WordNet taxonomy, as it is organised as a hierarchy. To address polysemy, it is important to capture the correct *sense* in which each word is used. To capture word sense correctly, the JCN Similarity algorithm (Jiang et al. 1997a) was used which captures the semantic relatedness between word senses in a phrase as per the method proposed by Jiang et al. (1997b). The algorithm returns a score denoting how similar the word senses of the two input synsets are. This score is used to choose the most relevant word sense for each subquery term, in context with the rest of the subquery, i.e., the one with the highest score is chosen.

(v) *Subquery Feature Vector Generation:* For each subquery, the terms, their synsets and the identified senses together form its feature vector $\vec{sq_t}$. This represents each subquery in the vector space similar to the service documents, which allows vector operations like dot product and cosine similarity to be performed to determine level of similarity between services and query.

- ***Simple Query Processing***

(i) *Processing as simple query:* As no coordinating conjunctions are found, each of the POS tagged query terms are considered as potential candidates for query vector generation.

(ii) *Stopword removal:* The stopword removal process is carried out similar to how it is done for each subquery, in the case of a complex query.

(iii) *Query Expansion:* The process of query term expansion is carried out as explained earlier, by identifying the WordNet synsets and also the word senses as returned by JCN algorithm.

(iv) *Query Feature Vector Generation:* Finally, the query vector is generated, using which further matchmaking with service cluster vectors is performed.

7.3.1.1 A Sample Query Resolution Scenario

The steps followed during query resolution for a given natural language query are described in detail below -

1. Initially, the submitted natural language query phrase is taken as a string. After case conversion to lowercase, removal of special characters and boundary detection, the resultant query words are tokenized. For example, consider the sample query, *"service to find rainfall for a zipcode"*. After tokenization, it results in the tokens - *'service', 'to', 'find', 'the', 'rainfall', 'for', 'a'* and *'zipcode'*.

2. The tokenized query is POS tagged used the NLTK POS tagger. This process deals with identifying the part of speech information of a particular query term in the context of the query. This results in -

   service/NN to/TO find/VB rainfall/NN for/IN a/DET zipcode/NN

   This means that - the terms, {*'service', 'rainfall', 'zipcode'*} are nouns, {*'find'*} is a verb and {*'to', 'in'*} are prepositions and {*'a'*} is a determiner. This information is used later, in the process of Word Sense Disambiguation.

3. Stopwords like *'to', 'for', 'a'* are removed. The function words, *'find', 'service'* are also discarded. Thus,

   Service Request (SR) = {*'rainfall', 'zipcode'*}

4. Next, each of the remaining SR terms are 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 for discovering the most appropriate web service satisfying the requested functionality. 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 SR terms as its leaf nodes. In the given example, this results in *'weather'* and *'postal address'*. These are added to the service request SR. Now,

   SR = {*'rainfall', 'zipcode', 'precipitation', 'postal address'*}

5. Before matching these terms to the service tagset, the WordNet synsets (synonyms) of each of the SR terms are extracted. Now, the problem is that each of these terms may have multiple word senses, based on possible English language usage. For

example, WordNet says '*zipcode*' is a noun, which has the synsets {*'zip', 'postal code', 'postcode'*}. The word '*zip*' may be used in multiple contexts (as noun, it means '*zipper*'; as verb, it means '*to run around fast*'; totally 7 different senses are identified by WordNet). Since *'zipcode'* was POS tagged as noun, its verb sense can be neglected. However, even the noun usage of *'zip'* has four different identified sense usages in English language (e.g., *"All their efforts were reduced to zip"*, *"the child is full of zip (energy)"*, *"postal code of a place"* and *"a zip for a pair of trousers"*).

To identify the correct sense in the context of the service request SR, the Word Sense Disambiguation Algorithm, *JCN Similarity* was used. 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 the equation -

$$Score = \frac{1}{(IC(s_1) + IC(s_2) - 2 * IC(lcs))} \tag{7.1}$$

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.

SR = {*('rainfall', 'rain'), ('zipcode', 'postal code', 'postcode'), ('precipitation', 'weather condition', 'atmospheric condition'), ('postal address')*}

6. This final SR *(i.e., query terms, their synsets and the identified senses together)* is used as the query feature vector $\vec{q_t}$. This represents the query in the vector space in a way similar to the service documents, which allows vector operations like dot product and cosine similarity to be performed to determine level of similarity between services and query.

After the query resolution, the unstructured natural language query is represented in a structured format as a semantic feature vector. At the same time, the services in the $\mathcal{DWDS}$ repository and also, the service clusters are represented by their weighted feature vectors, after metadata generation and service categorisation. Hence, all entities, i.e., the *query*, the *service clusters* and the *services* are represented in a common format, allowing effective matchmaking. The process of retrieving relevant services is described in Section 7.3.2.

7.3.2 Query-Service Similarity

For retrieving relevant services for a given query, the feature vectors of the query, the clusters and the services are used. Using their respective feature vectors, each of the entities are represented in common space called the vector space as per the Vector Space Model (VSM) (Salton et al. 1975). The VSM is a document space consisting of N documents, each identified by one or more weighted *(using techniques like Tf-idf)* or unweighted *(if term exists, value is 1; else value is 0)* index terms T. If z such index terms are present, then each document in the document space is represented by a z-dimensional vector, $D_n = (T_{1,z}, T_{2,z}, \ldots, T_{n,z})$, where $T_{i,z}$ represents weight of the i^{th} term in document D_n.

In $\mathcal{DWDS}$, the services are represented by their tagset (words), clusters are represented by their tags (words) and the query is also represented by its vector composed of words, hence each dimension in VSM is represented by a word vector, as shown below -

Cluster $c_j = (w_{1,j}, w_{2,j}, \ldots, w_{n,j})$

Service $s_k = (w_{1,k}, w_{2,k}, \ldots, w_{n,k})$

Query $q_t = (w_{1,t}, w_{2,t}, \ldots, w_{n,t})$

where, each term corresponds to the weight of a word w_i in that document (either cluster, service or query).

Using the weighted vector representation of clusters, services and the query in VSM, level of matching between a given query and possibly relevant services can be determined by using a similarity measure that computes the inner product of the two vectors, or alternatively, an inverse function of the angle between the corresponding vector pairs. This similarity measure is called *Cosine Similarity* ($\cos\theta$). When the terms assignment for a cluster/service vector and the query vector is similar, the angle θ approaches 0, due to which cosine similarity value is closer to 1, indicating high relevance to the query. Similarly, as the dissimilarity between a cluster/service and query vector increases, the θ between their vector representations increases, due to which cosine similarity value is nearer to zero, thus indicating that the cluster/service is irrelevant to the query. In this way, the computed cosine similarity values can be used to generate a ranked list of clusters/services for a given query.

Cosine similarity values between the query and a cluster/service is computed using the equation (7.2) in the case of clusters and equation (7.3) in the case of services.

$$\cos\theta_{c_j,q_t} = \frac{\vec{c_j}\cdot\vec{q_t}}{\|\vec{c_j}\|\|\vec{q_t}\|} = \frac{\sum_{i=1}^{n}(w_{i,j}\ .\ w_{i,q_t})}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2}\cdot\sqrt{\sum_{i=1}^{n} w_{i,q_t}^2}} \tag{7.2}$$

$$\cos\theta_{s_k,q_t} = \frac{\vec{s_k}\cdot\vec{q_t}}{\|\vec{s_k}\|\|\vec{q_t}\|} = \frac{\sum_{i=1}^{n}(w_{i,k}\ .\ w_{i,q_t})}{\sqrt{\sum_{i=1}^{n} w_{i,k}^2}\cdot\sqrt{\sum_{i=1}^{n} w_{i,q_t}^2}} \tag{7.3}$$

where, c_j represents a service cluster j, s_k represents a service k, q_t represents the terms in the query and w represents relative weight of a term with respect to s_k and q_t.

To make sure that the vector dot product is performed properly in case of unequal length of feature vectors, the cluster feature vectors are either padded or truncated temporarily, depending upon the length of the query vector. For example, when the dot-products of a service s_k and some query q_i is to be computed, the dot product is performed for the full length of the query vector and no further (i.e. if the service vector is longer, it is truncated). However, if service vector is shorter than the query vector, then the service vector is padded with additional zeros to normalise the length of the two vectors. After the dot product computation, the cosine similarity values are computed. Using the computed cosine similarity values, a list of all services matching the user requirements can be generated. This process is explained in Section 7.3.3.

7.3.3 Service Selection and Ranking

The next process is the task of actually determining the most relevant services for the processed query. This process is depicted in Figure 7.1, in view of the complete service discovery scenario for a given query. The way the entire process is handled is illustrated in Algorithm 7.1. As depicted, the cosine similarity between the query vector and the weighted feature vectors of the service clusters is computed first. This is done to minimise the time required to generate the most relevant service candidate set for a given user query, by eliminating the most irrelevant clusters representing other domains at the beginning itself. Next, the obtained query-cluster cosine similarity values are ranked, and the only the top-3 clusters with the highest values are chosen as potential candidates for service matching and are added to *candidate-cluster-set*.

After this, the query-service cosine similarity is computed for each service belonging to the clusters in the *candidate-cluster-set*. A predefined threshold of 0.75 is considered for good quality matching, and only if the computed value of query-service cosine similarity is above this threshold, such services are added to the *candidate-service-set*. After ranking all services, the results are returned to the user.

Algorithm 7.1 Service selection and ranking process

```
Input:
    Clusters and services represented in VSM
    Query $q$ represented in VSM
Output: Candidate set of most relevant services for q
 1: $candidate\text{-}cluster\text{-}set = \varnothing$
 2: $candidate\text{-}service\text{-}set = \varnothing$
 3: Compute cosine similarity between query and all cluster vectors.
 4: Rank clusters by their *cosine-similarity* value
 5: Select top 3 clusters as *candidate-cluster-set*
 6: for each service in *candidate-cluster-set* do
 7:     Compute cosine similarity with query vector
 8:     return *cosine-similarity* value of service
 9: if $cosineSimilarity(queryVector, service) > Threshold$ then    ▸ Threshold set to 0.75
10:     Add service to *candidate-service-set*
11: Rank services in *candidate-service-set* by cosine similarity value.
12: Return *candidate-service-set*
```

7.4 Experimental Results

In this section, the experimental evaluation of the improvement in the Web service discovery process using the techniques proposed for semantic analysis of query are presented. In Section 7.4.1, a comparative evaluation of improvement in Web service retrieval when the semantic query is used is discussed, using popular IR metrics like precision, recall and F-measure, for the $\mathcal{DWDS}$ service collection. Section 7.4.2 presents the Web service retrieval time to evaluate the performance of the proposed approach.

7.4.1 Web Service Retrieval ($\mathcal{DWDS}$ Repository)

In this section, we discuss the effective improvement achieved during Web service retrieval using the proposed techniques to analyse user requirements. Two different approaches were used to evaluate the effectiveness of the techniques proposed for query analysis. The first one is the *Natural Language (NL) Query Approach*, where the natural language query submitted by the user is directly used. The second approach uses the *Semantic Query*, which is generated by the Query Analysis Engine.

Since the proposed system is modelled as per traditional IR methods, the most appropriate metrics to evaluate the performance are *precision*, *recall* and *f-measure*. In this context, precision can be defined as the fraction of retrieved services that are relevant for the given query (equation (7.4)). Recall is the fraction of all relevant services successfully retrieved for the given query (equation (7.5)). F-measure is computed as a combination of precision and recall, as shown in Equation (7.6).

$$Precision\ P = \frac{|relevant\ services \cap retrieved\ services|}{|retrieved\ services|} \tag{7.4}$$

$$Recall\ R = \frac{|relevant\ services \cap retrieved\ services|}{|relevant\ services|} \tag{7.5}$$

$$F_\beta\text{-}measure = \frac{(\beta^2+1)PR}{\beta^2 P + R} \qquad where\ \ \beta^2 = \frac{1-\alpha}{\alpha} \tag{7.6}$$

where, $\alpha \in [0,1]$ and thus, $\beta^2 \in [0,\ \infty]$. When $\alpha = \frac{1}{2}$, the value of $\beta = 1$, which results in the balanced F-measure, commonly referred to as F_1. Values of $\beta < 1$ emphasise precision and values of $\beta > 1$ emphasise recall (Manning et al. 2008).

Table 7.1: Classes of queries used for performance evaluation

Class No.	**Query Class**	**Example Query**
1	Simple query (General)	*"find book author"*
2	Simple query (Particular)	*"hotel room for family with kids"*
3	Simple query (Proper nouns)	*"sunrise time in US city"*
4	Complex Query	*"stock symbol of company and current price"*

To evaluate the service discovery process, we conducted some experiments using different lengths of queries for each approach, i.e. by varying the number of keywords in the input query. Some query categories and sample queries are given in Table 7.1. A total of 100 different queries (30 each from simple query classes and 10 from the complex query class), were submitted to the system and the quality of results generated were observed. In Section 7.4.1.2, we discuss the results obtained for the simple query classes. The details about complex query classes is given in Section 7.4.1.1.

7.4.1.1 Serving Complex Queries

While running the queries on the system, we observed that complex queries were successfully identified and processed to generate a set of subqueries. However, to identify relevant services for each subquery, the system requires a special representation of services. This is because; both the constituent services and a valid sequence in which these should be invoked to serve the complex query need to be determined. This requires intelligent techniques to capture the dependencies between services. The techniques that enable composite service discovery using the developed semantic query mechanism and using captured service dependencies are discussed in Chapter 8.

7.4.1.2 Serving Simple Queries.

In this section, we present the observed results for the simple query classes. To evaluate the performance of the system, simple queries of length 1 to 10 terms were submitted to the system, 30 per class. Since each query is pre-processed, an average of 3.4 keywords was obtained per query. After identifying the correct sense of the terms in the query, enhancing the query by including each keyword's WordNet synsets in the query vector added an average of 8.1 more search terms per query. Due to this, each query resulted in an average of 11.5 keywords being submitted to the system.

Currently, the $\mathcal{DWDS}$ repository contains 12,231 services, categorized into 343 tagged clusters. Calculating recall for each query when the number of underlying documents is diverse is somewhat problematic as the retrieved document set will be very small when compared to the size of the dataset itself. Hence, the recall numbers would always be small. To overcome this limitation, a measure called *relative recall* (Frické 1998) was used. For computing relative recall, we assume that, the query which retrieved the highest number of services in the set of queries executed on the system is the one which achieved 100% recall. Then, the recall of all other queries can be measured against this value and hence the concept of relative recall. Using this strategy, the queries from different classes were executed and results are tabulated in Table 7.2, 7.3 and 7.4.

Table 7.2: Observed precision & relative recall values for Query Class 1, for *NL Query* and *Semantic Query* approaches over $\mathcal{DWDS}$ repository.

Query Length (in words)	**NL Query**		**Semantic Query**	
	Precision (%)	*Relative Recall (%)*	*Precision (%)*	*Relative Recall (%)*
1	84.56	11.24	79.13	18.19
2	75.78	17.13	83.12	39.31
3	69.02	24.18	86.57	48.93
4	63.14	31.49	89.18	59.97
6	58.04	37.12	87.79	73.45
8	51.29	43.97	83.62	87.19
10	43.89	47.15	76.37	100

Table 7.3: Observed precision & relative recall values for Query Class 2, for *NL Query* and *Semantic Query* approaches over $\mathcal{DWDS}$ repository.

Query Length (in words)	**NL Query**		**Semantic Query**	
	Precision (%)	*Relative Recall (%)*	*Precision (%)*	*Relative Recall (%)*
1	86.21	4.19	77.61	13.67
2	81.01	9.35	82.49	29.14
3	75.45	18.75	90.43	43.21
4	69.51	24.4	88.67	57.11
6	65.37	31.09	84.45	71.96
8	58.22	37.11	80.31	88.81
10	51.68	41.82	77.53	100

Table 7.4: Observed precision & relative recall values for Query Class 3, for *NL Query* and *Semantic Query* approaches over $\mathcal{DWDS}$ repository.

Query Length (in words)	**NL Query**		**Semantic Query**	
	Precision (%)	*Relative Recall (%)*	*Precision (%)*	*Relative Recall (%)*
1	92.14	1.31	90.16	9.71
2	87.31	4.39	93.32	21.31
3	81.26	7.17	95.76	43.22
4	76.53	12.34	89.91	66.51
6	67.13	17.18	84.03	81.09
8	56.11	22.91	79.34	96.15
10	48.96	24.31	73.11	100

Precision and Relative Recall. We analyse the results obtained in terms of precision and relative recall for various query classes. As seen from Table 7.2, for the *NL Query Approach*, precision is reasonably high at 84.56% when a single keyword is used, but recall is the lowest at 11.24% of highest recall observed. This is because the system literally matches the single keyword to the tags of the clusters formed and returns only those services having this tag. Also, it can be observed that as the number of query terms increase, the recall values improve slightly as more services would match the exact terms in the query. However, the precision deteriorates, as the query length increases, as the number of services matched literally to the keywords in the natural language query increases. When the query length was 10, the relative recall was still only about 47% of that obtained for *Semantic Query Approach*, while the precision was at an all-time low. Similar values were observed for query class 2, while for query class 3, since the submitted term was a proper noun, a very high precision value of 92% was obtained.

For the *Semantic Query Approach*, the relative recall was 100% for a query length of 10 words, as this essentially retrieved about 1612 services, which was the maximum number of services matched from the $\mathcal{DWDS}$ repository. Hence, as per the definition of *relative recall*, we consider this as the query which resulted in 100% recall, and measure all other query recalls against this value. For the highest recall, the precision observed was only about 77%.

Also, for query class 1 and for the *Semantic Query Approach*, the highest precision of 91.63% was observed when the query length was 3. This could be due to the fact that, effective POS tagging requires the analysis of a term's nearest neighbour terms. For example, when the length of the query is 1, a term like *'book'* can be either a verb or a noun-phrase. Therefore, results can differ and precision may suffer if the user meant it as either sense. However, if the query is changed to *'book author'*, which is a query of length 2, the context of the user is much clearer and so the precision improves. We observed the best precision was obtained when the query length was between 2 - 4, as the relationships between the words can be correctly captured and the most relevant services can be retrieved.

Similar results were observed for query class 2 and 3. Due to the existence of proper nouns in queries belonging to query class 3, the *NL Query* approach recorded the highest precision for query length 1, even though the recall was only 1.31%. The recall did not improve much, even for query length 10, and achieved only 49% precision. Overall, the *Semantic Query* approach achieved approximately 16% improvement in precision and 37% increase in recall, when compared to the *NL Query* approach.

F-measure. Using the observed precision and relative recall values, the F-measure values were computed, by using relative recall *(relR)* in place of recall in Equation (7.6). The balanced F-measure is given by β value of 1. Also, for testing the precision-oriented performance, a β value of 0.5 was used and for testing recall-oriented performance β value of 2 was used. Hence, the value of F_1 *-measure*, $F_{0.5}$ *-measure* and F_2 *-measure* are given by Equations (7.7), (7.8) and (7.9) respectively.

$$F_1 \text{ -}measure = \left(\frac{2\ .\ P\ .\ relR}{P + relR}\right) \tag{7.7}$$

$$F_{0.5} \text{ -}measure = \left(\frac{(1.25)\ .\ P\ .\ relR}{(0.25)\ .\ P + relR}\right) \tag{7.8}$$

$$F_2 \text{ -}measure = \left(\frac{5\ .\ P\ .\ relR}{4P + relR}\right) \tag{7.9}$$

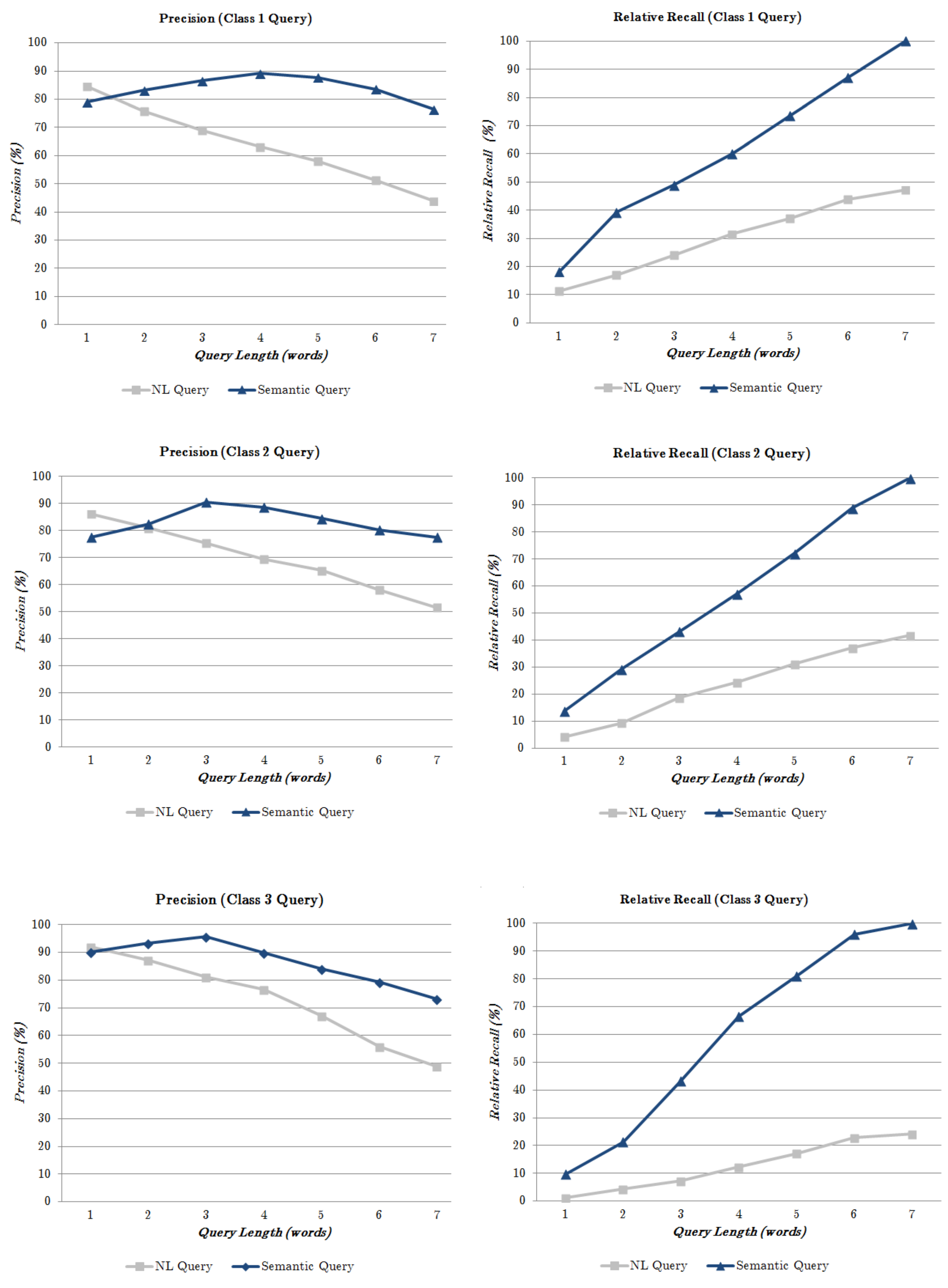


Figure 7.4: Observed Precision and Relative Recall for *NL Query* and *Semantic Query* Approaches (for Query Classes 1, 2 and 3)

F-measure is an indicator of the overall performance quality of an IR system and lies in the range 0 to 100% (best possible value). A F-measure value of 50% or more is considered as good retrieval performance (Makhoul et al. 1999). The computed values for both the approaches for $\beta = 1$, 0.5 and 2 are shown in Table 7.5. An average of precision and relative recall obtained for all simple query classes was used for F-measure computation.

Table 7.5: Average F-measure values for *NL Query* and *Semantic Query* approaches for $\beta = 1$, 0.5 and 2 for $\mathcal{DWDS}$ repository

Query Length (in words)	**F-measure (%) (NL Query)**			**F-Measure (%) (Semantic Query)**		
	F_1	$F_{0.5}$	F_2	F_1	$F_{0.5}$	F_2
1	10.49	22.24	6.87	23.72	41.40	16.62
2	18.27	34.17	12.47	44.44	62.68	34.42
3	27.33	44.23	19.78	60.31	75.58	50.18
4	34.30	49.34	26.29	72.61	81.76	65.30
6	39.31	50.96	31.99	80.16	83.24	77.30
8	42.59	49.36	37.45	85.63	82.85	88.61
10	42.34	45.66	39.47	86.15	79.54	93.96

Figure 7.5, 7.6 and 7.7 show the balanced, precision-oriented and recall-oriented F-measure values for *NL Query* and *Semantic Query* approaches. These results can be interpreted as follows.

1. *Balanced Performance (F_1):*

   (i) *For NL Query:* The F_1-*measure* values were in the range 11% to 42%, which indicates that the overall retrieval accuracy was poor when the natural language query was directly used.

   (ii) *For Semantic Query:* The balanced system performance, as indicated by the F_1-*measure* values, was found to be quite good. The worst performance was when the the query length at 23%, while the best performance was when the query length was 8 and 10 words, at 86%. This indicates that the system's overall performance and accuracy while supporting Web service discovery was very much improved, as evidenced by the 16% increase in precision and 37% increase in recall, on average.

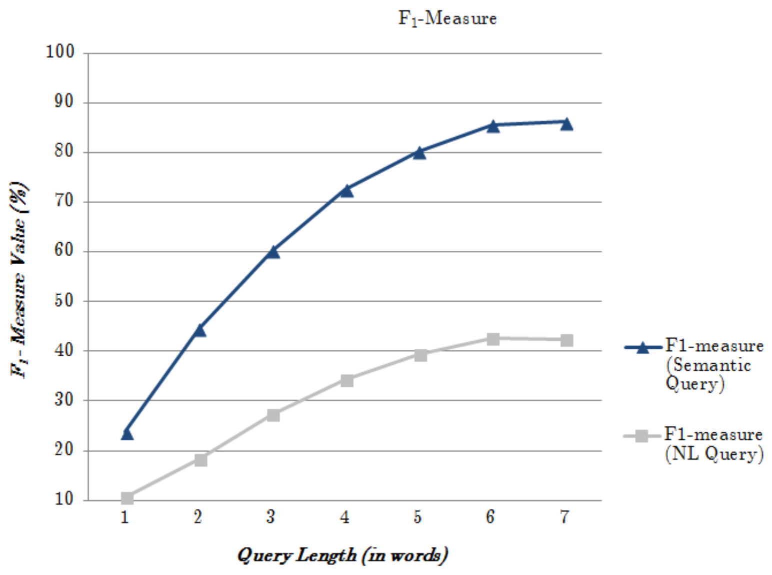


Figure 7.5: Balance performance for $\mathcal{DWDS}$ (Average F-measure at $\beta = 1$)

2. *Precision-oriented Performance ($F_{0.5}$):*

   (i) *For NL Query:* The values were in the range of 22% to 45% for plain natural language query. Hence, it can be concluded that, the precision obtained for natural language query was below average.

   (ii) *For Semantic Query:* The values achieved were in the range 41% to 79% indicating very good retrieval accuracy. This means that the semantic analysis of the natural language query was effective and was able to capture user context correctly, in most of the cases. As the query length exceed 4, the f-measure values deteriorate, to 79% for a query of length 10.

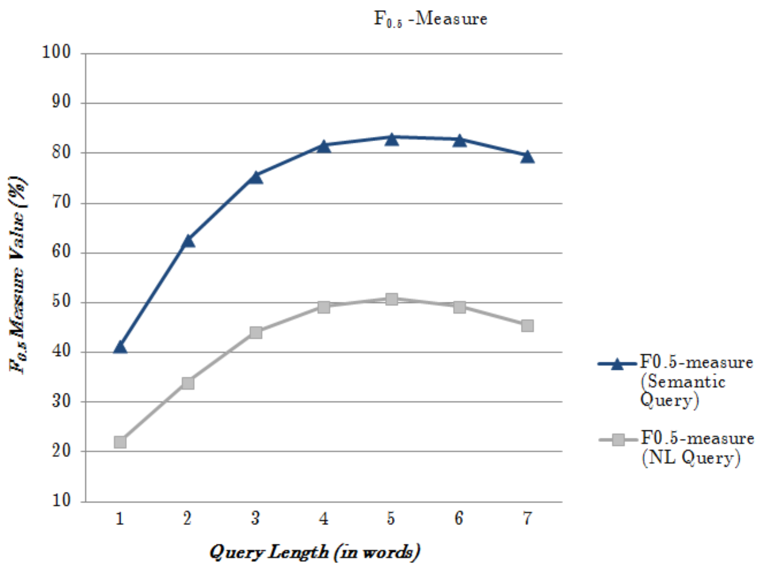


Figure 7.6: Precision-oriented performance for $\mathcal{DWDS}$ (Average F-measure at $\beta = 0.5$)

3. *Recall-oriented Performance (F_2):*

   (i) *For NL Query:* The recall achieved for this approach also was quite low, as the F-measure values were in the range 7% to 39%.

   (ii) *For Semantic Query:* The recall performance was also observed to be very good, as indicated by the F-measure values ranging from 16% to 94%. As a larger set of services is retrieved, the possibility of matching more relevant services to the varied words used in the user query increases, instead of literal keyword matching. Hence, good recall-oriented performance underscores the effectiveness of the proposed semantic analysis techniques.

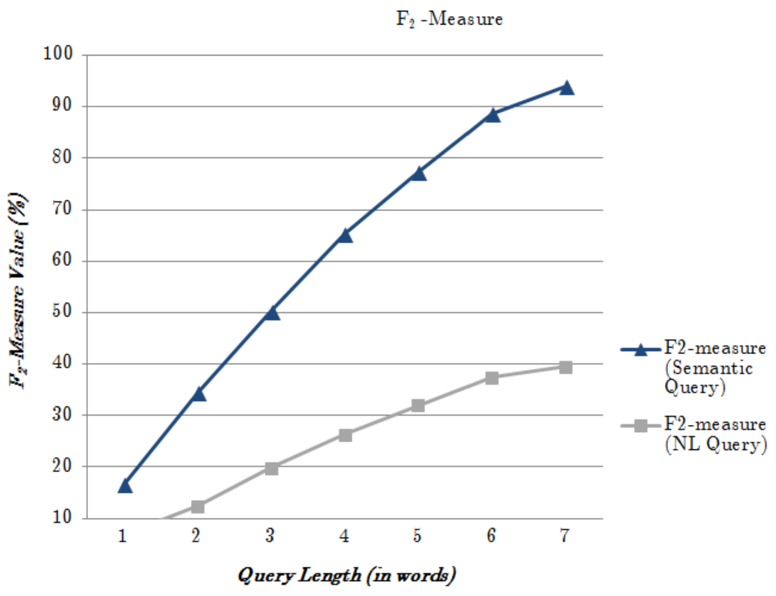


Figure 7.7: Recall-oriented performance for $\mathcal{DWDS}$ (Average F-measure at $\beta = 2$)

7.4.2 Web Service Retrieval Time

Precision and recall are the most popular measures for IR performance evaluation, but users may not always care about relevancy only. For example, depending on their need, users may sometimes not want all services relevant to their search; but may be more concerned about getting a good enough answer in a short amount of time. To evaluate this, an additional metric that measures the average time taken for serving a given user query was used. The *result generation time* is the time from the start until the completion of a service retrieval task, and is recorded in seconds. The average time required for producing the results for the *NL Query* and *Semantic Query* approaches over $\mathcal{DWDS}$ Repository is given in Table 7.6.

Table 7.6: Average Result Generation Time for *NL Query* and *Semantic Query* approaches using *DWDS* repository

Query Length (in words)	Time for NL Query (in seconds)	Time for Semantic Query (in seconds)
1	4	13
2	11	25
3	17	37
4	25	48
6	33	61
8	41	83
10	49	98

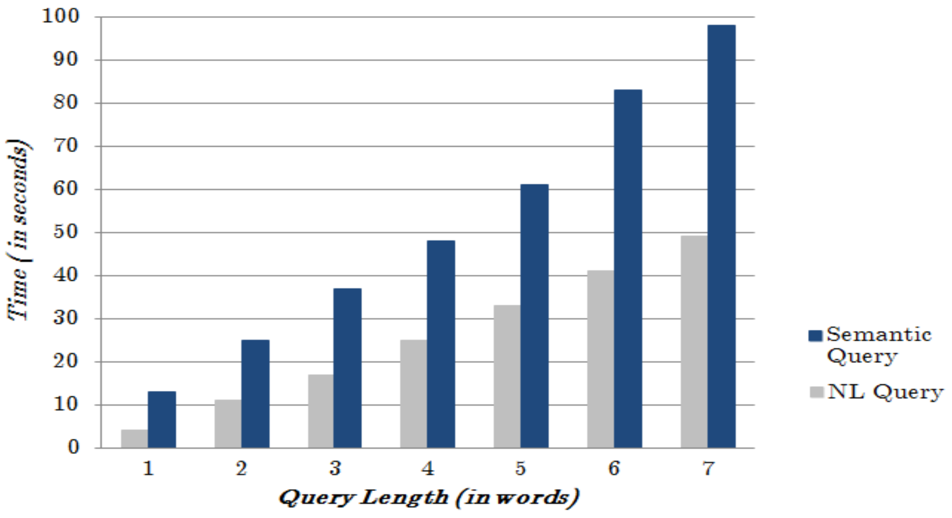


Figure 7.8: Comparison of Result Generation time of *NL Query* and *Semantic Query* approaches using *DWDS*

On an average, the *Semantic Query* approach took about 26.4 seconds more than the *NL Query* approach, to generate service search results. Even though some latency in introduced in the generation of service discovery results, when the semantic query approach is used, the improvement achieved in terms of precision and recall is significant, as a result a higher result generation time can be considered as an acceptable tradeoff.

7.5 Summary

In this chapter, the proposed techniques for semantically analysing the natural language query of the user to address the issues of synonymy, polysemy and hypernymy/hyponymy were discussed. Through experimental analysis over the *DWDS* repository, it was observed that the proposed techniques resulted in an average improvement of over 16% in precision and 37% in recall. The computed f-measure values, F_1, $F_{0.5}$ and F_2 indicate that the balances, precision-oriented and recall-oriented performance of the proposed approach was good. To evaluate the latency of the system, when the proposed techniques were used, the result generation time was computed, which showed that the proposed semantic query approach required an average of about 26.4 seconds more than the natural language query approach, to generate results. However, as the precision is significantly increased, this is considered an acceptable trade-off.

We also discussed the methodology designed for automatically identifying and processing complex queries to determine its subqueries in this chapter. However, to execute each subquery parallely and to determine the correct order in which the services are to be chained to serve the given complex query, requires intelligent mechanisms that can capture service dependencies. Such a technique is proposed and discussed in the next chapter.

Publications

(based on work presented in this chapter)

1. Sowmya Kamath S and Ananthanarayana V.S, *"Semantic Similarity based Context-aware Web Service Discovery using NLP Techniques"*, Journal of Web Engineering (JWE), Rinton Press, Princeton, New Jersey, Volume 15, Issue 1 & 2, 2016. [ISSN: 1540-9589] (SCI Indexed)

   Status: Online.

Chapter 8

Composite Service Discovery

8.1 Introduction

In Chapter 7, the process of performing Web service discovery in the $\mathcal{DWDS}$ framework when the user submitted query is simple was described. However, in SOA based systems, a crucial requirement is service reuse. The idea of service reuse is similar to that of code reuse where, newer and more advanced applications and systems can be built using modular components of existing systems. In a similar way, whenever a single basic service is inadequate for performing a given task, a composition of two or more such basic services may be required. Such a service is referred to as a composite Web service. It is composed of existing basic services (also known as constituent services or component services), invoked in a particular sequence, and with a designated flow structure. The functionality provided by such a composite service is an integration of the functionality of its constituent services.

In a large repository, each such constituent service chosen may often have one or more alternatives, which can possibly perform the subtask in a similar (or better) way as the originally chosen service. Hence, several alternate composite services may be possible; each using varied available services in different sequences, to achieve the same task. Generally, application designers have no choice but to resort to manually determining the most suitable services for developing a particular service based application, and to determine the order in which these should be used, to perform the required functionality. However, manual identification of such constituent services and their correct invocation sequence is a time consuming task, especially when the repository contains a large number of services. Also, it may be nearly impossible to manually identify all alternative composite services, so that the designer can have a choice of composite service templates. Hence, intelligent, automated methods that can capture service signatures and dependencies for generating all possible valid composite service templates can be very beneficial to application designers. In this chapter, we present a semantics based graph approach for capturing service interface dependencies, to enable composite service discovery in a time-aware manner. Our main contributions, as reported in this chapter 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 semantics based mechanism was amenable to determining the correct sequence in which services should be invoked 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 rest of this chapter is organised as follows. Section 8.2 describes the defined problem addressed in this chapter. In Section 8.3, we discuss the methodology designed for capturing the service dependencies of Web services from their service descriptions, representing the captured service dependencies in a formal manner and executing complex queries using the formal representation. In Section 8.4, we discuss experimental results on the performance of the proposed techniques in terms of accuracy and result generation time, and also present the theoretical complexity of the proposed composite service discovery approach. Section 8.5 summarizes the work presented in this chapter.

8.2 Problem Statement

The problem that is addressed in this chapter is defined here:

> *Given a set of Web services and a complex service request for composable service templates, to design an approach for capturing service dependencies that can identify one or more such templates, along with their constituent services and the correct order of invocation.*

We propose a solution that aims to capture the interface details and service dependencies of Web services and then uses them to identify both constituent services and the correct sequence for valid composite templates for a given query. As WSDLs do not provide any interface information explicitly, semantic service descriptions like OWL-S which contain such details within the description itself have to be employed. Hence, we chose to use the OWL-S descriptions of services specified in WSDL, as input and output details of a service are explicitly defined in the Profile Model of the OWL-S specification. In Section 8.3, we discuss the proposed composite service discovery methodology using the generated OWL-S and describe the techniques developed for capturing and using the service dependencies of services.

8.3 Proposed Methodology

In this section, the techniques designed for enabling composite service discovery using the captured service dependencies of all services in a given dataset are described in detail. To capture the functional semantics and also the input/output service dependencies correctly, the OWL-S[1] of a WSDL service description was used. Figure 8.1 provides an overall view of composite service discovery process.

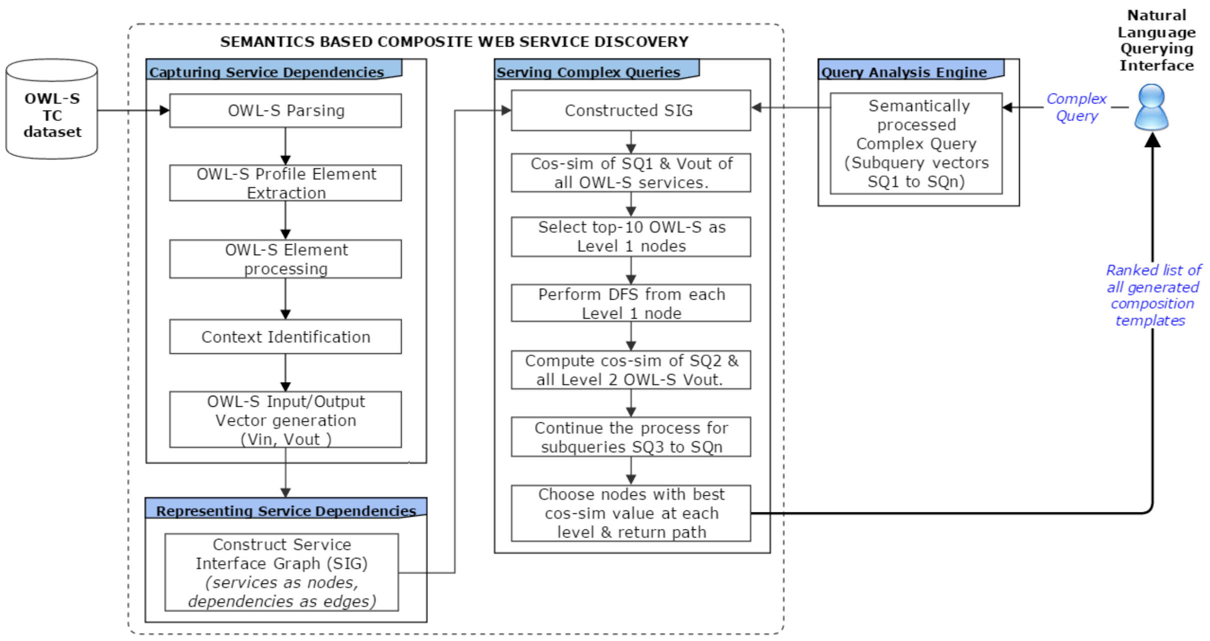

SEMANTICS BASED COMPOSITE WEB SERVICE DISCOVERY
OWL-S TC dataset
Capturing Service Dependencies
OWL-S Parsing
OWL-S Profile Element Extraction
OWL-S Element processing
Context Identification
OWL-S Input/Output Vector generation (Vin, Vout)
Representing Service Dependencies
Construct Service Interface Graph (SIG) (services as nodes, dependencies as edges)
Serving Complex Queries
Constructed SIG
Cos-sim of SQ1 & Vout of all OWL-S services.
Select top-10 OWL-S as Level 1 nodes
Perform DFS from each Level 1 node
Compute cos-sim of SQ2 & all Level 2 OWL-S Vout.
Continue the process for subqueries SQ3 to SQn
Choose nodes with best cos-sim value at each level & return path
Query Analysis Engine
Semantically processed Complex Query (Subquery vectors SQ1 to SQn)
Natural Language Querying Interface
Complex Query
Ranked list of all generated composition templates

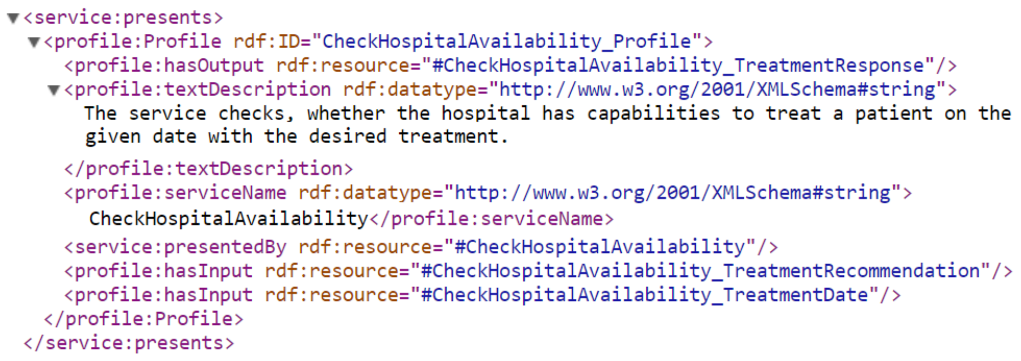
Figure 8.1: Proposed composite service discovery methodology

8.3.1 Capturing Service Dependencies

The 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

[1]The OWL-S TC 4 dataset was used for experimental evaluation of the proposed approach. Available at http://projects.semwebcentral.org/projects/owls-tc/

data about a web service while recommending relevant ones to satisfy user queries. A sample OWL-S profile model defining the interfaces of its corresponding Web service is shown in Figure 8.2.



```
▼<service:presents>
  ▼<profile:Profile rdf:ID="CheckHospitalAvailability_Profile">
     <profile:hasOutput rdf:resource="#CheckHospitalAvailability_TreatmentResponse"/>
    ▼<profile:textDescription rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
       The service checks, whether the hospital has capabilities to treat a patient on the
       given date with the desired treatment.
     </profile:textDescription>
     <profile:serviceName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
       CheckHospitalAvailability</profile:serviceName>
     <service:presentedBy rdf:resource="#CheckHospitalAvailability"/>
     <profile:hasInput rdf:resource="#CheckHospitalAvailability_TreatmentRecommendation"/>
     <profile:hasInput rdf:resource="#CheckHospitalAvailability_TreatmentDate"/>
   </profile:Profile>
 </service:presents>
```

Figure 8.2: Profile model of a sample OWL-S document

OWL-S Parsing. The profile model of each OWL-S service in the dataset is first captured while parsing the OWL-S file. To extract the information, a OWL-S Document Parser was used, to parse each file and extract the contents of required elements - the <profile:hasInput>, <profile:hasOutput> and <profile:textDescription>.

OWL-S Profile Element Extraction. During the indexation phase, pertinent information from the OWL-S profile is extracted, to enable Web service representation in the form of service dependencies. The OWL-S profile model has a set of elements that provide information about the service inputs, outputs and capabilities - <profile:hasInput>, <profile:hasOutput> and <profile:textDescription>. These elements are extracted and are processed further to extract information that can be used to index the web services conceptually.

Element Processing. The extracted OWL-S profile elements, <profile:hasInput> and <profile:hasOutput> contain names that represent the functionality of the service. These are natural language phrases, which are extracted and processed further for obtaining the functional semantics and context of the service. The processing applied is similar to the extraction processes applied to WSDLs (discussed in Chapter 5, Section 5.4). Various techniques like determining term splitting positions, tokenization, stopword & function word removal are performed at this stage.

Context Identification. The overall functionality of a 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. As the number of terms in the <profile:hasInput> and <profile:hasOutput> is limited, the <profile:textDescription> content is used for capturing the domain of a service. Hence, POS tagging is performed on this extracted description, and any identified stopwords and function words were removed.

Next, the synonym sets (referred to as synsets) are found using WordNet (Miller et al. 1990) to determine the multiple meaning of the words. To identify the correct synset so that the correct context is captured, a word sense disambiguation algorithm called JCN Similarity algorithm (Jiang et al. 1997a) is applied. JCN returns a score denoting how similar the word senses of the two input synsets are. This similarity measure is based on a combination of using edge counts in the WordNet *'is-a'* hierarchy and using the information content values of the WordNet concepts. It computes values that indicate the semantic distance between words and hence the inverted value is used as a measure of semantic relatedness. The computed values are used to choose the most relevant word sense for each feature term. Term weighting and ranking techniques are applied to the resultant feature set to determine the top 5 terms, which are then used as a weighted feature vector for a OWL-S document.

OWL-S Input/Output Vector Generation. Finally, the weighted and ranked features obtained from the <profile:hasInput> element are saved as the OWL-S document's input vector ($\vec{V_{in}}$) and those obtained from the <profile:hasOutput> are used as the OWL-S document's output vector ($\vec{V_{out}}$). These vectors contain the top-5 terms obtained after tag weighting and ranking, that most represent the interface details of the service.

This procedure is repeated for all OWL-S files in the dataset, at the end of which, an indexed dataset that contains the input/output vector representation of every OWL-S document is obtained. These input and output vectors represent the interface dependencies of each OWL-S service, and are used for constructing a service dependency graph called the *Service Interface Graph (SIG)*. The process of representing these captured service dependencies in a formal manner is described in Section 8.3.2.

8.3.2 Representing Service Dependencies

Determining composite service templates is a complex process that has to consider several crucial requirements, like -

- Capturing the interface details of each basic service to find its capabilities.
- Searching and identifying one or more sets of suitable basic services that can together satisfy the user requirements, based on the service dependencies.

- Determining the sequence in which different basic services are to be placed, such that the output of the preceding service in the sequence matches the input expected by the next service, to yield the final desired output.

To serve these requirements, the proposed methodology uses a Service Interface Graph (SIG) to formally represent the captured service dependencies. 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 'S_i' to node 'S_j' signifies that the output of a service 'S_i' is similar to the input accepted by service 'S_j' i.e., service 'S_i' and 'S_j' can be chained together in order. Algorithm 8.1 depicts the process of constructing the service interface graph.

Algorithm 8.1 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 $\vec{V}_{out}$ of service $S_i$ with $\vec{V}_{in}$ of all other services $S_k$
 4:     for each service node $S_k$ do
 5:         if $cos\text{-}sim(\vec{V}_{out_{S_i}}, \vec{V}_{in_{S_k}}) > cutoff$ 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 a particular service ($\vec{V}_{out_{S_i}}$) and input vector of all other services ($\vec{V}_{in_{S_k}}$) is determined as per equation (8.1).

$$\cos\theta(\vec{V}_{out_{S_i}},\ \vec{V}_{in_{S_k}}) = \frac{\vec{V}_{out_{S_i}} \cdot \vec{V}_{in_{S_k}}}{\|\vec{V}_{out_{S_i}}\|\ \ \|\vec{V}_{in_{S_k}}\|} = \frac{\sum_{j=1}^{n}(w_{j,i}\ .\ w_{j,k})}{\sqrt{\sum_{j=1}^{n} w_{j,i}^2} \cdot \sqrt{\sum_{j=1}^{n} w_{j,k}^2}} \tag{8.1}$$

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 some other service S_k, in the considered dataset.

The pre-defined similarity cutoff for adding edges is currently fixed at 0.75 and a high value is chosen to ensure that there is a near-perfect match between the interfaces. If the computed cosine similarity value is found to be greater than 0.75, then an edge is added between the two service nodes. After the addition of each edge, the graph is checked for any cycles. If the newly added edge has introduced a cycle, then the recently added edge is discarded. The main objective here 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 process is continued for each service node S_k, and the SIG is finally constructed. Once constructed, the SIG is stored in the memory using its adjacency list representation. Within the graph, each node contains -

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

The process of SIG construction takes place only once, when the server is initially started for the first time. Hence, it is also a part of the pre-processing step and is an off-line process. Once constructed, the graph resides in main memory and the same graph is used for serving each complex query, by performing a graph traversal as described in Section 8.3.3.

8.3.3 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 (as per the process described in section 7.3, Chapter 7). This task is performed by the Query Analysis Engine and the output of this processing is - the query vector (in case of simple query) or subquery vectors (in case of complex query). As discussed, for a simple query, there are no *'and'/'or'* keywords, so the cosine similarity value between query 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 SIG has to be traversed based on the identified subquery components, to determine relevant services for each subquery.

For a complex query, firstly, we assume that the user always specifies what is wanted (i.e. the output). Therefore, the terms obtained for each subquery are considered as part of its output vector. For the processed complex query, the set of subquery feature vectors $\{SQ_1, SQ_2 \ldots SQ_n\}$ are considered in order. Based on the identified occurrence of *'and'* and *'or'*, the logical structure of the complex query is also considered.

Figure 8.3 depicts the abstract workflow while executing a sample complex query - *"book hotel or resort and hire taxi"*. The logical representation of this complex query is $((SQ_1 \vee SQ_2) \wedge SQ_3)$. Hence a valid composite template for this complex query is - $\{$*best hotel booking service, best taxi booking service*$\}$ and also $\{$*best resort booking service, best taxi booking service*$\}$. Hence, the logical representation of the complex query is used to determine the way composition templates should be returned to the user.

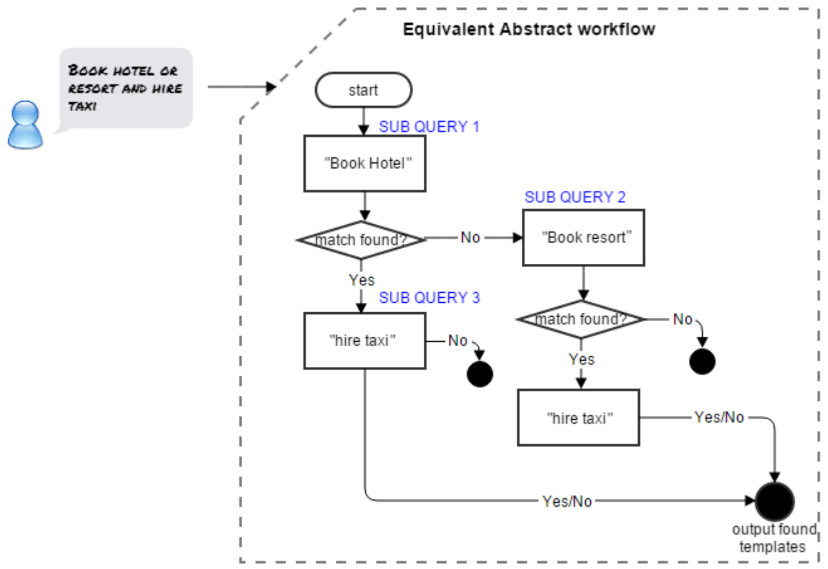


Figure 8.3: Equivalent abstract workflow for a sample complex query

Algorithm 8.2 depicts the process of executing complex queries using the constructed SIG. To serve a complex query, again the concept of similarity between subquery vectors and service interface vectors is considered. For the first subquery sq_1 of a complex query, cosine similarity of subquery vector $\vec{V}_{sq_1}$ with the output vectors of all services in the SIG is computed using the equation (8.2).

$$\cos\theta(\vec{V}_{out_{S_k}}, \vec{V}_{sq_t}) = \frac{\vec{V}_{out_{S_k}} \cdot \vec{V}_{sq_t}}{\|\vec{V}_{out_{S_k}}\| \ \|\vec{V}_{sq_t}\|} = \frac{\sum_{j=1}^{n}(w_{j,k} \ . \ w_{j,sq_t})}{\sqrt{\sum_{j=1}^{n} w_{j,k}^2} \ . \ \sqrt{\sum_{j=1}^{n} w_{j,sq_t}^2}} \tag{8.2}$$

where,

$\vec{V}_{out_{S_k}}$ represents the output vector terms of a service S_k in the SIG.

$\vec{V}_{sq_t}$ represents subquery vector terms for a complex query with n subqueries.

w represents relative weight of a term with respect to service S_k & subquery sq_t.

Algorithm 8.2 Executing complex queries using SIG

```
Input:
    Service Interface Graph SIG
    Number of subqueries $n$ of complex query $Q$
    Subquery vectors $\vec{V}_{sq_n}$ 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 $\vec{V}_{sq_1}$ and $\vec{V}_{out}$ 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 $\vec{V}_{sq_l}$ & $\vec{V}_{out}$ 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.
```

Using the computed cos-sim values, a ranking is performed, and ten services with the best cos-sim values are selected. This gives the top-10 start-points for finding all potential composite service templates, for the rest of the subqueries. This is considered as the first level of services that satisfy a first 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 level 1 input service as source.

During DFS, 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. To optimize the result generation time when a complex query is received, the graph is traversed a predefined number of times (10 times in our implementation), for each possible source node, in parallel. 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.

8.4 Experimental Results and Analysis

In this section, a discussion on the theoretical and experimental evaluation of the proposed approaches for discovering composite services is presented. The retrieval performance of the method for various complex query types and for different dataset sizes is summarized in Section 8.4.1. The number of accurate composite service templates generated by the system was observed and the results are presented in Section 8.4.3. A theoretical analysis of the proposed methodology's complexities is discussed in Section 8.4.4.

8.4.1 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. The domain-wise service distribution of this dataset is shown in Table 8.1.

Table 8.1: OWL-S TC 4 Service dataset statistics

Service Domain	**No.of Available Services**
Communication	56
Economy	349
Education	285
Food	34
Geography	60
Medical	73
Simulation	16
Travel	165
Weapon	38

We also considered different types of complex queries, based on the occurrence of *'and'* & *'or'* (as shown in Table 8.2). To evaluate the service discovery process, experiments were conducted using these different types of queries over various dataset sizes. 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. The number of OWL-S documents considered for the various testcases were 100, 300, 500 and 1076 and all three types of complex queries were submitted to the system. Sample queries in each complex query class and the considered OWL-S category are shown in Table 8.3 below.

Table 8.2: Query classes and details

Class	**Complex Query Type**	**Description**
1	Sequence	contains one or more *'and'*
2	Choice	contains one or more *'or'*
3	Mixed	contains both *'and'* + *'or'*

Table 8.3: Experimental setup and sample queries used for different testcases

Test case No.	**No.of services used**	**Service Domains**	**Complex Query Type**	**Sample Queries**
1	100	Medical (60) Geography (40)	sequence	*treatment and hospital room availability*
			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 country*
			mixed	*scholarship or research funding and part time job in country*
3	500	Travel (165) Geography (60) Economy (275)	sequence	*hotel in city and car hire*
			choice	*sports or adventure activity near address*
			mixed	*sports events and weather in city or country*
4	1076	All categories	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*

The metrics used to evaluate the discovery performance were *precision* and *recall* as per equations (8.3) and (8.4). Balanced F-measure, which is the harmonic mean of the precision and recall performance of a IR system was also computed (as given by equation (8.5)).

$$Precision = \frac{|relevant\ services| \cap |retrieved\ services|}{|retrieved\ services|} \tag{8.3}$$

$$Recall = \frac{|relevant\ services| \cap |retrieved\ services|}{|relevant\ services|} \tag{8.4}$$

$$F\text{-}Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{8.5}$$

Table 8.4 summarizes the observed experimental results for the various cases considered. 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.

Table 8.4: Experimental results for Web service discovery for various testcases

Test case No.	**No.of services used**	**Complex Query Type**	**Precision (%)**	**Recall (%)**	**F-Measure (%)**	**Result Generation Time (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

Figure 8.4 shows the precision-recall performance of observed during Web service discovery. The balanced F-measure values were computed and are as shown in Figure 8.5. 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%.

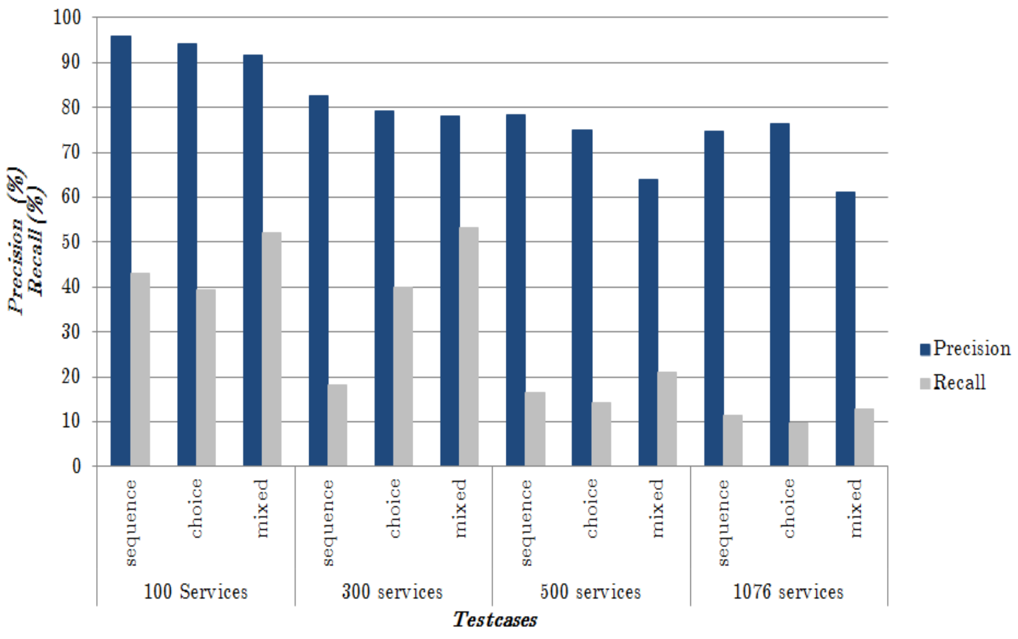


Figure 8.4: Observed precision-recall values

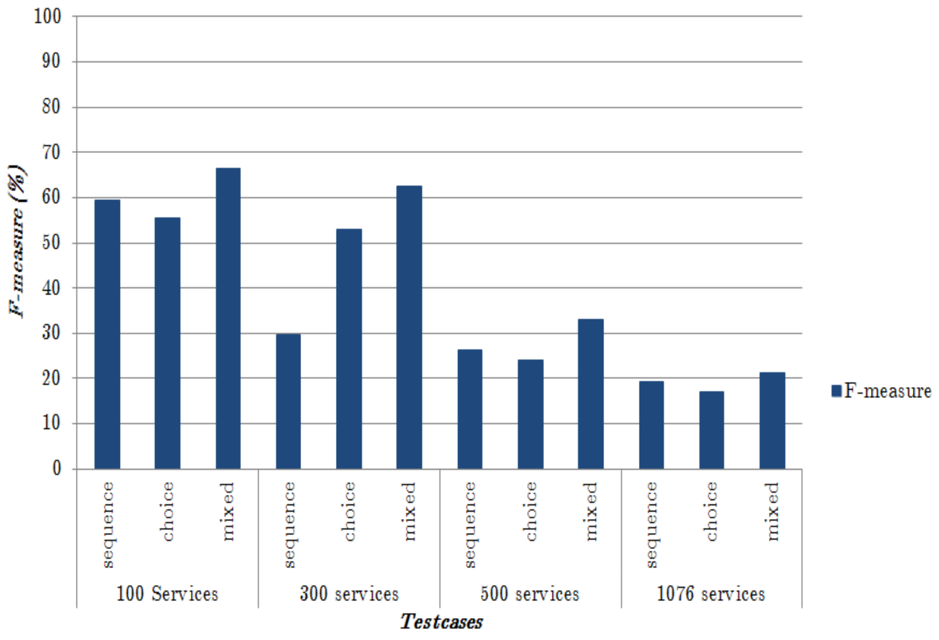


Figure 8.5: Observed f-score values

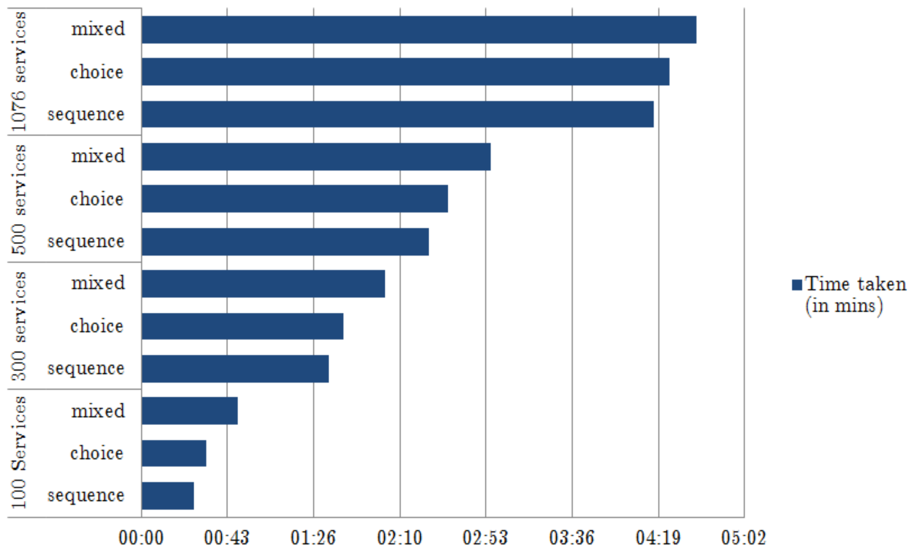


Figure 8.6: Result Generation Time

Figure 8.6 shows the time taken to generate the results in each testcase for different complex query types. The average result generation time was about 2 minutes, 22 seconds. Even though the result generation was reasonable, the important factor here is to evaluate if valid composite service templates were identified. To demonstrate the process of generation of composite service templates, a sample scenario is considered and discussed in Section 8.4.2.

8.4.2 Composite Service Discovery - An Example Scenario

Consider a sample complex query from an application designer, who wishes to build an application that helps people book their hotel room or resort, and check out local activities at one go. The submitted complex query is -

Query Q = *"Service to book hotel room or resort and surfing spots at a city"*.

However, there may not be any one service that can do all this, so the designer has to plan a composition of suitable services. In such a situation, the SIG can be helpful in identifying those services that can be chained together to get the required output.

1. During query processing, the complex query would be resolved into its constituent subqueries, by first splitting it at the tokens marked as coordinating conjunctions and then each subquery is processed as per the process described earlier (in Question (a) of Part III). After this each subquery is represented as a semantic feature vector.

2. The next task is to construct the Service Interface Graph (SIG) (Algorithm 8.1). For this, all OWL-S services are represented as input/output vectors.

3. To identify and recommend any potentially composable service sets to the service designer, the SIG is used to capture the input/output dependencies. The SIG is constructed by recursively computing the cosine similarity between a service S_i's $\vec{V}_{out}$ and the $\vec{V}_{in}$ of all other services. If this computed value is more than a pre-defined cut-off of 0.75, then an edge is added between these two nodes in the SIG. This process is continued for all services in the dataset (some services may not match with any other service, these are left out of the SIG).

4. For finding the services matching that can serve each subquery, Algorithm 8.2 is used. The different subquery semantic vectors (after splitting and all processing) of the example given above are -

   $\vec{V}_{sq_1} = \{$*('book', 'reserve', 'hold'), ('hotel'), ('room')*$\}$

   $\vec{V}_{sq_2} = \{$*('resort', 'luxury hotel', 'holiday resort')*$\}$

   $\vec{V}_{sq_3} = \{$*('surf', 'surfboard', 'surfride'), ('spot', 'point', 'place'), ('city', 'metropolis', 'urban center')*$\}$

5. First, the cos-sim of $\vec{V}_{sq_1}$ with the $\vec{V}_{out}$ of all the services in the SIG is computed. The top 10 cos-sim values are chosen as Level 1 services. Next, a DFS is performed from each node in level 1, and for each node visited, the cosine similarity is computed between the node's $\vec{V}_{out}$ and the $\vec{V}_{sq_i}$ vectors. The node with the highest cos-sim value is stored. Finally, the path and all nodes between source node and node with best output is returned. This is one possible composite service template. Similarly, other possible templates can be identified by recursively traversing the SIG.

For the complex query considered, Table 8.5 shows one possible composite service template, that can serve the application designer's need. Table 8.6 shows another option for the same requirement. This example demonstrated how the SIG helps in capturing the service interface dependencies, which can be used for discovering one or more service composition templates. To evaluate this accuracy and performance of the discussed approach, the quality of generated templates is taken into consideration, the results of which are presented in Section 8.4.3.

Table 8.5: One possible Composite Service Template for given complex query

No.	Service Name	Input vector $\vec{V}_{in}$	Output vector $\vec{V}_{out}$
WS309	citycountryduration_ Hotel -service	{city, country, duration}	{hotel, reservation}
WS281	CityLuxuryhotel	{city}	{luxury, hotel}
WS957	surfing_beach_service	{city, country}	{surf, beach, address}

Table 8.6: An alternate Composite Service Template identified for given complex query

No.	Service Name	Input vector $\vec{V}_{in}$	Output vector $\vec{V}_{out}$
WS12	hotel_Worldwideservice	{hotel, city, date}	{reservation}
WS	Geographical-regionLuxuryhotel	{geographic, region, date}	{luxury, hotel}
WS940	surfingorganization_destii _Bestservice	{organization}	{surf, point, city}

8.4.3 Composite 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.
4. *Templates with I/O datatype conflicts ($CT_{conflict}$).* Templates that may be correct at the semantic level of 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. If this service is recommended in a composite service template, where other services require 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.

Table 8.7 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, incorrect* and *I/O datatype conflict* templates. As per this criteria, the accuracy of composite service template generation using the proposed methodology is calculated as per equation (8.6).

$$\%Accuracy = \frac{(CT_{correct} + CT_{partial} - CT_{incorrect} - CT_{conflict})}{CT_{total}} * 100 \tag{8.6}$$

As seen from Table 8.7, the average composite service template accuracy observed for all the testcases considered was 70.68%. 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. In the next section, we present a theoretical analysis of the performance of the developed approach.

Table 8.7: Composition template generation accuracy

Services used	**Complex Query**	**Composite Service Templates generated**				**Accuracy (%)**
		Correct	*Partial*	*Incorrect*	*I/O Conflict*	
100	sequence	2	1	0	0	100
	choice	2	0	0	0	100
	mixed	1	1	0	0	100
300	sequence	3	1	1	0	60
	choice	4	1	0	0	100
	mixed	2	1	0	1	50
500	sequence	4	2	1	1	50
	choice	3	2	1	1	42.86
	mixed	2	2	1	1	33.33
1076	sequence	4	3	1	2	40
	choice	4	2	1	1	50
	mixed	3	3	1	2	33.33
					Average Accuracy	**70.687**

8.4.4 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.

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 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.

8.5 Summary

In this chapter, a methodology for discovering composite service templates as per user's complex requirements is proposed, that uses semantics based graph traversal techniques to capture service dependencies. The approach is based on formal representation of captured service dependencies using a Service Interface Graph, which is traversed to determine constituent services and correct invocation sequence for a given complex query. To evaluate performance of this approach, several experiments using varied sized OWL-S datasets were used and the performance was observed, in terms of retrieval precision and recall, and time taken for result generation. The average value of observed precision for all the testcases was 79.28% and the average result generation time was about 2 minutes, 22 seconds. In addition to retrieval performance, the number of correct, partial, incorrect and templates with I/O datatype conflicts were determined by manual inspection, to measure accuracy. The overall accuracy of composite service template generation was found to be 70.68%. The experimental results show that the developed approach generated composite service templates with a good level of accuracy and also is scalable to handle larger datasets efficiently.

Publications

(based on work presented in this chapter)

1. Sowmya Kamath S and Ananthanarayana V.S, *"Discovering Composable Web Services based on Functional Semantics and Service Dependencies using Natural Language Requests"*, Journal on Information System Frontiers, Springer Hiedelberg, ISSN: 1387-3326 (SCI Indexed)

   Status: In press

Chapter 9

Conclusions and Future Work

With the explosive growth of the Web data and services, there is a significant need for simplifying the process of service discovery to provide better matching, composition and integration capabilities without human intervention. In this thesis, we investigated four main problems in the domain of Web service discovery - finding distributed Web services from heterogeneous sources on the Web; effective metadata generation and dynamic categorization for large service collections; understanding user context and requirements during the service querying process; and enabling automated composite Web service discovery. Each problem posed its own challenges, and we proposed novel approaches to address these challenges effectively. We summarize our contributions in each of these areas below.

In Chapter 4, we discussed a distributed Web service discovery framework, $\mathcal{DWDS}$, for finding and retrieving published service descriptions on the Web to enable semantics based service discovery. The developed framework extends autonomic features for repository management and redundancy control for automating the service repository management process. *Periodic active crawling* to find more services and *selective crawling* to check the status of indexed repository services ensured that the service collection is kept up-to-date and valid. The main objectives were to minimize human effort and involvement in the effective management of the repository and also to support scalability. Experimental results and 3-year statistics presented in this chapter, showed that the approaches used in developing $\mathcal{DWDS}$ were effective in achieving both objectives.

In Chapter 5, intelligent mechanisms for inferring the functional semantics of services from their service descriptions, and automatically generating service-specific metadata for each indexed service were presented. Service similarity was computed using semantic relatedness between service features for enabling domain-specific categorization for the large service collection. Traditional clustering and machine learning classification were performed on a small sample of the tagged service collection to determine the effectiveness of the developed metadata generation and similarity computation techniques. Experimental evaluation confirmed that the tagging generated for each service was meaningful and similarity based domain-specific grouping helped in increasing the precision and recall performance during Web service retrieval.

It was observed that categorization approaches like traditional clustering and classification were not suitable due to the dynamic nature of $\mathcal{DWDS}$ and the constant small changes in the service collection after each active/selective crawling. Hence, a novel dynamic clustering algorithm, BI^2C, modelled as per bird flocking rules was designed and presented in Chapter 6. This eliminates the need for off-line cluster re-computation and generates updated clusters on the fly. Due to this, it is possible to dynamically cluster new services introduced thus reducing the effort and time required for the re-clustering process. During experimental evaluation, BI^2C was effective and achieved a speedup of more than 57%, when compared to traditional hierarchical clustering.

To understand the user context and requirements during the Web service discovery process, an intelligent mechanism using semantics and natural language processing techniques was presented in Chapter 7. The mechanisms designed for identifying whether a given query required composite service discovery were also described in this chapter. Experimental evaluation showed that the semantic query achieved more that 17% improvement in precision and 37% increase in recall over the natural language query. However, the result generation time in the case of the semantic query approach was an average of 26 seconds more than the keyword based matching approach.

In Chapter 8, a methodology for discovering composite service templates as per user's complex requirements was discussed. The approach is based on formal representation of captured service dependencies using a Service Interface Graph, which is traversed to determine constituent services and correct invocation sequence for a given complex query. To evaluate performance of this approach, several experiments using varied sized OWL-S datasets were used and the average precision observed was 79%. The overall accuracy of composite service template generation was found to be 70.68%.

9.1 Future Scope

Some possible future works are summarized as follows:

- Currently, the repository of the $\mathcal{DWDS}$ framework is centralized, possibly rendering it a single point of failure. We intend to mitigate this potential problem by adopting a cloud based deployment model, thus making $\mathcal{DWDS}$ truly scalable and distributed.

- Elaborate theoretical analysis of the proposed framework and its processes will be performed, to completely assess its suitability and scalability for real-world deployment.

- The $\mathcal{DWDS}$ framework uses a bottom-up approach of finding and retrieving distributed Web services, instead of relying on service providers to publish and maintain their services, as in other approaches like the UDDI and the UBR. In future, service providers may themselves be allowed access to their services indexed in the $\mathcal{DWDS}$ repository, to add their details or QoS statistics for their services. This will enable further enrichment of the $\mathcal{DWDS}$ service collection. Further, QoS based matching and selection can also be incorporated, if large-scale QoS statistics for all indexed services become available. Similarly, query serving can be improved by allowing QoS based conditions, as per user requirement, when such QoS statistics become available. This means that the user will not only be provided with the most relevant service for a given query, but also possibly the fastest or most reliable service.

- For enabling personalisation based service discovery, user feedback mechanisms could be incorporated in $\mathcal{DWDS}$ that allows automatic collection of user comments or ratings. These can be used to improve the ranking mechanisms, using techniques like sentiment analysis and collaborative filtering. Thus, QoE (Quality of Experience) based querying capabilities can also be added to $\mathcal{DWDS}$ functionalities.

- Incremental clustering can be further enhanced by incorporating dynamic iteration adjustment, instead of using predefined number of iterations, ideal cluster number prediction etc. Hybrid techniques that allow local optimization at cluster level may be investigated that can improve the goodness of clustering further, to reduce the effect of cluster overlap (caused by common natural language tags).

- In $\mathcal{DWDS}$, the time required to serve the semantic user query takes about 26 seconds more when compared to keyword based matching approach (natural language query). Adopting query and result caching mechanisms may help in reducing this time, by possibly reusing already generated templates while service similar requests at a later point of time. In this way, subsequent queries that are partly or fully similar can be served faster. Also, we intend to develop an intuitive and user-friendly user interface for $\mathcal{DWDS}$.

- Currently, composite service discovery has been verified using a standard semantic Web service description dataset, OWL-S TC. When bigger OWL-S datasets are available, the proposed technique can be tested and the results can be verified for ensuring scalability. Also, during composite service template generation, some generated templates displayed I/O data type conflicts, i.e., a service that can be chained with another service in order was found to be incompatible due to datatype mismatch. This is because, currently we have considered on the OWL-S profile's <profile:hasInput>, <profile:hasOutput> and <profile:textDescription>. This problem can be addressed so the accuracy can be further improved. Also, the querying process can be further enhanced by allowing users to specify preconditions and effects implicitly, in addition to the required outputs. Other factors like subquery level QoS and user specified constraints could also be potentially incorporated.

Publications based on Research Work

Journal Publications

1. Sowmya Kamath S and Ananthanarayana V.S, *"Discovering Composable Web Services based on Functional Semantics and Service Dependencies using Natural Language Requests"*, Journal on Information System Frontiers, Springer Hiedelberg, ISSN: 1387-3326 (SCI Indexed)

   Status: In press

2. Sowmya Kamath S and Ananthanarayana V.S, *"Semantic Similarity based Context-aware Web Service Discovery using NLP Techniques"*, Journal of Web Engineering (JWE), Rinton Press, Princeton, New Jersey, Volume 15, Issue 1 & 2, 2016. [ISSN: 1540-9589] (SCI Indexed)

   Status: Online.

3. Sowmya Kamath S and Ananthanarayana V.S, *"A Bio-inspired, Incremental Clustering Algorithm for Semantics-based Web Service Discovery"*, International Journal of Reasoning-based Intelligent Systems, Inderscience Publishers, Volume 7, Issue 3-4, pgs. 261-275, 2015. [ISSN: 1755-0564] (Scopus and EI Indexed)

   Status: Online.

4. Sowmya Kamath S and Ananthanarayana V.S, *"Semantics based Web Service Classification using Morphological Analysis and Ensemble Learning Techniques"*, International Journal on Data Science and Analytics, Springer Hiedelberg, ISSN: 2364-4168

   Status: Accepted

Conference Publications

1. Sowmya Kamath S., and Ananthanarayana V.S., *"Towards Semantic Web Services: An Empirical Evaluation of Service Ontology Generation Tools"*, 12th IEEE India Conference on Electronics, Energy, Environment, Communication, Computer, Control $E^3 - C^3$, (IEEE INDICON 2015), pp.1-6, 2015.

2. Sowmya Kamath S., and Ananthanarayana V.S., *"Change propagation based incremental data handling in a Web service discovery framework"* in Signal Processing and Information Technology (ISSPIT), 2014 IEEE International Symposium on, vol. 14, no., pp.474-479, 15-17 Dec. 2014.

3. Sowmya Kamath S., and Ananthanarayana V.S., *"A Service Crawler based Framework for Similarity based Web Service Discovery"*, at 11th India Conference on Emerging Trends and Innovation in Technology (IEEE INDICON 2014), 2014 Annual IEEE, pp. 1-6. IEEE, 2014.

4. Sowmya Kamath, S., and V. S. Ananthanarayana. *"Similarity analysis of service descriptions for efficient Web service discovery."* In Data Science and Advanced Analytics (DSAA), 2014 IEEE/ACM International Conference on, Shanghai, China., pp. 142-148. IEEE, 2014.

5. Sowmya Kamath S., and Ananthanarayana V. S. *"A bottom-up approach towards achieving semantic web services."*, In Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on, pp. 1317-1322. IEEE, 2013.

6. Sowmya Kamath S and Ananthanarayana V.S, *"Semantic Web Services Discovery, Selection and Composition Techniques"*, at the Third International Conference on Computer Science and Information Technology (CCSIT 2013), February 18 - 20th, 2013, Bangalore, India

7. Sowmya Kamath S and Prakash S. Raghavendra, *"Semantic Web - Applications, Challenges and Directions"*, at the 3rd International Conference in Information Technology and Business Intelligence (ITBI 2011), November 25 - 27th, 2011, Hyderabad, India

Bibliography

Aboud, Nour-Alhouda, Gabriela Arevalo, Jean-Remy Falleri, Marianne Huchard, Chouki Tibermacine, Christelle Urtado, and Sylvain Vauttier (2009). “Automated software component classification using concept lattices”. In: *Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on.* IEEE, pp. 21–30.

Abowd, Gregory, Anind Dey, Peter Brown, Nigel Davies, Mark Smith, and Pete Steggles (1999). “Towards a better understanding of context and context-awareness”. In: *Handheld and ubiquitous computing.* Springer, pp. 304–307.

AbuJarour, M., F. Naumann, and M. Craculeac (2010). “Collecting, annotating, and classifying public web services”. In: *On the Move towards Meaningful Internet Systems: OTM 2010.* Springer, pp. 256–272.

AbuJarour, Mohammed and Felix Naumann (2010). “Towards a diamond SOA operational model”. In: *Service-Oriented Computing and Applications (SOCA), 2010 IEEE Intl. Conf. on.* IEEE, pp. 1–4.

Agarwal, Sudhir, Siegfried Handschuh, and Steffen Staab (2004). “Annotation, composition and invocation of semantic web services”. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 2.1, pp. 31–48.

Akkiraju, R., R. Goodwin, P. Doshi, and S. Roeder (2003). “A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI”. In: *Workshop on Information Integration on the Web (IIWeb) in conjunction with 18th International Joint Conference on Artificial Intelligence*, pp. 87–92.

Akkiraju, Rama (2007). “Semantic Web Services”. In: *Semantic Web Services: Theory, Tools and Applications: Theory, Tools and Applications.* Ed. by Jorge Cardoso. IGI Global. ISBN: ISBN 978-1-59904-045-5.

Akkiraju, Rama, Biplav Srivastava, Anca Ivan, Richard Goodwin, and Tanveer Syeda-Mahmood (2006). “Semantic matching to achieve web service discovery and composition”. In: *E-Commerce Technology, 2006. The 8th IEEE International Conference on and Enterprise Computing, E-Commerce, and E-Services.* IEEE, pp. 70–70.

Almasri, E and Q H Mahmoud (2008). “Discovering web services in search engines”. In: *IEEE Internet Computing* 3, pp. 74–77.

Almasri, Eyhab and Q H Mahmoud (2007). “QoS-based discovery and ranking of web services”. In: *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*. IEEE, pp. 529–534.

Alonso, Gustavo, Fabio Casati, Harumi Kuno, and Vijay Machiraju (2004). *Web services*. Springer Verlag. ISBN: 3540440089.

Alrifai, Mohammad et al. (2012). “A hybrid approach for efficient Web service composition with end-to-end QoS constraints”. In: *ACM Transactions on the Web (TWEB)* 6.2, p. 7.

Alrifai, Mohammad, Dimitrios Skoutas, and Thomas Risse (2010). “Selecting skyline services for QoS-based web service composition”. In: *19th International Conference on World Wide Web*. ACM, pp. 11–20.

Atkinson, Colin, Philipp Bostan, Oliver Hummel, and Dietmar Stoll (2007). “A practical approach to web service discovery and retrieval”. In: *Web Services, 2007. ICWS 2007. IEEE International Conference on*. IEEE, pp. 241–248.

Austin, Daniel, Abbie Barbir, Christopher Ferris, and Sharad Garg (2002). *Web Services Architecture Requirements, W3C Working Draft, 11 October 2002*. http://www.w3.org/TR/2002/WD-wsa-reqs-20021011. accessed 12-10-2013.

Aversano, Lerina, Gerardo Canfora, and Anna Ciampi (2004). “An algorithm for web service discovery through their composition”. In: *Web Services, 2004. Proceedings. IEEE International Conference on*. IEEE, pp. 332–339.

Azmeh, Zeina, Marianne Huchard, Chouki Tibermacine, Christelle Urtado, and Sylvain Vauttier (2008). “Wspab: A tool for automatic classification & selection of web services using formal concept analysis”. In: *on Web Services, 2008. ECOWS’08. IEEE Sixth European Conference*. IEEE, pp. 31–40.

Bau III, David, Adam Bosworth, Gary S Burd, Roderick A Chavez, and Kyle W Marvin (2008a). *Annotation based development platform for stateful web services*. US Patent 7,437,710.

Bau III, David, Adam Bosworth, Gary Burd, Roderick Chavez, and Kyle Marvin (2008b). *Annotation based development platform for asynchronous web services*. US Patent 7,356,803.

Bellifemine, F, A Poggi, and G Rimassa (1999). “JADE – A FIPA-compliant agent framework”. In: *Proceedings of PAAM*. Vol. 9. 33. London, pp. 97–108.

Bellifemine, F., Agostino Poggi, and Gi Rimassa (2001). “Developing multi-agent systems with JADE”. In: *Intelligent Agents VII Agent Theories Architectures and Languages*. Springer, pp. 89–103.

Benatallah, Boualem, Marlon Dumas, M-C Fauvet, Fethi A Rabhi, and Quan Z Sheng (2002). “Overview of some patterns for architecting and managing composite web services”. In: *ACM SIGecom Exchanges* 3.3, pp. 9–16.

Benatallah, Boualem, Mohand-Said Hacid, Alain Leger, Christophe Rey, and Farouk Toumani (2005). “On automating web services discovery”. In: *The VLDB Journal* 14.1, pp. 84–96.

Benjamins, V Richard (2008). “Near-term prospects for semantic technologies”. In: *Intelligent Systems, IEEE* 23.1, pp. 76–88.

Berardi, Daniela, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella (2005). “Automatic service composition based on behavioral descriptions”. In: *International Journal of Cooperative Information Systems* 14.04, pp. 333–376.

Berners-Lee, Tim, James Hendler, and Ora Lassila (2001). “The Semantic Web. A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities”. In: *Scientific American* 284.5, pp. 1–5.

Birukou, Aliaksandr, Enrico Blanzieri, Vincenzo D’Andrea, Paolo Giorgini, and Natallia Kokash (2007). “Improving web service discovery with usage data”. In: *Software, IEEE* 24.6, pp. 47–54.

Bishop, Christopher M (1995). *Neural networks for pattern recognition.* Oxford university press.

Bosca, Alessio, Fulvio Corno, Giuseppe Valetto, and Roberta Maglione (2006). “On-the-fly construction of web services compositions from natural language requests”. In: *Journal of Software* 1.1, pp. 40–50.

Bosca, Alessio, Andrea Ferrato, Fulvio Corno, Ilenia Congiu, and Giuseppe Valetto (2005). “Composing Web services on the basis of natural language requests”. In: *ICWS 2005.* IEEE, pp. 817–818.

Breiman, Leo (2001). “Random forests”. In: *Machine learning* 45.1, pp. 5–32.

Brockmans, Saartje et al. (2009). “Service-Finder: First steps toward the realization of web service discovery at web scale”. In: *Camogli (Genova), Italy June 25th, 2009 Co-located with SEBD*, p. 73.

Broens, Tom, Stanislav Pokraev, Marten Van Sinderen, Johan Koolwaaij, and Patricia Dockhorn Costa (2004). “Context-aware, ontology-based service discovery”. In: *Ambient Intelligence.* Springer, pp. 72–83.

Brogi, Antonio, Sara Corfini, Jose F Aldana, and Ismael Navas (2006). “Automated discovery of compositions of services described with separate ontologies”. In: *Service-Oriented Computing–ICSOC 2006.* Springer, pp. 509–514.

Brogi, Antonio, Sara Corfini, and Razvan Popescu (2005). “Composition-oriented service discovery”. In: *Software Composition.* Springer, pp. 15–30.

— (2008). “Semantics-based composition-oriented discovery of web services”. In: *ACM Transactions on Internet Technology (TOIT)* 8.4, p. 19.

Bruno, Marcello, Gerardo Canfora, Massimiliano Di Penta, and Rita Scognamiglio (2005). “An approach to support web service classification and annotation”. In: *e-Technology, e-Commerce and e-Service, 2005. EEE’05. Proceedings. The 2005 IEEE International Conference on.* IEEE, pp. 138–143.

Burstein, Mark et al. (2004). “OWL-S: Semantic markup for web services”. In: *W3C Member Submission.*

Can, Fazli (1993). “Incremental clustering for dynamic information processing”. In: *ACM Transactions on Information Systems (TOIS)* 11.2, pp. 143–164.

Cao, Jie, Zhiang Wu, Youquan Wang, and Yi Zhuang (2013). “Hybrid Collaborative Filtering algorithm for bidirectional Web service recommendation”. In: *Knowledge and information systems* 36.3, pp. 607–627.

Cardoso, Jorge and Amit P Sheth (2006). *Semantic web services, processes and applications.* Vol. 3. Springer Science & Business Media.

Castro, Miguel, Peter Druschel, Anne-Marie Kermarrec, and Antony IT Rowstron (2002). “SCRIBE: A large-scale and decentralized application-level multicast infrastructure”. In: *Selected Areas in Communications, IEEE Journal on* 20.8, pp. 1489–1499.

Chakrabarti, Soumen, Martin Van den Berg, and Byron Dom (1999). “Focused crawling: a new approach to topic-specific Web resource discovery”. In: *Computer Networks* 31.11, pp. 1623–1640.

Chan, Nguyen Ngoc, Walid Gaaloul, and Samir Tata (2012). “A recommender system based on historical usage data for web service discovery”. In: *Service Oriented Computing and Applications* 6.1, pp. 51–63.

Chan, Nguyen, Walid Gaaloul, and Samir Tata (2011). “A web service recommender system using vector space model and latent semantic indexing”. In: *Advanced Information Networking and Applications (AINA), 2011 IEEE International Conference on.* IEEE, pp. 602–609.

Chao, Kuo-Ming, Muhammad Younas, Chi-Chun Lo, and Tao-Hsin Tan (2005). “Fuzzy matchmaking for web services”. In: *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on.* Vol. 2. IEEE, pp. 721–726.

Chen, Liang, Yilun Wang, Qi Yu, Zibin Zheng, and Jian Wu (2013a). “WT-LDA: user tagging augmented LDA for web service clustering”. In: *Service-Oriented Computing.* Springer, pp. 162–176.

Chen, Lin et al. (2013b). “Automatic web services classification based on rough set theory”. In: *Journal of Central South University* 20, pp. 2708–2714.

Chen, Xi et al. (2010). “Regionknn: A scalable hybrid collaborative filtering algorithm for personalized web service recommendation”. In: *Web Services (ICWS), 2010 IEEE International Conference on.* IEEE, pp. 9–16.

Cheng, Yu, Alberto Leon-Garcia, and Ian Foster (2008). “Toward an autonomic service management framework: A holistic vision of SOA, AON, and autonomic computing”. In: *Communications Magazine, IEEE* 46.5, pp. 138–146.

Chifu, Viorica Rozina, Cristina Bianca Pop, Ioan Salomie, Mihaela Dinsoreanu, Vlad Acretoaie, and Tudor David (2010). “An ant-inspired approach for semantic web service clustering”. In: *Roedunet International Conference (RoEduNet), 2010 9th.* IEEE, pp. 145–150.

Cilibrasi, Rudi and Paul Vitanyi (2007). “The google similarity distance”. In: *Knowledge and Data Engineering, IEEE Transactions on* 19.3, pp. 370–383.

Corella, MA and Pablo Castells (2006a). “Taxonomy-Based Web service categorization using conceptual parameter descriptions”. In: *Proc. of the International Workshop on Semantic Matchmaking and Resource Retrieval: Issues and Perspectives (SMR 2006) at the 32nd International Conf. on Very Large Data Bases (VLDB 2006). Seoul: Morgan Kaufmann Publishers.* Citeseer.

Corella, Miguel Angel and Pablo Castells (2006b). “A heuristic approach to semantic web services classification”. In: *Knowledge-Based Intelligent Information and Engineering Systems.* Springer, pp. 598–605.

— (2006c). “Semi-automatic semantic-based web service classification”. In: *Business Process Management Workshops.* Springer, pp. 459–470.

Crasso, Marco, Alejandro Zunino, and Marcelo Campo (2008a). “AWSC: An approach to Web service classification based on machine learning techniques.” In: *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial* 12.37, pp. 25–36.

— (2008b). “Easy web service discovery: A query-by-example approach”. In: *Science of Computer Programming* 71.2, pp. 144–164.

Cremene, Marcel, Jean-Yves Tigli, Stephane Lavirotte, Florin-Claudiu Pop, Michel Riveill, and Gaetan Rey (2009). “Service composition based on natural language requests”. In: *Services Computing, 2009. SCC’09. IEEE International Conference on.* IEEE, pp. 486–489.

Curbera, Francisco, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana (2002). “Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI”. In: *IEEE Internet computing* 2, pp. 86–93.

Davies, David L and Donald W Bouldin (1979). “A cluster separation measure”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 2, pp. 224–227.

De Bruijn, Jos et al. (2005). “Web service modeling ontology (wsmo)”. In: *Interface* 5, p. 1.

Della Valle, Emanuele, Dario Cerizza, Irene Celino, Andrea Turati, Holger Lausen, Nathalie Steinmetz, Michael Erdmann, and Adam Funk (2008). “Realizing Service-Finder: Web service discovery at web scale”. In: *European Semantic Technology Conference (ESTC), Vienna.*

Devis, Bianchini, De Antonellis Valeria, and Melchiori Michele (2008). “Flexible semantic-based service matchmaking and discovery”. In: *World Wide Web* 11.2, pp. 227–251.

D’Mello, Demian et al. (2008). “A QoS broker based architecture for dynamic web service selection”. In: *Modeling & Simulation, 2008. AICMS 08. Second Asia International Conference on.* IEEE, pp. 101–106.

D’Mello, Demian and VS Ananthanarayana (2009). “Effective Web Service Discovery Based on Functional Semantics”. In: *Advances in Computing, Control, & Telecommunication Technologies, 2009. ACT’09. International Conference on.* IEEE, pp. 1–3.

Dong, Xin, Alon Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang (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.

Doulkeridis, Christos, Nikos Loutas, and Michalis Vazirgiannis (2006). “A system architecture for context-aware service discovery”. In: *Electronic Notes in Theoretical Computer Science* 146.1, pp. 101–116.

Dunn, J. (1973). “A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters”. In:

El Bouhissi, Houda, Mimoun Malki, and Amine Sidi Ali Cherif (2014a). “From User’s Goal to Semantic Web Services Discovery: Approach Based on Traceability”. In: *International Journal of Information Technology and Web Engineering (IJITWE)* 9.3, pp. 15–39.

— (2014b). “Improve Web Service discovery: Goal-based approach”. In: *Computer Systems and Applications (AICCSA), 2014 IEEE/ACS 11th International Conference on.* IEEE, pp. 26–33.

Elgazzar, Khalid, Ahmed E Hassan, and Patrick Martin (2010). “Clustering wsdl documents to bootstrap the discovery of web services”. In: *Web Services (ICWS), 2010 IEEE International Conference on.* IEEE, pp. 147–154.

Ertek, Gurdal et al. (2013). “Text mining with rapidminer”. In: *RapidMiner: Data Mining Use Cases and Business Analytics Applications*, p. 241.

Ester, Martin, Hans-Peter Kriegel, Jorg Sander, Michael Wimmer, and Xiaowei Xu (1998). “Incremental clustering for mining in a data warehousing environment”. In: *VLDB*. Vol. 98. Citeseer, pp. 323–333.

Fan, Jianchun and Subbarao Kambhampati (2005). “A snapshot of public web services”. In: *ACM SIGMOD Record* 34.1, pp. 24–32.

Fang, Lu, Lijie Wang, Meng Li, Junfeng Zhao, Yanzhen Zou, and Lingshuang Shao (2012). “Towards automatic tagging for web services”. In: *Web Services (ICWS), 2012 IEEE 19th International Conference on*. IEEE, pp. 528–535.

Farrag, Tamer Ahmed, Ahmed Ibrahim Saleh, and Hesham Arafat Ali (2013). “Toward SWSs discovery: mapping from wsdl to owl-s based on ontology search and standardization engine”. In: *Knowledge and Data Engineering, IEEE Transactions on* 25.5, pp. 1135–1147.

Fenza, Giuseppe, Vincenzo Loia, and Sabrina Senatore (2008). “A hybrid approach to semantic web services matchmaking”. In: *International Journal of Approximate Reasoning* 48.3, pp. 808–828.

Fenza, Giuseppe and Sabrina Senatore (2010). “Friendly web services selection exploiting fuzzy formal concept analysis”. In: *Soft Computing* 14.8, pp. 811–819.

Fielding, Roy Thomas (2000). “Architectural styles and the design of network-based software architectures”. PhD thesis. University of California, Irvine.

Finkel, Raphael and Jon-Louis Bentley (1974). “Quad trees a data structure for retrieval on composite keys”. In: *Acta informatica* 4.1, pp. 1–9.

Freiderici, Peter (2009). *Explaining Bird Flocks*. Audobon Magazine, March-April 2009.

Frické, Martin (1998). “Measuring recall”. In: *Journal of Information Science* 24.6, pp. 409–417.

Gao, Yan, Bin Zhang, Jun Na, Lei Yang, Yu Dai, and Qiang Gong (2005). “Optimal selection of web services for composition based on interface-matching and weighted multistage graph”. In: *Parallel and Distributed Computing, Applications and Technologies, 2005. PDCAT 2005. Sixth International Conference on*. IEEE, pp. 336–338.

Garcia-Sanchez, Francisco, Rafael Valencia-Garcia, Rodrigo Martinez-Bejar, and Jesualdo T Fernandez-Breis (2009). “An ontology, intelligent agent-based framework for the provision of semantic web services”. In: *Expert Systems with Applications* 36.2, pp. 3167–3187.

Giantsiou, Lemonia, Nikolaos Loutas, Vassilios Peristeras, and Konstantinos Tarabanis (2009). *Semantic Service Search Engine (S3E): An approach for finding services on the Web*. Springer.

Gottschalk, Karl, Stephen Graham, Heather Kreger, and James Snell (2002). “Introduction to web services architecture”. In: *IBM systems Journal* 41.2, pp. 170–177.

Grigori, Daniela, Juan Carlos Corrales, and Mokrane Bouzeghoub (2006). “Behavioral matchmaking for service retrieval”. In: *Web Services, 2006. ICWS’06. International Conference on.* IEEE, pp. 145–152.

Grossi, Davide, Frank Dignum, John-Jules Ch Meyer, et al. (2004). “Contextual taxonomies”. In: *CLIMA*. Springer, pp. 33–51.

Guinard, Dominique, Vlad Trifa, Patrik Spiess, Bettina Dober, and Stamatis Karnouskos (2009). “Discovery and on-demand provisioning of real-world web services”. In: *Web Services, 2009. ICWS 2009. IEEE International Conference on.* IEEE, pp. 583–590.

Han, Liangxiu and Dave Berry (2008). “Semantics-supported and agent-based decentralized grid resource discovery”. In: *Future Generation Computer Systems* 24.8, pp. 806–812.

Hao, Yanan and Yanchun Zhang (2007). “Web services discovery based on schema matching”. In: *Proceedings of the thirtieth Australasian conference on Computer science-Volume 62.* Australian Computer Society, Inc., pp. 107–113.

Hashemian, Seyyed and Farhad Mavaddat (2006). “A graph-based framework for composition of stateless web services”. In: *Web Services, 2006. ECOWS’06. 4th European Conference on.* IEEE, pp. 75–86.

Hastie, Tibshirani and Friedman (2009). “Hierarchical Clustering, Elements of Statistical Learning (2nd ed)”. In:

Hendler, James (2001). “Agents and the semantic web”. In: *IEEE Intelligent systems* 2, pp. 30–37.

Heß, Andreas and Nicholas Kushmerick (2003). “Learning to attach semantic metadata to web services”. In: *The Semantic Web-ISWC 2003.* Springer, pp. 258–273.

— (2004). “Assam: A tool for semi-automatically annotating semantic web services”. In: *The Semantic Web–ISWC 2004.* Springer, pp. 320–334.

Hofreiter, Birgit, Christian Huemer, and Wolfgang Klas (2002). “ebXML: Status, research issues, and obstacles”. In: *Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems, 2002. RIDE-2EC 2002. Proceedings. Twelfth International Workshop on.* IEEE, pp. 7–16.

Hong, Jongyi, Eui-Ho Suh, Junyoung Kim, and SuYeon Kim (2009). “Context-aware system for proactive personalized service based on context history”. In: *Expert Systems with Applications* 36.4, pp. 7448–7457.

Hosmer, David W and Stanley Lemeshow (2000). “Introduction to the logistic regression model”. In: *Applied Logistic Regression, Second Edition*, pp. 1–30.

Issarny, Valerie, Nikolaos Georgantas, Sara Hachem, Apostolos Zarras, Panos Vassiliadist, Marco Autili, Marco Aurelio Gerosa, and Amira Ben Hamida (2011). “Service-oriented middleware for the future internet: state of the art and research directions”. In: *Journal of Internet Services and Applications* 2.1, pp. 23–45.

Jaccard, Paul (1912). “The distribution of the flora in the alpine zone. 1”. In: *New phytologist* 11.2, pp. 37–50.

Jain, Anil K (2009). “Data clustering: 50 years beyond K-means”. In: *Pattern recognition letters* 31.8, pp. 651–666.

Jiang, Jay and David Conrath (1997a). “Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy”. In: *CoRR* cmp-lg/9709008.

— (1997b). “Semantic similarity based on corpus statistics and lexical taxonomy”. In: *Proceedings of ROCLING X, Taiwan.*

Katakis, Ioannis, Georgios Meditskos, Grigorios Tsoumakas, Nick Bassiliades, et al. (2009). “On the combination of textual and semantic descriptions for automated semantic web service classification”. In: *Artificial Intelligence Applications and Innovations III.* Springer, pp. 95–104.

Kehagias, Dionysios, Konstantinos Giannoutakis, George Gravvanis, and Dimitrios Tzovaras (2012). “An ontology-based mechanism for automatic categorization of web services”. In: *Concurrency and Computation: Practice and Experience* 24.3, pp. 214–236.

Keller, Uwe, Ruben Lara, Holger Lausen, and Dieter Fensel (2006). “Semantic Web service discovery in the WSMO framework”. In: *Semantic Web: Theory, Tools and Applications. Idea Publishing Group.*

Kim, Su Myeon and Marcel-Catalin Rosu (2004). “A survey of public web services”. In: *E-commerce and web technologies.* Springer, pp. 96–105.

Klema, Virginia C and Alan J Laub (1980). “The singular value decomposition: Its computation and some applications”. In: *Automatic Control, IEEE Transactions on* 25.2, pp. 164–176.

Klusch, M (2012). *Intelligent information agents: agent-based information discovery and management on the Internet.* Springer Science & Business Media.

Klusch, Matthias, Benedikt Fries, and Katia Sycara (2006). “Automated semantic web service discovery with OWLS-MX”. In: *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems.* ACM, pp. 915–922.

Kohavi, Ron (1995). “A study of cross-validation and bootstrap for accuracy estimation and model selection”. In: *International Joint Conference on Artificial Intelligence (IJCAI 1995).* Vol. 14. 2, pp. 1137–1145.

Kokash, Natallia, Aliaksandr Birukou, and Vincenzo D'Andrea (2007). "Web service discovery based on past user experience". In: *Business Information Systems*. Springer, pp. 95–107.

Kopecky, Jacek, Tomas Vitvar, Carine Bournez, and Joel Farrell (2007). "Sawsdl: Semantic annotations for wsdl and xml schema". In: *Internet Computing, IEEE* 11.6, pp. 60–67.

Kourtesis, Dimitrios and Iraklis Paraskakis (2008). "Web service discovery in the FUSION semantic registry". In: *Business Information Systems*. Springer, pp. 285–296.

Krill, Peter (2005). *Microsoft, IBM, SAP discontinue UDDI registry effort*. Available from http://www.infoworld.com, December 2005.

Kritikos, Kyriakos and Dimitris Plexousakis (2009). "Requirements for QoS-based web service description and discovery". In: *Services Computing, IEEE Transactions on* 2.4, pp. 320–337.

Krummenacher, Reto, Martin Hepp, Axel Polleres, Christoph Bussler, and Dieter Fensel (2005). "WWW or What is Wrong with Web services". In: *Web Services, 2005. ECOWS 2005. Third IEEE European Conference on*. IEEE, 9–pp.

Kuang, Li et al. (2012). "Personalized services recommendation based on context-aware QoS prediction". In: *Web Services (ICWS), 2012 IEEE 19th International Conference on*. IEEE, pp. 400–406.

Kuang, Li, Ying Li, Shuiguang Deng, and Zhaohui Wu (2007). "Inverted indexing for composition-oriented service discovery". In: *Web Services, 2007. ICWS 2007. IEEE International Conference on*. IEEE, pp. 257–264.

Kumar, Ravi, Prabhakar Raghavan, Sridhar Rajagopalan, D Sivakumar, Andrew Tomkins, and Eli Upfal (2000). "Stochastic models for the web graph". In: *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*. IEEE, pp. 57–65.

Kumara, Banage TGS, Incheon Paik, and Wuhui Chen (2013). "Web-service clustering with a hybrid of ontology learning and information-retrieval-based term similarity". In: *Web Services (ICWS), 2013 IEEE 20th International Conference on*. IEEE, pp. 340–347.

Lara, Ruben, Dumitru Roman, Axel Polleres, and Dieter Fensel (2004). "A conceptual comparison of WSMO and OWL-S". In: *Web services*. Springer, pp. 254–269.

Laranjeiro, Nuno, Rui Oliveira, and Marco Vieira (2010). "Applying text classification algorithms in web services robustness testing". In: *Reliable Distributed Systems, 2010 29th IEEE Symposium on*. IEEE, pp. 255–264.

Li, Xia, Fang Qian, Chong Li, and Jianjun Yu (2011). "Similar Web Services Discovery and Matching Based on P2P and Topic Model Learning". In: *Wireless Communications,* *Networking and Mobile Computing (WiCOM), 2011 7th International Conference on.* IEEE, pp. 1–4.

Li, Yan, Jinhua Xiong, Xinran Liu, Hong Zhang, and Peng Zhang (2014). “Folksonomy-Based In-Depth Annotation of Web Services”. In: *Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on.* IEEE, pp. 243–249.

Li, Yan, Liangjie Zhang, Ge Li, Bing Xie, and Jiasu Sun (2007). “An exploratory study of web services on the internet”. In: *Web Services, 2007. ICWS 2007. IEEE International Conference on.* IEEE, pp. 380–387.

Liang, Qianhui Althea and Stanley YW Su (2005). “AND/OR graph and search algorithm for discovering composite web services”. In: *International Journal of Web Services Research (IJWSR)* 2.4, pp. 48–67.

Lim, JongHyun and Kyong-Ho Lee (2010). “Constructing composite web services from natural language requests”. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 8.1, pp. 1–13.

Lin, Maria and David W Cheung (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.* IEEE Computer Society, pp. 258–265.

Liu, Chenguang et al. (2013). “A Web Service Recommendation Approach Based on Situation Awareness”. In: *Services Computing (SCC), 2013 IEEE International Conference on.* IEEE, pp. 432–437.

Liu, Yanchi, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu (2010). “Understanding of internal clustering validation measures”. In: *Data Mining (ICDM), 2010 IEEE 10th International Conference on.* IEEE, pp. 911–916.

Liu, Zhen, Anand Ranganathan, and Anton Riabov (2007). “Modeling Web services using semantic graph transformations to aid automatic composition”. In: *Web Services, 2007. ICWS 2007. IEEE International Conference on.* IEEE, pp. 78–85.

Liu and Wong (2009). “Web service clustering using text mining techniques”. In: *International Journal of Agent-Oriented Software Engineering* 3.1, pp. 6–26.

Lopes, Cassio and Ali Sayed (2007). “Incremental adaptive strategies over distributed networks”. In: *Signal Processing, IEEE Transactions on* 55.8, pp. 4064–4077.

Ma, Jiangang et al. (2007). “A probabilistic semantic approach for discovering web services”. In: *16th international conference on World Wide Web.* ACM, pp. 1221–1222.

Ma, Jiangang, Yanchun Zhang, and Jing He (2008). “Efficiently finding web services using a clustering semantic approach”. In: *Proceedings of the 2008 international workshop* *on Context enabled source and service selection, integration and adaptation: organized with the 17th International World Wide Web Conference (WWW 2008)*. ACM, p. 5.

Maamar, Zakaria, Pedro Bispo Santos, Leandro Krug Wives, Youakim Badr, Noura Faci, and Jose Palazzo M De Oliveira (2011). “Using social networks for web services discovery”. In: *Internet Computing, IEEE* 15.4, pp. 48–54.

Makhoul, John, Francis Kubala, Richard Schwartz, Ralph Weischedel, et al. (1999). “Performance measures for information extraction”. In: *Proceedings of DARPA broadcast news workshop*, pp. 249–252.

Manning, Christopher D, Prabhakar Raghavan, Hinrich Schutze, et al. (2008). *Introduction to information retrieval*. Vol. 1. Cambridge university press Cambridge.

Martin, Daniel and John Domingue (2007). “Semantic web services, part 1”. In: *Intelligent Systems, IEEE* 22.5, pp. 12–17.

Al-Masri, Eyhab et al. (2009). “A Broker for Universal Access to Web Services”. In: *Seventh Annual Communication Networks and Services Research Conf.* IEEE.

Al-Masri, Eyhab and Qusay H Mahmoud (2007a). “A framework for efficient discovery of web services across heterogeneous registries”. In: *2007 4th IEEE Consumer Communications and Networking Conference.*

— (2007b). “Crawling multiple UDDI business registries”. In: *Proceedings of the 16th international conference on World Wide Web*. ACM, pp. 1255–1256.

— (2008). “Investigating web services on the world wide web”. In: *Proceedings of the 17th international conference on World Wide Web*. ACM, pp. 795–804.

Al-masri, Eyhab and Qusay H Mahmoud (2009). “A broker for universal access to web services”. In: *Communication Networks and Services Research Conference, 2009. CNSR’09. Seventh Annual*. IEEE, pp. 118–125.

McIlraith, Sheila A, Tran Cao Son, and Honglei Zeng (2001). “Semantic web services”. In: *IEEE intelligent systems* 16.2, pp. 46–53.

Michlmayr, Anton, Florian Rosenberg, Christian Platzer, Martin Treiber, and Schahram Dustdar (2007). “Towards recovering the broken SOA triangle: a software engineering perspective”. In: *International workshop on Service oriented Software Engineering: in conjunction with the 6th ESEC/FSE Joint Meeting*. ACM, pp. 22–28.

Miller, George, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller (1990). “Introduction to wordnet: An on-line lexical database*”. In: *International Journal of Lexicography* 3.4, pp. 235–244.

Milojicic, Dejan S, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu (2002). *Peer-to-peer computing.*

Miltsakaki, Eleni, Rashmi Prasad, Aravind K Joshi, and Bonnie L Webber (2004). “The Penn Discourse Treebank”. In: *LREC*.

Mohebbi, Keyvan, Suhaimi Ibrahim, and Mazdak Zamani (2013). “A Pre-matching Filter to Improve the Query Response Time of Semantic Web Service Discovery”. In: *Journal of Next Generation Information Technology* 4.6.

Mokhtar, Sonia Ben, Davy Preuveneers, Nikolaos Georgantas, Valerie Issarny, and Yolande Berbers (2008). “EASY: Efficient semAntic Service discoverY in pervasive computing environments with QoS and context support”. In: *Journal of Systems and Software* 81.5, pp. 785–808.

Al-Muhammed, Muhammed J and David W Embley (2006). “Resolving underconstrained and overconstrained systems of conjunctive constraints for service requests”. In: *Advanced Information Systems Engineering*. Springer, pp. 223–238.

Murphy, Kevin P (2006). “Naive bayes classifiers”. In: *University of British Columbia*.

Namgoong, Hyun (2006). “Effective semantic Web services discovery using usability”. In: *Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference*. IEEE, pp. 5–11.

Nayak, Richi and Brian Lee (2007). “Web service discovery with additional semantics and clustering”. In: *IEEE/WIC/ACM International Conference on Web Intelligence*. IEEE, pp. 555–558.

Newman, Mark EJ (2005). “Power laws, Pareto distributions and Zipf’s law”. In: *Contemporary physics* 46.5, pp. 323–351.

O’Brien, Patrick D and Richard C Nicol (1998). “FIPA—-towards a standard for software agents”. In: *BT Technology Journal* 16.3, pp. 51–59.

Oldham, Nicole, Christopher Thomas, Amit Sheth, and Kunal Verma (2005). “Meteor-s web service annotation framework with machine learning classification”. In: *Semantic Web services and Web process composition*. Springer, pp. 137–146.

Paliwal, Aabhas et al. (2006). “Web service discovery via semantic association ranking and hyperclique pattern discovery”. In: *2006 IEEE/WIC/ACM International Conference on Web Intelligence*. IEEE Computer Society, pp. 649–652.

— (2007). “Web service discovery: Adding semantics through service request expansion and latent semantic indexing”. In: *Services Computing, 2007. SCC 2007. IEEE International Conference on*. IEEE, pp. 106–113.

— (2012). “Semantics-based automated service discovery”. In: *Services Computing, IEEE Transactions on* 5.2, pp. 260–275.

Pant, Gautam, Padmini Srinivasan, and Filippo Menczer (2004). “Crawling the web”. In: *Web Dynamics*. Springer, pp. 153–177.

Paolucci, Massimo, Takahiro Kawamura, Terry R Payne, and Katia Sycara (2002). “Semantic matching of web services capabilities”. In: *The Semantic Web—ISWC 2002*. Springer, pp. 333–347.

Papazoglou, Mike P (2003). “Service-oriented computing: Concepts, characteristics and directions”. In: *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on.* IEEE, pp. 3–12.

Pawar, Pravin and Andrew Tokmakoff (2006). “Ontology-based context-aware service discovery for pervasive environments”. In: pp. 1–7.

Pedersen, Ted, Siddharth Patwardhan, and Jason Michelizzi (2004). “WordNet::Similarity: measuring the relatedness of concepts”. In: *Demonstration papers at HLT-NAACL 2004.* Association for Computational Linguistics, pp. 38–41.

Platzer, Christian and Schahram Dustdar (2005). “A vector space search engine for web services”. In: *Web Services, 2005. ECOWS 2005. Third IEEE European Conference on.* IEEE, 9–pp.

Platzer, Christian, Florian Rosenberg, and Schahram Dustdar (2009). “Web service clustering using multidimensional angles as proximity measures”. In: *ACM Transactions on Internet Technology (TOIT)* 9.3, p. 11.

Plebani, Pierluigi and Barbara Pernici (2009). “URBE: Web service retrieval based on similarity evaluation”. In: *Knowledge and Data Engineering, IEEE Transactions on* 21.11, pp. 1629–1642.

Pop, Cristina Bianca, Viorica Rozina Chifu, Ioan Salomie, Mihaela Dinsoreanu, Tudor David, and Vlad Acretoaie (2010). “Semantic web service clustering for efficient discovery using an ant-based method”. In: *Intelligent Distributed Computing IV.* Springer, pp. 23–33.

Quarteroni, Silvia et al. (2012). “Evaluating Multi-focus Natural Language Queries over Data Services.” In: *LREC*, pp. 2547–2552.

— (2013). “A bottom-up, knowledge-aware approach to integrating and querying web data services”. In: *ACM Transactions on the Web (TWEB)* 7.4, p. 19.

Rajasekaran, Preeda, John Miller, Kunal Verma, and Amit Sheth (2005). “Enhancing web services description and discovery to facilitate composition”. In: *Semantic Web Services and Web Process Composition.* Springer, pp. 55–68.

Ran, Shuping (2003). “A model for web services discovery with QoS”. In: *ACM Sigecom exchanges* 4.1, pp. 1–10.

Rasch, Katharina, Fei Li, Sanjin Sehic, Rassul Ayani, and Schahram Dustdar (2011). “Context-driven personalized service discovery in pervasive environments”. In: *World Wide Web* 14.4, pp. 295–319.

Real, Raimundo and Juan Vargas (1996). “The probabilistic basis of Jaccard’s index of similarity”. In: *Systematic biology*, pp. 380–385.

Reynolds, Craig W. (1987). “Flocks, Herds and Schools: A Distributed Behavioral Model”. In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’87. ACM, pp. 25–34.

Sabou, Marta, Chris Wroe, Carole Goble, and Gilad Mishne (2005). “Learning domain ontologies for web service descriptions: an experiment in bioinformatics”. In: *Proceedings of the 14th international conference on World Wide Web*. ACM, pp. 190–198.

Saha, Suman, CA Murthy, and Sankar K Pal (2008). “Classification of web services using tensor space model and rough ensemble classifier”. In: *Foundations of Intelligent Systems*. Springer, pp. 508–513.

Sajjanhar, Atul, Jingyu Hou, and Yanchun Zhang (2004). “Algorithm for web services matching”. In: *Advanced Web Technologies and Applications*. Springer, pp. 665–670.

Salton, Gerard, Anita Wong, and Chung-Shu Yang (1975). “A vector space model for automatic indexing”. In: *Communications of the ACM* 18.11, pp. 613–620.

Sangers, Jordy, Flavius Frasincar, Frederik Hogenboom, and Vadim Chepegin (2013). “Semantic Web service discovery using natural language processing techniques”. In: *Expert Systems with Applications* 40.11, pp. 4660–4671.

Schmidt, Cristina and Manish Parashar (2004). “A peer-to-peer approach to web service discovery”. In: *World Wide Web* 7.2, pp. 211–229.

Schulte, Stefan, Ulrich Lampe, Julian Eckert, and Ralf Steinmetz (2010). “LOG4SWS. KOM: self-adapting semantic web service discovery for SAWSDL”. In: *Services (SERVICES-1), 2010 6th World Congress on*. IEEE, pp. 511–518.

Segev, Aviv and Quan Z Sheng (2012). “Bootstrapping ontologies for web services”. In: *Services Computing, IEEE Transactions on* 5.1, pp. 33–44.

Serhani, M Adel, Rachida Dssouli, Abdelhakim Hafid, and Houari Sahraoui (2005). “A QoS broker based architecture for efficient web services selection”. In: *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, pp. 113–120.

ShaikhAli, Ali, Omer F Rana, Rashid Al-Ali, and David W Walker (2003). “Uddie: An extended registry for web services”. In: *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on*. IEEE, pp. 85–89.

Shao, Lingshuang, Jing Zhang, Yong Wei, Junfeng Zhao, Bing Xie, and Hong Mei (2007). “Personalized QoS prediction forweb services via collaborative filtering”. In: *Web Services, 2007. ICWS 2007. IEEE International Conference on*. IEEE, pp. 439–446.

Sheshagiri, Mithun, Norman Sadeh, and Fabien Gandon (2004). “Using semantic web services for context-aware mobile applications”. In: *MobiSys 2004 Workshop on Context Awareness*.

Sheth, Amit, Kunal Verma, and Karthik Gomadam (2006). “Semantics to energize the full services spectrum”. In: *Communications of the ACM* 49.7, pp. 55–61.

Shima, Hideki (2013). *WS4J-WordNet Similarity for Java*. Available from https://code.google.com/p/ws4j/.

Shin, Donghoon et al. (2009). “Automated generation of composite web services based on functional semantics”. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 7.4, pp. 332–343.

Shin, Dong-Hoon and Kyong-Ho Lee (2007). “An automated composition of information web services based on functional semantics”. In: *Services, 2007 IEEE Congress on*. IEEE, pp. 300–307.

Si, Huayou, Zhong Chen, Yong Deng, and Lian Yu (2013). “Semantic web services publication and OCT-based discovery in structured P2P network”. In: *Service Oriented Computing and Applications* 7.3, pp. 169–180.

Sioutas, Spyros, Evangelos Sakkopoulos, Ch Makris, Bill Vassiliadis, A Tsakalidis, and Peter Triantafillou (2009). “Dynamic Web Service discovery architecture based on a novel peer based overlay network”. In: *Journal of Systems and Software* 82.5, pp. 809–824.

Sivashanmugam, Kaarthik, Kunal Verma, and Amit Sheth (2004). “Discovery of web services in a federated registry environment”. In: *Web Services, 2004. Proceedings. IEEE International Conference on*. IEEE, pp. 270–278.

Skoutas, D. et al. (2010a). “Ranking and clustering web services using multi-criteria dominance relationships”. In: *Services Computing, IEEE Transactions on* 3.3, pp. 163–177.

Skoutas, Dimitrios et al. (2008). *Efficient semantic web service discovery in centralized and p2p environments*. Springer.

Skoutas, Dimitrios, Mohammad Alrifai, and Wolfgang Nejdl (2010b). “Re-ranking web service search results under diverse user preferences”. In: *VLDB, Workshop on Personalized Access, Profile Management, and Context Awareness in Databases*, pp. 898–909.

Skurichina, Marina and Robert PW Duin (2002). “Bagging, boosting and the random subspace method for linear classifiers”. In: *Pattern Analysis & Applications* 5.2, pp. 121–135.

Song, Henry, Doreen Cheng, Alan Messer, and Swaroop Kalasapur (2007). "Web service discovery using general-purpose search engines". In: *Web Services, 2007. ICWS 2007. IEEE International Conference on.* IEEE, pp. 265–271.

Spanoudakis, George, Khaled Mahbub, and Andrea Zisman (2007). "A Platform for Context Aware Runtime Web Service Discovery." In: *ICWS*, pp. 233–240.

Srinivasan, Naveen, Massimo Paolucci, and Katia Sycara (2006). "Semantic web service discovery in the OWL-S IDE". In: *System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on.* Vol. 6. IEEE, 109b–109b.

Steinmetz, Nathalie et al. (2009). "Web service search on large scale". In: *Service-Oriented Computing.* Springer, pp. 437–444.

Steinmetz, Nathalie, Mick Kerrigan, Holger Lausen, Martin Tanler, and Adina Sirbu (2008). "Simplifying the Web Service Discovery Process." In: *SeMMA*, pp. 31–45.

Stoica, I, R Morris, D Liben-Nowell, DR Karger, MF Kaashoek, F Dabek, and H Balakrishnan (2001). "Chord: A scalable P2P lookup protocol for Internet applications". In: *Proc. of ACM SIGCOMM.*

Stoica, Ion, Robert Morris, David Liben-Nowell, David R Karger, M Frans Kaashoek, Frank Dabek, and Hari Balakrishnan (2003). "Chord: a scalable peer-to-peer lookup protocol for internet applications". In: *Networking, IEEE/ACM Transactions on* 11.1, pp. 17–32.

Stollberg, Michael, Martin Hepp, and Jorg Hoffmann (2007). *A caching mechanism for semantic web service discovery.* Vol. 4825. Lecture Notes in Computer Science. Springer.

Stroulia, Eleni and Yiqiao Wang (2005). "Structural and semantic matching for assessing web-service similarity". In: *International Journal of Cooperative Information Systems* 14.04, pp. 407–437.

Swain, Philip H and Hans Hauska (1977). "The decision tree classifier: Design and potential". In: *Geoscience Electronics, IEEE Transactions on* 15.3, pp. 142–147.

Sycara, Katia et al. (2003). "Automated discovery, interaction and composition of semantic web services". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 1.1, pp. 27–46.

Sycara, Katia and Massimo Paolucci (2004). "Dynamic discovery and coordination of agent-based semantic web services". In: *Internet Computing, IEEE* 8.3, pp. 66–73.

Torres, Romina, Hernan Astudillo, and Rodrigo Salas (2011). "Self-adaptive fuzzy QoS-driven web service discovery". In: *Services Computing (SCC), 2011 IEEE International Conference on.* IEEE, pp. 64–71.

Toutanova, Kristina and Christopher D Manning (2000). "Enriching the knowledge sources used in a maximum entropy part-of-speech tagger". In: *38th Annual Meeting* *of the Association for Computational Linguistics-Volume 13*. Association for Computational Linguistics, pp. 63–70.

Universal Description, Discovery and Integration (UDDI). http://uddi.xml.org/. Accessed: 1-5-2013.

Varguez-Moo, Martha, Francisco Moo-Mena, and Victor Uc-Cetina (2013). "Use of Classification Algorithms for Semantic Web Services Discovery". In: *Journal of Computers* 8.7, pp. 1810–1814.

Verma, Kunal, Karthik Gomadam, et al. (2005). "The METEOR-S approach for configuring and executing dynamic web processes". In: Technical Report.

Vu, Le-Hung, Manfred Hauswirth, and Karl Aberer (2006). "Towards p2p-based semantic web service discovery with QoS support". In: *Business Process Management Workshops*. Springer, pp. 18–31.

Wagstaff, Kiri, Claire Cardie, Seth Rogers, Stefan Schrodl, et al. (2001). "Constrained k-means clustering with background knowledge". In: *ICML*. Vol. 1, pp. 577–584.

Wang, Hongbing. et al. (2010). "Web service classification using Support Vector Machine". In: *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*. Vol. 1. IEEE, pp. 3–6.

Wang, Lijuan, Jun Shen, and Jianming Yong (2012). "A survey on bio-inspired algorithms for web service composition". In: *Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on*. IEEE, pp. 569–574.

Wang, Shuying et al. (2006). "An agent-based Web service workflow model for inter-enterprise collaboration". In: *Expert Systems with Applications* 31.4, pp. 787–799.

Wang, Y and E Stroulia (2003). "Flexible interface matching for web-service discovery". In: *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*. IEEE, pp. 147–156.

Weka 3 - Data Mining Software in Java. Visited: Jan 2015.

Wu, Chen and Elizabeth Chang (2007). "Searching services 'on the web': A public web services discovery approach". In: *Signal-Image Technologies and Internet-Based System, 2007. SITIS'07. Third International IEEE Conference on*. IEEE, pp. 321–328.

Wu, Chen, Elizabeth Chang, and Ashley Aitken (2008). "An empirical approach for semantic web services discovery". In: *Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*. IEEE, pp. 412–421.

Wu, Yaowu, Chun-Gang Yan, Zhenyang Ding, Guo-Ping Liu, Peng Wang, Chao Jiang, and MengChu Zhou (2015). "A Multilevel Index Model to Expedite Web Service Discovery and Composition in Large-Scale Service Repositories". In: *Services Computing, IEEE Transactions on* 99, pp. 1–10. ISSN: 1939-1374.

Xiao, Hua, Ying Zou, Joanna Ng, and Leho Nigul (2010). “An approach for context-aware service discovery and recommendation”. In: *Web Services (ICWS), 2010 IEEE International Conference on.* IEEE, pp. 163–170.

Xie, Fei, Haitao Gong, Donghua Deng, Shu Wang, George Wang, Jicheng Hu, and Phillip Sheu (2006). “Integrating semantic web services for declarative accesses in natural language”. In: *Multimedia, 2006. ISM’06. Eighth IEEE International Symposium on.* IEEE, pp. 201–208.

Xie, Ling-li, Fu-zan Chen, and Ji-song Kou (2011). “Ontology-based semantic web services clustering”. In: *Industrial Engineering and Engineering Management (IE&EM), 2011 IEEE 18Th International Conference on.* IEEE, pp. 2075–2079.

Xu, Jiuyun and Stephan Reiff (2008). “Towards heuristic web services composition using immune algorithm”. In: *Web Services, 2008. ICWS’08. IEEE International Conference on.* IEEE, pp. 238–245.

Xu, Ziqiang et al. (2007). “Reputation-enhanced QoS-based web services discovery”. In: *Web Services, 2007. ICWS 2007. IEEE International Conference on.* IEEE, pp. 249–256.

Ye, Lei and Bin Zhang (2006). “Discovering web services based on functional semantics”. In: *Services Computing, 2006. APSCC’06. IEEE Asia-Pacific Conference on.* IEEE, pp. 348–355.

Yu, Jian, Quan Z Sheng, Jun Han, Yanbo Wu, and Chengfei Liu (2012). “A semantically enhanced service repository for user-centric service discovery and management”. In: *Data & Knowledge Engineering* 72, pp. 202–218.

Zapater, Samper, Dolores Escriva, Francisco Garcia, Juan Dura, and Jose Martinez (2015). “Semantic web service discovery system for road traffic information services”. In: *Expert Systems with Applications* 42.8, pp. 3833–3842.

Zhang, Liang-Jie, Rama Kalyani Tirumala Akkiraju, Henry Chang, Tian-Jy Chao, Jen-Yao Chung, David B Flaxer, Jun-jang Jeng, Pooja Yadav, and Qun Zhou (2007). *Method and structure for federated web service discovery search over multiple registries with result aggregation.* US Patent 7,177,862.

Zhang, Yilei, Zibin Zheng, and Michael Lyu (2010). “WSexpress: A QoS-aware search engine for web services”. In: *Web Services (ICWS), 2010 IEEE International Conference on.* IEEE, pp. 91–98.

— (2011). “WSPred: A Time-Aware Personalized QoS Prediction Framework for Web Services”. In: *Proceedings of IEEE Symposium on Software Reliability Engineering (ISSRE’11).*

Zhao, Ben Y, Ling Huang, Jeremy Stribling, Sean C Rhea, Anthony D Joseph, and John D Kubiatowicz (2004). "Tapestry: A resilient global-scale overlay for service deployment". In: *Selected Areas in Communications, IEEE Journal on* 22.1, pp. 41–53.

Zheng, Zibin, Yilei Zhang, and Michael R Lyu (2014). "Investigating QoS of real-world web services". In: *Services Computing, IEEE Transactions on* 7.1, pp. 32–39.

Zhou, Yang, Ling Liu, Chang-Shing Perng, Alfons Sailer, Ignacio Silva-Lepe, and Zhiyuan Su (2013). "Ranking services by service network structure and service attributes". In: *Web Services (ICWS), 2013 IEEE 20th International Conference on*. IEEE, pp. 26–33.

Bio-data

Name:	Sowmya Kamath S
Address:	Assistant Professor Department of Information Technology, NITK Surathkal
Email:	sowmyakamath@nitk.ac.in
Mobile No:	+91 824 247 3557
Qualification:	Ph.D. in Information Technology, NITK Surathkal
	M.Tech. in Computer Science & Engineering, Manipal University
	B.Tech. in Electronics & Electrical Engineering, Manipal University