

A SERVICE ORIENTED PEER TO PEER WEB SERVICE DISCOVERY
MECHANISM WITH CATEGORIZATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUSTAFA ONUR ÖZORHAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

FEBRUARY 2010

Approval of the thesis:

**A SERVICE ORIENTED PEER TO PEER WEB SERVICE DISCOVERY
MECHANISM WITH CATEGORIZATION**

submitted by **MUSTAFA ONUR ÖZORHAN** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Müslim Bozyiğit
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Nihan Kesim Çiçekli
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Assoc. Prof. Dr. Ali Doğru
Computer Engineering Dept., METU

Assoc. Prof. Dr. Nihan Kesim Çiçekli
Computer Engineering Dept., METU

Assoc. Prof. Dr. Ahmet Coşar
Computer Engineering Dept., METU

Asst. Prof. Dr. Pınar Şenkul
Computer Engineering Dept., METU

Asst. Prof. Dr. Aysu Betin Can
Informatics Institute, METU

Date: 05.02.2010

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Mustafa Onur Özorhan

Signature :

ABSTRACT

A SERVICE ORIENTED PEER TO PEER WEB SERVICE DISCOVERY MECHANISM WITH CATEGORIZATION

Özorhan, Mustafa Onur

M.S., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Nihan Kesim Çiçekli

February 2010, 133 pages

This thesis, studies automated methods to achieve web service advertisement and discovery, and presents efficient search and matching techniques based on OWL-S. In the proposed system, the service discovery and matchmaking is performed via a centralized peer-to-peer web service repository. The repository has the ability to run on a software cloud, which improves the availability and scalability of the service discovery. The service advertisement is done semi-automatically on the client side, with an automatic WSDL to OWL-S conversion, and manual service description annotation. An OWL-S based unified ontology -Suggested Upper Merged Ontology- is used during annotation, to enhance semantic matching abilities of the system. The service advertisement and availability are continuously monitored on the client side to improve the accuracy of the query results. User-agents generate query specification using the system ontology, to provide semantic unification between the client and the system during service discovery. Query matching is performed via complex Hilbert Spaces composed of conceptual planes and categorical similarities for each web service. User preferences following the service queries are monitored and used to improve the service match scores in the long run.

Keywords: Peer-to-Peer Web Service Discovery, Categorization, Ranking

ÖZ

KATEGORİZASYON DESTEKLİ, HİZMET ODAKLI, EŞLER ARASI ÖRÜN SERVİS KEŞİF MEKANİZMASI

Özorhan, Mustafa Onur

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Nihan Kesim Çiçekli

Şubat 2010, 133 sayfa

Bu tez ürün servisi tanıtm ve eşleştirmenin otomatik olarak yapılabilmesi için OWL-S teknolojisi temelli bir teknik sunar. Sunulan çözümde servislerin keşfi eşler arası merkezi bir servis kütüphanesi ile sağlanır. Bu kütüphane servis keşfini daha ölçeklenebilir ve erişilebilir kılmak için bulut mimarisi üzerinde çalışabilmektedir. Servis tanımları istemci tarafında yarı-otomatik olarak gerçekleştirilir. Servis WSDL dokümanları otomatik olarak OWL-S formatına çevrilir, ve servis tanımları için ek bilgiler el ile eklenir. Arama sürecinde anlamsal eşleştirmeyi daha iyi yapabilmek için OWL-S tabanlı birleştirilmiş bir üst katman ontolojisi kullanılmıştır. Sorgulara gelen yanıtların doğruluğunu geliştirmek için servis tanıtm ve erişilebilirlikleri istemci tarafında sürekli izlenmektedir. Kullanıcılar servis arama süresinde sistem ontolojisini kullanarak istemci ve sistem arasında anlamsal birlikteliği sağlayan sorgu tanımları üretirler. Her ürün servisi için sorgu eşleştirmesi, kavramsal düzlemler ve kategorilere göre benzerlik değerleri içeren kompleks Hilbert Uzayları aracılığı ile yapılır. Uzun vadede servis sorgularını takip eden kullanıcı tercihleri izlenir ve bu veriler ile sonraki seferlerde daha uygun servisler bulunmaya çalışılır.

Anahtar Kelimeler: Eşler Arası Örün Servis Keşfi, Kategorizasyon, Sıralama

To my wife, mother, father and sister

ACKNOWLEDGEMENTS

I would like to express my most sincere gratitude and appreciation to my supervisor Assoc. Prof. Dr. Nihan Kesim Çiçekli for her endless encouragement and support throughout this study. I am extremely lucky to have such a friendly, wise, patient and benignant supervisor.

I am deeply grateful to my parents, who devoted their life to their children, my loving wife Esra who has been there for me at all times and my sister Pınar Fulya who has been a role model for me, for their love and support. Without them, this work could not have been completed.

I would also like to thank the Central Bank of the Republic of Turkey (TCMB) and the Scientific and Technological Research Council of Turkey (TÜBİTAK) for providing the financial and temporal means throughout this study.

Finally, my special thanks go to Google App Engine, CMU Atlas and SUMO teams who created the infrastructure this thesis builds upon.

TABLE OF CONTENTS

ABSTRACT.....	iv
ÖZ	v
ACKNOWLEDGEMENTS.....	vii
TABLE OF CONTENTS	viii
LIST OF FIGURES.....	xii
CHAPTERS	
1 INTRODUCTION.....	1
2 BACKGROUND INFORMATION AND RELATED WORK	7
2.1 Background Information	7
2.1.1 Web Services	7
2.1.2 Hilbert Spaces.....	12
2.1.3 Suggested Upper Merged Ontology (SUMO).....	14
2.1.4 Distributed Computing Systems	15
2.1.5 Technologies Used.....	18
2.2 Related Work on Web Service Discovery	20
2.2.1 Ontological Approaches.....	21
2.2.2 Service Description with Folksonomies.....	24

2.2.3	Service Description Unification	25
2.2.4	Vector Space Model.....	26
3	USE OF HILBERT SPACES	30
3.1	Hilbert Space Representation of a Sample Web Service	35
4	SYSTEM ARCHITECTURE	42
4.1	Service Discovery Client.....	43
4.2	Service Publisher Peer	44
4.2.1	Service Publishing Wizard	45
4.2.2	Service Annotation Editor.....	45
4.2.3	Service Repository Peer	46
4.3	Service Repository Cloud	47
4.3.1	Service Repository Publisher	47
4.3.2	Service Discovery Manager	48
4.3.3	Service and Publisher Presentation Layer	49
4.4	Cloud Computing and Peer to Peer Infrastructure.....	49
4.4.1	Setting Up a Cloud Computing Infrastructure.....	50
4.4.2	Data Store Restrictions.....	50
4.4.3	Query Processing In a Peer to Peer Network	52
4.5	Service Discovery and Publishing API.....	55
5	SEMI-AUTOMATED SERVICE PUBLISHING	57

5.1	Publisher Identification and Service Selection.....	58
5.2	WSDL to OWL-S Conversion.....	60
5.3	Service Descriptor Annotation and Main Category Selection	62
5.4	Representing of SUMO Concepts	66
5.4.1	Concepts, Classes and WordNet Words.....	67
5.4.2	Concept Plane Categorization	72
5.4.3	Concept Node Leveling.....	73
5.5	Service Description Upload.....	74
5.6	Service Description Extraction.....	75
5.7	Category Planes and Main Categories	77
5.7.1	Service Category Plane Generation	78
5.7.2	Dimension Significance	87
5.7.3	Category Plane Similarity Measures.....	88
5.7.4	Space Modification with User Feedback	90
6	AUTOMATED SEMANTIC SERVICE DISCOVERY	92
6.1	Query Generation.....	93
6.1.1	Query Generation by a User Agent.....	93
6.1.2	Query Generation by a Software Agent	94
6.1.3	OWL-S Ontology Concepts	95
6.1.4	AND/OR Filter Operators	95

6.2	Query Parsing and Categorization	96
6.3	Categorical Service Matching	98
6.4	Query Forwarding and Query Responses	100
7	TESTS AND PERFORMANCE ANALYSIS	104
7.1	System Performance	104
7.1.1	Concept Request for Annotation	107
7.1.2	Web Service Publishing	108
7.1.3	Web Service Discovery.....	110
7.2	Discovery Performance.....	111
7.3	Performance Review	118
8	CONCLUSION AND FUTURE WORK.....	120
	REFERENCES	125
	APPENDIX A ALGORITHM CONSTANTS	133

LIST OF FIGURES

FIGURES

Figure 2-1 Web Service Architecture	8
Figure 2-2 Structure of a SOAP Message	9
Figure 2-3 Contents of a WSDL Document.....	10
Figure 2-4 Elements of an OWL-S File	12
Figure 2-5 Three Dimensional Representation of a Hilbert Cube	13
Figure 3-1 Three Dimensional View of a Concept.....	32
Figure 3-2 Three Dimensional Excerpt from the Skew Coordinate System.....	33
Figure 3-3 Two Dissimilar Conceptual Categories	34
Figure 3-4 Two Similar Conceptual Categories.....	34
Figure 3-5 Annotated Profile of “Server Time Synchronization Service”	36
Figure 3-6 Computing and Weight Dimensions.....	38
Figure 3-7 Engineering and Weight Dimensions	39
Figure 3-8 Generic and Weight Dimensions.....	40
Figure 4-1 A General Outline of the Proposed System	43
Figure 4-2 Direct Connection to Service Repository Cloud	54
Figure 4-3 Direct Connection to Service Publisher Peer	54
Figure 4-4 Central Repository Cloud Forwards to Service Repository Peer	55

Figure 5-1 Descriptor URL Mappings for OWL-S Descriptors.....	61
Figure 5-2 Service Repository Cloud Ontology Reference	62
Figure 5-3 OWL-S Translation of a Simple Type in WSDL	63
Figure 5-4 OWL-S Translation of a Complex Type in WSDL	64
Figure 5-5 Main Categories in the Service Repository Cloud	66
Figure 5-6 A Sample SUMO Concept.....	67
Figure 5-7 SUMO Concept Structure	68
Figure 5-8 SUMO Class Structure.....	69
Figure 5-9 WordNet Concept Structure	71
Figure 5-10 Concept Plane Categorization Algorithm	73
Figure 5-11 Concept Node Leveling Algorithm.....	74
Figure 5-12 IOPE Information Extracted from OWL-S Descriptors.....	76
Figure 5-13 Service Name and Description Extracted from OWL-S Descriptors	77
Figure 5-14 SUMO and WordNet Ontology Mappings.....	79
Figure 5-15 A Sample Category Plane Generated by Service Repository Cloud	80
Figure 5-16 A Sample Parent Child Relationship	81
Figure 5-17 Weights for a Sample Parent Child Relationship	82
Figure 5-18 Service Repository Cloud Plane Generation Algorithm	83
Figure 5-19 Weight Limitation Descriptions	84
Figure 5-20 Non-Conceptual Data Point Selection	86

Figure 5-21 Dimension Significance Computation Algorithm	87
Figure 5-22 Category Plane Similarity Measure Algorithm	89
Figure 6-1 Service Discovery Query Representation	93
Figure 6-2 Example Query Generated by a User Agent	94
Figure 6-3 Query Criteria Importance Order.....	96
Figure 6-4 Query Categorization Algorithm	97
Figure 6-5 Service Match Ranking Algorithm.....	99
Figure 6-6 Contents of a Service Match Record	102
Figure 6-7 Categorical Graphic for a Sample Web Service.....	103
Figure 7-1 Google App Engine Resource List	105
Figure 7-2 Entity Distribution in Data Store.....	106
Figure 7-3 Storage Space by Property Type	107
Figure 7-4 Concept Request Performance	108
Figure 7-5 Web Service Publishing Performance	109
Figure 7-6 Web Service Publishing Detailed Performance	110
Figure 7-7 Web Service Discovery Performance	111
Figure 7-8 Hilbert Space Statistics for Published Web Services.....	112
Figure 7-9 Sample Web Service Profile.....	113
Figure 7-10 Group 1 Query Sample.....	114
Figure 7-11 Group 2 Query Sample.....	114

Figure 7-12 Group 3 Query Sample.....	115
Figure 7-13 Summary Results for 300 Group 1 Queries	116
Figure 7-14 Summary Results for 300 Group 2 Queries	116
Figure 7-15 Summary Results for 300 Group 3 Queries	117
Figure 7-16 Aggregated Summary Results for All Types of Queries	118

CHAPTER 1

INTRODUCTION

Service oriented technologies like Web Services [1] and Service Oriented Architecture [2] revolutionized the modern world of computing with the paradigm shift they brought. The increasingly used technologies provide an answer to one of the fundamental problems enterprises face: complexity. Since the beginning of software development, software being developed has gotten far more complex everyday. This process has both been driven by technology, upon the increase of processing power and storage space, and user requirements. The software of the modern world contains far more lines of code than its ancestors. Then again, the newly developed, complex software depends on a wide range of other modern and complex software too.

The unstoppable increase in software complexity brings a huge problem to the world of software development, because the main actor in the equation -the humans- are still there, and they are prone to make errors in software development lifecycle more than ever. Therefore in this new world, a new paradigm shift should happen, a shift which can change the way software is developed, maintained and used.

This change is the Web Services paradigm, because with Web Services, a unit of work can be done in a contained environment. Multiple operations are carried out either by multiple processes in a web service or by multiple different web services. The functionality that a web service provides, the inputs of the process and the outputs that can be expected from a web service are all defined in a descriptor document, allowing easier software development and execution.

The Web Service paradigm also accepts change as a natural fact in the process of software development and manages change through functionality based service development. This way, the impact of a change is limited to the service that provides the functionality it affects.

Since they provide a new perspective to the software development world, and are widely used ever since their introduction in 2003, web services are a very important aspect of Information Technology. Therefore, a large number of enterprises nowadays is implementing a SOAP/WSDL/UDDI layer on top of existing applications or components and is assembling applications by consuming web-services [3].

However, in time, the common usage of web services brought new problems. For instance, for a service user to be able to use a Web Service, he must have access to the descriptor of the service, which contains crucial information on how the service is to be accessed. Then again, the service user should be able to access the Web Service descriptions of multiple Web Services from a preferably single, but by all means pre-defined location.

In general, these pre-defined locations, containing service descriptions are called service repositories. The standard for service repositories is UDDI [4], short for Universal Description Discovery and Integration. UDDI is a standard managed by the OASIS [5] group, and several UDDI implementations have surfaced in the past, including a relatively popular implementation by Microsoft, named UDDI Business Registry [6]. Even though the concept of a central registry for services seemed

logical, Microsoft, IBM and SAP all closed down their UDDI registries in year 2006, due to little interest shown by service publisher companies.

There are mainly three types of service repositories, (i) centralized service registries, (ii) peer-to-peer service registries and (iii) service search engines. UDDI belongs to the first group. In the first two types of service repositories, the content is willingly provided to the service repository, whereas in the third alternative, a bot crawls the web to find services, and adds them to its own database.

The common property among all three types of repositories is that they work with standardized service descriptions such as WSDL [7] or OWL-S [8].

Web services are traditionally described with the use of the WSDL, which unfortunately cannot adequately represent their actual semantics, i.e. capabilities, inputs and outputs [9]. WSDL is shorthand for Web Service Description Language, and it can be used to describe where a service resides on a network, how it can be connected to, and what kind of parameters should be passed to that service in order to get a meaningful response. The response to be received from the service can also be described in a WSDL file. The primary problem about a WSDL file is that, it does not have semantic information which can be used by a software-agent to make automated service selection or invocation decisions. Yet again, WSDL files cannot specify the preliminary requirements for and outcomes of a service.

These problems can be solved by the semantic web technology which supports annotation of service descriptions via ontologies. Most prominent ontology-based approaches in Web services description are OWL-S, WSMO and SAWSDL [9]. Our system uses OWL-S, since it is a W3C recommendation. OWL-S is based on Web Ontology Language (OWL) and can be used to represent services semantically for software agents. Semantic properties are added to OWL-S descriptions via RDF [10] based ontologies, which can be represented in OWL files. These semantic ontologies are developed by third parties, to describe a specific part of the world with as much

detail, and as much entities as possible. Certain relationships between entities describing the world are also setup in the ontologies.

Using ontological concepts for service description in OWL-S files, the meaning of the functionality of a service can be passed to a software-agent, eliminating human intervention in many processes such as service discovery and service composition.

OWL-S also brings more advantages to WSDL with its service composition constructs for service orchestration and precondition and effect constructs for service input and output description [8].

Therefore, software agents have a better chance of executing services properly given that the descriptions are based on OWL-S rather than WSDL. However, to be able to properly execute a web service, first a selection from a set of web services should be made. This process is called *Web Service Discovery*, and it can be done using Web Service Repositories, or by other methods discussed in section 2.2.

In Web Service Discovery, a user-agent or a software-agent tries to find suitable services to move from a start state to a goal state. In this challenge, the Service Repository provides a list of services based on agent's preferences, such as input-output names, counts and conceptual similarities between agent's query and the web service.

Most of the Web Service Discovery systems are manual, containing a human actor selecting a service from a set of available services. But nowadays there has been an enormous increase in the number of available web services and the web service discovery has been a complex task for a human being to accomplish correctly and efficiently. As a result, automated approaches for discovering web services have been emerged, and became quite popular.

The main problems faced in Web Service Discovery are the availability of the Service Repositories, availability of the Web Services and semantically deficient Web Services. Enabling available Service Repositories is a relatively easy task, with

many centralized and decentralized peer to peer architectures available. Increasing the number of Web Services in a repository also depends on the availability, scalability and semantic power of such a Service Repository. A system is described as scalable if it is able to accommodate an increasing number of elements or objects and/or to process growing volume of work gracefully [11]. Our system, and many other systems [9,11,12,13], challenge scalability with cloud computing and peer to peer architecture. Therefore the main problem in Web Service Discovery is semanticity.

As stated in [13], semantic discovery is enabled by adding semantic annotations to Web service specifications either in registries or service descriptions. Web services are described using WSDL descriptions currently, which provide operational information only. Even in the case that the service description is in OWL-S format, there is a problem regarding which ontology was used to annotate the service. Since there are an indefinite number of ontologies available and every service publisher is free to select an ontology of his own choice, OWL-S annotations cannot be useful at all times either.

This thesis solves these problems by a centralized, peer-to-peer service repository, which runs on a cloud computing architecture on the service repository side. The proposed system targets the solution of the availability and scalability problems commonly experienced with service repositories. The thesis proposes an Automated Service Discovery approach which allows a software agent to provide a query and get responses to the query, which are ranked based on a certain similarity metric.

The provided architecture provides solutions to the service semanticity related problems with semi-automatic methods for WSDL to OWL-S (formerly DAML-S [14]) conversion, and annotation via a unified ontology which merges a wide range of ontologies together in a single ontology, namely SUMO [15]. WSDL is selected as the source for service descriptor conversion, since it is the current standard for web service description, and is widely used in the industry.

Service discovery related problems are also targeted with the discovery system which uses the unified ontology of the system for query criteria.

The contributions of this thesis can be summarized as follows:

- A scalable and service oriented service discovery and publishing architecture, which makes use of cloud computing and peer-to-peer computing paradigms
- Web service similarity computation based on categories and concepts in Hilbert Spaces
- Semi-automated, semantic web service publishing, using a single, unified ontology
- Automated semantic web service discovery, with web service ranking
- Web service semanticization via web service discovery queries

The rest of the thesis is organized as follows. Chapter 2 provides background information on Web Services, Peer to Peer and Cloud Computing architectures. The supporting technologies are also outlined in this section. There is also a discussion about the related work on automated web service discovery. Chapter 3 shows how we use Hilbert Spaces in our work. Chapter 4 outlines the system architecture, and describes the modules participating in our solution. Chapter 5 describes the methodology used in Semi Automated Service Publishing technique suggested by this thesis, and discusses the issues faced in mapping SUMO ontology to a relational data store. In Chapter 6, the algorithms used in query processing and service ranking are described. In Chapter 7, tests and performance analysis conducted on a stable installation of the software are provided. Chapter 8 concludes the study performed throughout this thesis, and provides a pathway for the future work that can be done.

CHAPTER 2

BACKGROUND INFORMATION AND RELATED WORK

This chapter consists of two main parts. In the first part, the background information about the technical concepts and terminology in this thesis are presented. In the second part, the ideas behind the previous work in the literature about web service discovery are described. The strengths and weaknesses of the previous work are discussed with respect to the system described in this thesis.

2.1 Background Information

In this part, first web services and standardized technologies related web services are presented. Later, the technologies used for the infrastructure in this thesis are investigated.

2.1.1 Web Services

Web Services are defined by W3C as a software system designed to support interoperable machine-to-machine interaction over a network [1]. Web service architecture is composed of three major entities: the service provider, service registry, and service requester [16].

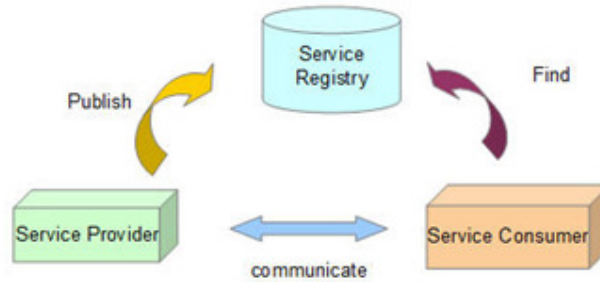


Figure 2-1 Web Service Architecture [1]

In this scheme, a service consumer can use a previously unknown service by discovering it from a service registry; while service publishers can make their services available to a greater set of service consumers by publishing their services.

Web Services provide an interface of the functionality provided by the software, and allow the web service clients to use the functionality provided that they adhere to the specifications in the service description. The main advantages of the Web Service paradigm can be listed as follows:

- Web Services abstract the internal implementation details of the functionality they provide, preventing implementation level dependencies from the service consumer.
- Web Services bring interoperability with technology independence and well defined functional interfaces.
- Web Service can be consumed by software-agents automatically, allowing complex applications to be built faster and with less effort.

The Web Service specification specifies SOAP as the Web Service messaging format and WSDL as service description standard [1].

2.1.1.1 SOAP

SOAP stands for Simple Object Access Protocol, and is an XML based communication protocol intended for distributed applications, including web services. SOAP is an application level protocol, which describes how data should be packaged and unpackaged [17].

A SOAP message consists of three parts: (i) SOAP Envelope, (ii) SOAP Header and (iii) SOAP Body. The SOAP Envelope is a wrapper for the SOAP message being transmitted, and describes what is in the message and how the information contained can be processed. The SOAP Header contains auxiliary information such as transactional or permission attributes. The SOAP Body contains the core information being exchanged between the connecting parties.

An example SOAP message is shown in Figure 2-2.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
      <n:priority>1</n:priority>
      <n:expires>2010-01-22T14:00:00-15:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Print your thesis Friday.</m:msg>
    </m:alert>
  </env:Body>
</env:Envelope>
```

Figure 2-2 Structure of a SOAP Message

2.1.1.2 WSDL

WSDL [7] stands for Web Service Description Language, and is an XML based industry standard for describing functionality of a web service. The language allows the service publishers to describe the functionality, inputs, outputs and network location (i.e. service host, port) of a web service.

A typical WSDL document contains the items outlined in Figure 2-3 when describing a service.

WSDL Item	Definition
Types	A container for data type definitions using a type system.
Message	An abstract, typed definition of the data being communicated.
Operation	An abstract description of an action supported by the service.
Port Type	An abstract set of operations supported by one or more endpoints.
Binding	A concrete protocol and data format specification for a particular port type.
Port	A single endpoint defined as a combination of a binding and a network address.

Figure 2-3 Contents of a WSDL Document

2.1.1.3 OWL-S

OWL [18], formerly DAML+OIL [14], stands for Web Ontology Language, and is a semantic markup language standard created by W3C for web ontologies. OWL-S is built upon OWL technology, and is the most widely used standard for describing semantic web services. OWL-S uses the RDF technology for markup.

OWL-S has certain advantages over the current web service description language WSDL. OWL-S provides ontological annotations for concepts used in a service description, allowing machines to understand and process service descriptions and reach semantic decisions. Also OWL-S provides service composition capabilities, and thus allows composite web services to be defined in a single service description file [8].

An OWL-S specification of a web service consists of three main parts, which are:

- Service Profile: used for service advertisement and discovery
- Service Process Model: used for describing service's operation structure in detail
- Service Grounding: used to provide technical details on how to communicate with the service.

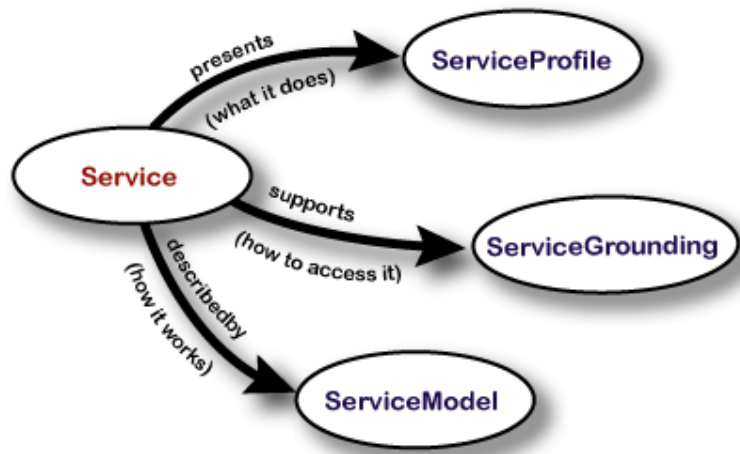


Figure 2-4 Elements of an OWL-S File [8]

2.1.2 Hilbert Spaces

Hilbert Space is a mathematical concept, found by David Hilbert, which extends the methods used in two and three dimensional spaces to a finite or infinite number of dimensions [19]. In engineering, mathematics and physics, Hilbert Spaces are used as infinite dimensional function spaces. The Hilbert Space is used for a similar purpose in this thesis, with the difference that functions in each dimension are dynamically changed and they are discrete.

A data point in a Hilbert Space can be specified by its coordinates with respect to a set of dimensions, similar to Cartesian coordinates commonly used in planes. However, the dimensions of the Hilbert Space can be infinite, thus a data point can be an infinite sequence of coordinates, and hence a function by itself.

Due to their infinite dimensional nature, visually representing a Hilbert Space in the normal world is very hard. However this changes when the Hilbert Space is not

complete (i.e. a function's value cannot be determined based on the previous finite values), and values in a single dimension makes sense on their own. In this case each dimension can be visualized as a 2 dimensional plane in the real world, which is the case in this thesis. A 3 dimensional representation of a Hilbert Space can be seen in Figure 2-5.

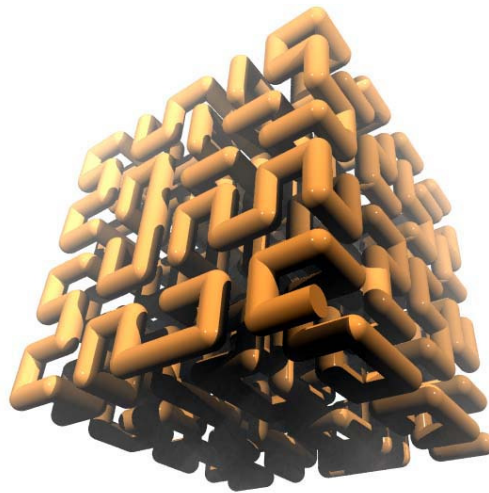


Figure 2-5 Three Dimensional Representation of a Hilbert Cube [20]

The Hilbert Space also contains the concepts of orthogonality and angles. Inter-space angles can be defined and inter-dimensional distances can be measured by traditional Euclidian distance method.

In our system, web services are represented as Hilbert Spaces. The dimensions in the Hilbert Space are the categories present in the system ontology. The data points in the dimensions are concepts used to annotate the web services. The similarity

relations between categories are denoted with pre-computed inter-dimensional intersection angles for category axes.

Similarly, each query presented to the system is also translated to a Hilbert Space. The concepts in the query are extracted, their categories are determined, and a Hilbert Space is created with the given categories and conceptual data points. The query matching algorithm finds the most similar web service Hilbert Space for the created query Hilbert Space.

2.1.3 Suggested Upper Merged Ontology (SUMO)

An ontology is similar to a dictionary or glossary, but with greater detail and structure that enables computers to process its content. An ontology consists of a set of concepts, axioms, and relationships that describe a domain of interest. An upper ontology is limited to concepts that are meta, generic, abstract and philosophical, and therefore are general enough to address a broad range of domain areas. Concepts specific to given domains are included; however, SUMO provides a structure and a set of general concepts upon which domain ontologies (e.g. medical, financial, engineering, etc.) could be constructed [21].

SUMO (Suggested Upper Merged Ontology) [15] was developed within the IEEE Standard Upper Ontology Working Group and it is free. The goal of this Working Group is to develop a standard ontology that will promote data interoperability, information search and retrieval, automated inferencing, and natural language processing.

The SUMO began as a synthesis of a wide range of publicly available formal content, and this synthesis was guided by feedback from the SUO Working Group. SUMO and its domain ontologies form the largest formal public ontology in existence today. The SUMO consists of approximately 4,000 assertions (including over 800 rules) and 1,000 concepts. The SUMO is designed to be relatively small so that these assertions and concepts will be easy to understand and apply.

Some of the general topics covered in the SUMO include [22]:

- Structural concepts such as instance and subclass
- General types of objects and processes
- Abstractions including set theory, attributes, and relations
- Numbers and measures
- Temporal concepts, such as duration
- Parts and wholes
- Basic semiotic relations
- Agency and intentionality

They are being used for research and applications in search, linguistics and reasoning. SUMO is the only formal ontology that has been mapped to the entire WordNet lexicon. SUMO is written in the SUO-KIF language. The ontologies that extend SUMO are available under GNU General Public License [21].

SUMO is used in this thesis because there are a lot of ontologies and the peers in the system need to have an agreement on the ontology they are using.

2.1.4 Distributed Computing Systems

Distributed computing systems consist of multiple autonomous computers which communicate through a network and interact with each other to accomplish a common goal. There are several architectures for distributed computing [23], including:

- Client-Server: A smart client connects to the server, retrieves data to be processed, and commits the results to the server.
- Clustered: A cluster of machines work in parallel to complete a fraction of a subdivided task.

- Peer to Peer: All responsibilities are divided among the machines equally; management of work and resources can be managed by peers themselves.
- Cloud Computing: Resources in a network are virtualized to create a single environment where the work is completed. Decoupling in processing time and data storage is achieved.

Our system uses Peer to Peer and Cloud Computing architectures, which are discussed in detail in sections 2.1.4.1 and 2.1.4.2 respectively.

2.1.4.1 Peer to Peer Systems

There are many computers in homes and offices whose resources are not fully utilized most of the time. The motivation behind peer-to-peer (P2P) systems is that these empty resources might be used for doing something useful like sharing content or computation, instead of being idle.

According to [24] a peer-to-peer (P2P) system is defined as any distributed network architecture composed of participants that make a portion of their resources (such as processing power, disk storage or network bandwidth) directly available to other network participants, without the need for central coordination instances (such as servers or stable hosts). Unlike traditional client-server architecture, peers are both suppliers and consumers at the same time.

P2P systems first became popular by file sharing systems and began to be used widely, and then they also gained a lot of attention in the social, academic, and commercial communities.

Currently, there are several different architectures for P2P networks [24]:

- Centralized: Napster [25] and eMule [26] are examples of such systems. In this approach an upto date directory of nodes in the network are kept in a central location (i.e. the web site of the software). The participating nodes issue queries to the central location to find which nodes possess a given resource.

- Decentralized but Structured: Freenet P2P [27] network is an example of such a system. Unlike centralized systems, there are no central directory servers indexing the files and nodes. However, the files are placed at the nodes in a structured, algorithmic manner. This way queries can be answered faster, since the file placement algorithm is pre-known.
- Decentralized and Unstructured: Gnutella [28] is an example of such a system. These systems both lack a centralized directory and an algorithm for file placement. The system has no control of the network topology of the nodes.

2.1.4.2 Cloud Computing

Cloud computing is a new computing paradigm, in which the end user of the system (i.e. the developer) is unaware of the details of the internals of the system, such as where the data is stored, computation is made or certain services are running [29].

In cloud computing, the services provided to the developer are dynamically scalable, and extendable. This is mostly accomplished by virtualization of resources as services. With cloud computing, applications can scale to the limits of the hardware provided, and the transitions in the hardware are transparent to the users of the system, developers and the application itself.

Even though the term “Cloud” in Cloud Computing refers to the internet, Cloud Computing architectures can be setup in a local network, to make use of the scalability and abstraction features of the cloud. There are commercially available or free to use Cloud Computing platforms on the internet, including Azure [30], Google App Engine [31] and Rackspace Cloud [32].

There are also Cloud Computing infrastructures that can be setup in an intranet environment, which enables the developers to setup their own cloud and run applications on. Amazon EC2 [33], Xen and Eucalyptus [29] are such infrastructures.

2.1.5 Technologies Used

Our system uses a wide range of web services and distributed computing technologies to provide a scalable and versatile infrastructure. These technologies are discussed in this section.

2.1.5.1 Restlet

REST [34] stands for Representational State Transfer, and is the software architecture used in HTTP. REST is a stateless programming model, in which clients are separated from servers by a uniform interface [35]. In the REST model, servers are not concerned with the user interface and user state, so the server software is simpler and more scalable.

Restlet is a RESTful web framework for Java, and unifies Web Services and Web Sites to Web Applications that are ready for the mobile and semantic web [36].

REST clients use simple HTTP commands (i.e. GET, PUT, POST, DELETE etc.) for commanding the REST server. REST servers receive commands from entities using basic and intuitive URL references (e.g. a GET request to a URL “*http://host/grades/{studentid}*” returns the grades of the student with the given id).

RESTful architecture and Restlets are used in our system to provide simple web services for every kind of service consumer, including mobile device users.

2.1.5.2 Google App Engine

Google App Engine [31] is a platform for developing and hosting web applications in Google-managed data centers. It lets developers run their own applications on Google’s infrastructure. It was first released as a beta version in April 2008.

Google App Engine is cloud computing technology. It virtualizes applications across multiple servers and data centers, making it easy to write scalable applications. This

increases availability which is very important in peer-to-peer systems, and this is the reason behind using this platform in this thesis.

Currently it supports applications written in Java and Python programming languages. In this thesis its Java runtime environment is used.

App Engine includes the following features:

- Dynamic web serving, with full support for common web technologies
- Persistent storage with queries, sorting and transactions
- Automatic scaling and load balancing
- APIs for authenticating users and sending email using Google Accounts
- Scheduled tasks for triggering events at specified times and regular intervals

Restrictions of the infrastructure are as follows [31]:

- Developers have read-only access to the file system on App Engine.
- App Engine limits the maximum rows returned from an entity set to 1000 rows per Data Store call.
- Java applications may only use a subset (The JRE Class White List) of the classes from the JRE standard edition.
- Data Store cannot use inequality filters on more than one entity property per query.

2.1.5.3 App Scale

AppScale [37] is an open source extension to the Google App Engine (GAE) Platform-as-a-Service (PaaS) [38] cloud technology providing a multi-language, multi-component framework for running GAE applications on virtualized cluster systems. It is built upon the GAE SDK to facilitate distributed execution of GAE applications over Xen [39] based clusters, including Infrastructure-as-a-Service (IaaS) [40] cloud systems. AppScale provides a framework with which researchers

can investigate the interaction between PaaS and IaaS systems as well as the inner workings of, and new technologies for, PaaS cloud technologies using real GAE applications [37].

The AppScale image implements multiple system components: an AppController, Database Master/Slaves, and AppServers. The AppController automatically spawns all other components, provides the initial contact point for GAE application users (for load-balancing purposes), implements resource monitoring and cloud expansion/contraction. GAE applications execute via the AppServers which are the instances with which GAE users interact once a session is initiated with the AppController. AppScale decouples the database backend of GAE to support different database technologies [37].

AppScale is used in this thesis because it enables users to execute GAE applications using their own clusters with greater scalability and reliability than the GAE SDK provides. Moreover, AppScale executes automatically and transparently over cloud infrastructures such as the Amazon Web Services (AWS) Elastic Compute Cloud (EC2) and Eucalyptus, the open-source implementation of the AWS interfaces.

2.2 Related Work on Web Service Discovery

Web Service Discovery has been a popular research topic recently, and there exists a large number of important previous works. While some approaches use a centralized service repository, some perform discovery in a fully decentralized peer to peer environment. Despite the huge attention paid to collaborative usage of peer-to-peer and Web Services, most of the proposed solutions are primarily focused on enhancing web service discovery by replacing centralized service registries with distributed peer-to-peer architectures [3]. Some approaches focusing on the semantic properties of the procedure make use of computational linguistics and word processing techniques to mine for semantic information in the web services, and some use ontological service descriptions. In this section some of the most important and influencing works in the field of Web Service Discovery are outlined.

2.2.1 Ontological Approaches

There are multiple approaches for web service discovery, which rely on ontologies to semantically annotate the functionalities of web services, including [12], [13], and [41].

The last version of UDDI protocol supports multiple registries and standardizes the communication infrastructure among them, however discovering from and publishing to hundreds of registries at the same time is hard for the service providers. METEOR-S Web Service Discovery Infrastructure (MWSDI) [13] tries to resolve this registry location problem, and has a built-in “Registries” ontology, which keeps track of all the registries registered to it.

In METEOR-S approach, registries are categorized ontologically, and a service is published to the registry which is responsible for its category. The registration is made via a semantic service descriptor. Service registry operators can create their own domain specific ontologies and thus extend the system. The registries are interconnected with a P2P messaging layer, based on JXTA [42].

There are levels of peers: Operator Peer, Gateway Peer, Auxiliary Peer and the Client Peer. Operator Peer controls a registry and provides certain services on that registry. Gateway Peers allows operator peers to join the MWSDI network. When a new registry is added to the system, the registries ontology is updated by the gateway peer. The Auxiliary Peers serve the registries ontology, for safeguarding delivery of the ontology. Client Peers are the consumers of the system.

In a pure P2P network, all peers have equal roles and there is no centralization. In hybrid P2P networks, some resources or services are centralized [13]. Although the infrastructure of the METEOR-S approach seems similar, our system is more of a hybrid P2P network, since it is centralized, and client peers do not need to discover any neighboring peers, since they are forwarded automatically by the central system when necessary. Our system tries to achieve scalability through cloud computing

architecture and large scale service publishers, rather than relying on individual service publishers.

Semantically, METEOR-S contains multiple ontologies, and requires a level of ontological mapping for discovery queries. This requires service publisher generated ontologies to be carried around Operator Peers, and complicates the discovery procedure as number of publisher generated ontologies increase. Our approach however uses a unified, merged ontology which is valid for all the peers in the system.

METEOR-S uses WSDL files for service discovery and relies entirely on WordNet to find similarities between a query and a WSDL file, which is not semantically annotated. The similarities are found by parsing WSDL files and generating additional words via WordNet relations (i.e. synonym, hypernym, acronym etc.). The matching is done via the NGram algorithm and results of the mappings are displayed to the user for user feedback so that he can accept or reject these mappings. This makes METEOR-S a semi-automated web service discovery stack, while our system is fully automated and does not require user feedback since mappings are not necessary.

The work presented in [12], has a similar ontological approach. According to the authors, majority of currently existing approaches focuses on centralized architectures and deals with efficiency typically by pre-computing and storing the results of the semantic matcher for all possible query concepts. The web service discovery technique described in [12], matches a request and an advertisement in constant time.

The service representations are indexed to prune the search space to minimize the number of required comparisons. The degree of match between two concepts is assessed by the extent to which the subtrees of their concepts overlap. The concept overlapping is defined as ranges, and they are calculated for once via the ontology's acyclic graph once the service is added. Later, each interval for the parameters is

represented as a point in a 2-dimensional space. This way containment between intervals is a range query on this 2-dimensional space.

When a service is to be searched, parallel searches are conducted for each request parameter as range queries, and the results are finally intersected to compute the final matches. When searching, if an exact match is required, a point is searched in the 2-dimensional space; for plug-in and subsumes matches, an interval is being searched in the 2-dimensional space.

The infrastructure relies on a spatial P2P network, which operates on spatial data. The peers are organized with respect to relatedness, and the search is propagated as a range query to the nodes that have matching characteristics. When a new service description is published, the description is encoded using the given interval computation, and each encoded service is hashed to and eventually stored by a peer whose ID is closer to its value in the 2-dimensional space. Therefore similar services are stored by the same or neighbor peers.

The work is similar to our approach in the sense that ontological concepts are mapped to a plane and similarities between concepts are enriched with ancestry relationships. However, our approach uses a complex space, in which there exists an indefinite number of conceptual dimensions, whereas the described approach uses a single 2-dimensional space. In our system inter-dimensional distances are calculated, and intra-dimensional calculations are made via conceptual weighing schemes. Also in our system there are plane significance measures to control the importance of concepts stored in a plane, which naturally lacks in the system described, due to the existence of a single 2-dimensional plane.

However, the work described in the paper performs much better in means of performance with a constant time algorithm for service and advertisement matching, due to precomputation of both data points and ranges at publishing time; whereas our system makes comparisons at $O(nm)$ where there are n dimensions and m data points at each service discovery request.

The work described uses an entirely decentralized, peer to peer architecture, while our system uses a centralized system and relies much less on the peers for service discovery.

2.2.2 Service Description with Folksonomies

Folksonomies are collaboratively generated and managed tag clouds, intended to annotate and categorize content. In contrast with ontologies, the metadata used to categorize content is not generated by experts in the field, but by creators or end users of the content. In this sense folksonomies are uncontrolled, user generated and dynamic ontologies [9].

Works presented in [9] and [43] use folksonomies for semantic resolution of web services. In [9], the tags created by the users of the system are supported by words found via WordNet relations. During matchmaking, Term Frequency - Inverse Document Frequency [44] technique is used to find the similarity between the service descriptor and the query.

Using the top-k terms in all documents in a corpus, domain folksonomies are constructed and services annotated by the end users are categorized into domains, to restrict the service search space during matchmaking.

The infrastructure used for peer to peer communication is JXTA, and is a semi-decentralized architecture. The peers are organized into domain specific groups with respect to the created domain folksonomies, and services in the same domain are stored locally in these peer groups. With each web service publishing operation, WSDL files are multicasted in the domain specific peer subnet.

The system described is similar to our architecture in the sense that peer groups are organized into categorical subnets, and system is semi-decentralized. However in our system, a service is published to a single location (i.e. a peer, or the central server), and data is not disseminated among peers in a categorical subnet. Therefore

the data stores of peers in the same category do not carry the same amount of information, as in the described work. This makes our system more centralized.

Semantically, the system relies on consumer's tagging behavior and WordNet relations to enhance the semantic properties of the services described with a WSDL descriptor after publishing. In our system OWL-S files generated from WSDL files are annotated with a unified ontology, prior to publishing, and annotations are enriched with the queries of the users, without the explicit consent of the user.

In [43], in addition to the tagging system described in [9] a rating system is incorporated to the service matchmaking process. The rating system allows the users to rate the web services they use in terms of functionality. The matchmaker aggregates the user feedback data and rewards the services with better ratings in the service matchmaking process.

2.2.3 Service Description Unification

The work described in [45] underlines the fact that there are, and there will be, many different languages for describing web services, and they argue that, enforcing a single descriptor or trying to convert service descriptors back and forth is an ineffective way of dealing with web service discovery.

They propose a new technique named "Metric Space Approach", in which heterogeneous service descriptors are semantically evaluated and modeled as metric objects to the same metric space, regardless of the concrete service description languages. Thus the web service discovery problem is transformed to a similarity search problem, and the transformation is described as a meta-model on the existing web services description language, in which, all heterogeneous web services are modeled as similar metric objects regardless of the concrete description languages, and thereby the discovery problem can be treated as similarity search in the metric space with a uniform criterion [45]. For similarity search, two techniques are used: p-KNN query [46] is used for the functional semantics of the service descriptor and Range Query is used for the non-functional semantics of the service descriptor.

Input, Output, Precondition and Effect variables of a web service are accepted as functional semantics of the web service, whereas Quality of Service and context policy are accepted as non-functional semantics.

The work is similar to the technique described in this thesis, in the sense that service descriptors are modeled as entities in a space. Our work categorizes the conceptual descriptions and represents the web service as a complex Hilbert Space, whereas the system described simplifies the web service to a metric space, and places each service on the same space.

Our similarity measure depends on both intra-plane and inter-plane conceptual intersections and concept significance values; however the work described uses a single metric representation for each service and tries to find nearest neighbors in a metric space for semantic search. Non-functional semantics of the web service are taken into account in the described work, and similarities regarding QoS parameters and context policies are searched via range queries, however in our system only Last Online Date is taken into account.

2.2.4 Vector Space Model

Traditional semantic web service discovery frameworks rely on accurate description of web services. Most of the services in the service repositories of these frameworks are simple web services. The discovered services are composed by the user or software agents to create complex applications. The works described in [47] and [48], try to replace this process with a search engine using the Vector Space Model [49] which can index both simple and already composed complex web services.

The approach described in [47] does not use a semantic service description framework as OWL-S, rather relies on text mining techniques to extract semantic information from data (i.e. method names, URLs etc.) that is already present in the WSDL documents.

The services are collected by a search engine that crawls the Internet, and other data sources for service descriptions. The crawler can process both WSDL service descriptors and UDDI registry entries. An option for direct upload to the search engine is also provided to service publishers.

Once a service is collected, it is processed for valuable semantic information, and a Term Space is created for it. TF-IDF [44] technique is used to determine which keywords to add to the Term Space. Term weights are assigned to the words based on their frequency in the document.

The terms created in the Term Space are added to the Vector Space of the service. Lengths of the vectors are defined by the weights of the terms. Vector Space Model is a search mechanism that is widely used in modern information retrieval systems. The model describes a web service as a set of vectors that represent keywords extracted from the service description document. Each vector resides in its own dimension, and distances between these vectors are measured via several mathematical methods. The mathematical method used to calculate the distances between vectors in this paper is Cosine Similarity [50].

The Vector Space Model resembles our system in the sense that it has a multi-dimensional model for representing web services. The Vector Space Model contains a single vector in each dimension, and these dimensions represent keywords. In our system, dimensions represent conceptual categories, and each dimension has a descriptor function consisting of multiple data points, which represent conceptual data points and their weights.

Vector Space Model uses Cosine Similarity to find distances between vectors in different dimensions, whereas our approach uses plane distance and concept level measures. Our approach contains significance values for each conceptual plane and weight values for each data point, whereas Vector Space Model only contains weights for each keyword.

The underlying architecture in the described paper is a semantic web service search engine; whereas our approach uses centralized peer to peer service repositories that run on cloud computing architecture.

The work described in [48] relies on Vector Space Model too; however it uses a Query by Example Approach to discover Web Services, that is called WSQBE. WSQBE works incrementally, by first reducing the search space to a subspace, and then comparing the query to services in the relatively small subspace.

The novelty of the approach comes from the set of queries accepted by the system, WSQBE supports queries expressed as partial web service descriptions, source code method declarations or natural language descriptions of the expected service or its operations [48].

Service clients generate a query by creating a service description skeleton, which contains a partial functional description, related keywords and expected interface. The queries can be expressed in natural language, or they can be composed with a Java interface. Each query is converted to a WSDL file before the system processes it. The service descriptors created for queries are treated as actual services, and categorized by machine learning algorithms, which are trained on actual UDDI registries. The services published to the repository of the system are also categorized, by TF-IDF technique.

When a query is received by the system, only the services in the same category with the query are processed. During service discovery, services are described as n dimensional vectors, and vector comparison is made. To measure the similarity between the query vector and the services, cosine similarity technique is used.

The approach represents the services in multiple dimensions, and in this way has a similar perspective to us. However in our approach, dimensions are categorical and contain data point generating functions, rather than concept vectors. Also, in our system similarity is measured via significance, level and weight variants of concepts, while in the described work similarity is based on vector angles.

The described work includes a very flexible query generation system, which provides the users a variety of easy to use options; whereas in our system there's only a single query generation mechanism, which uses the system ontology for target values.

CHAPTER 3

USE OF HILBERT SPACES

Hilbert Spaces are complex multi-dimensional spaces with applications in various disciplines including mathematics, physics and engineering. In Mathematics, Hilbert Spaces are used for Functional Analysis [51], which mostly deals with infinitely dimensional, topological vector spaces. In our system, Hilbert Spaces are used in a similar manner; in order to represent web services and queries in infinite dimensional spaces with ontologies.

Hilbert Spaces are selected in our system for three main reasons:

- Hilbert Spaces are infinitely dimensional. Even though our system currently uses finite number of dimensions, the number of dimensions can increase as the unified ontology of the system grows, and new categories are introduced.
- New dimensions can be added to a Hilbert Space within time, since data points in Hilbert Spaces are represented with infinite coordinates. The descriptions of a web service may change in time, and new concepts from previously missing category dimensions can be introduced to Hilbert Spaces.

- Hilbert Spaces support the Skew Coordinate system, with which we can mathematically model the relationships between categories. This allows us to compute inter-dimensional concept similarities easily.

Traditionally, ontologies are directed acyclic graphs, where entities might have more than one parent. Our system uses the unified ontology SUMO, which is created by merging ontologies of multiple categories, and is a directed acyclic graph itself.

Therefore each of the concepts in the unified ontology belongs to one or more categories, and might have parents originating from different categories. Our idea is to represent these concepts and categories in a space.

First of all, the concepts in the ontology are clustered by their categories. Since the ontology allows concepts with parents from multiple categories, certain concepts can be present in multiple categories. The clusters are basically formed with respect to mid-level ontologies that contribute to SUMO's unified ontology.

Each unique category is represented with a dimension in the Hilbert Space. Each dimension carries concepts from their own category. Even though the number of categories in our space is limited with our current ontology, our model can carry an infinite number of categories. Since an infinite number of dimensions are available in the Hilbert Space, additional dimensions can be added to the space, if the ontology is modified, and new conceptual categories are provided.

Additionally, our Hilbert Space contains a weight dimension and a level dimension, as shown in Figure 3-1. Both of these dimensions are orthogonal to the remaining dimensions in the complex space. Weight dimension is used to store the weight information about the concepts added to the space and level dimension is used to store the hierarchical levels of the concepts. The algorithms to obtain the level and weight of ontology concepts are discussed later in sections 5.4.3 and 5.7.1.1 respectively.

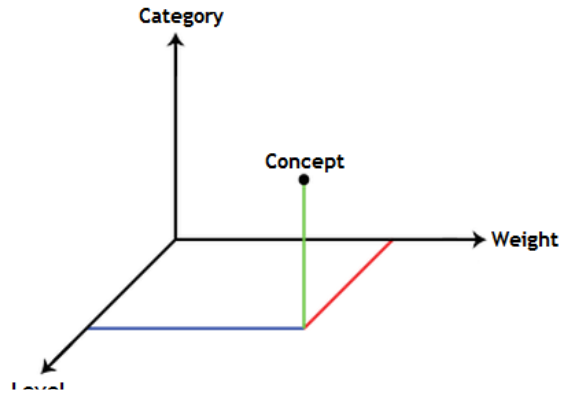


Figure 3-1 Three Dimensional View of a Concept

Due to their infinite nature, the data points in the Hilbert Space are also represented with infinite number of coordinates. In our system, a data point in a given conceptual dimension carries positive values for only two other dimensions, other than its own dimension. These dimensions are the weight and the level dimensions. For all the remaining dimensions, origin coordinates are provided.

Another property of Hilbert Spaces is that, they do not need to be square summable. This means that only countably finite axes that are orthogonal are enough to form an orthonormal basis for the entire space [19]. Unlike a Cartesian Space, in which all the dimensions are orthogonal to each other; Hilbert Spaces can use coordinates that intersect with different angles. A sample space with non-orthogonal intersecting angles is shown in Figure 3-2.

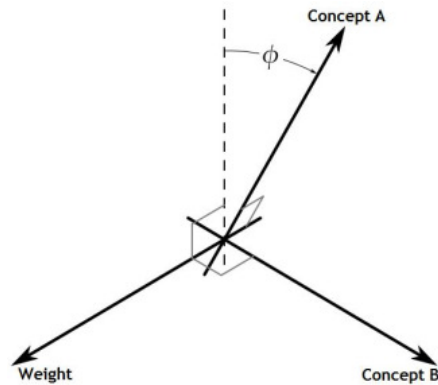


Figure 3-2 Three Dimensional Excerpt from the Skew Coordinate System

This property allows us to use the Skew coordinate system instead of Cartesian coordinate system, which includes axes intersecting with different angles. The Skew coordinate system is especially useful when working with problems that fit well to a skewed system.

Categorical representation of web services concepts is such a problem, since our system depends on not only the similarity of concepts in the same dimension, but also the similarity of concepts in different dimensions, and different dimension pairs have different similarities.

Therefore in our space, similar categories have less than 90° intersection angles, and dissimilar categories have more than 90° intersection angles. As the similarity increases, the intersection angle decreases. Hence concepts in similar dimensions are mathematically closer to each other, and concepts in dissimilar dimensions are farther from each other. Figure 3-3 and Figure 3-4 displays the alignment of two similar and dissimilar categories.

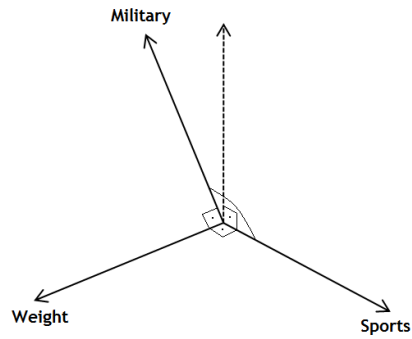


Figure 3-3 Two Dissimilar Conceptual Categories

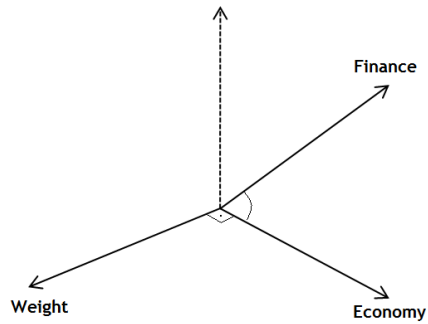


Figure 3-4 Two Similar Conceptual Categories

The computations are carried out by the algorithm described in section 5.7.3. Since each dimension interacts with any other dimension in the space, the similarities between each dimension are computed. Even though the conceptual dimensions are oblique to each other, all the conceptual dimensions are orthogonal to weight and

level dimensions of the space and the weight and level dimensions are orthogonal to each other.

3.1 Hilbert Space Representation of a Sample Web Service

In this section, we present an example web service, and the Hilbert Space generated for it by our system. Although the presented service is simple, the generated category planes and data points reflect the powers and weaknesses of our approach thoroughly.

The example web service is named “Server Time Synchronization Service”, and its main function is remotely synchronizing the time of a server in a specified network with respect to a given time zone. The date and time used in the synchronization process, and a status message showing whether the operation has succeeded is returned to the web service caller. The profile definition contains the information shown in Figure 3-5. The service name provided by the publisher to the system is “ServerTimeSynchronization” and the main category selected for the web service is “computing”.

OWL-S Profile Description

```
<process:Input rdf:ID="ServerTimeSoap_GetNetwork_parameters_IN">
  <process:parameterType rdf:datatype="&xsd:anyURI">
    &wsones;#network
  </process:parameterType>
</process:Input>
<process:Input rdf:ID="ServerTimeSoap_GetServer_parameters_IN">
  <process:parameterType rdf:datatype="&xsd:anyURI">
    &wsones;#server
  </process:parameterType>
</process:Input>
<process:Input rdf:ID="ServerTimeSoap_GetDateTime_parameters_IN">
  <process:parameterType rdf:datatype="&xsd:anyURI">
    &wsones;#calendar
  </process:parameterType>
</process:Input>
<process:Output rdf:ID="ServerTimeSoap_GetDate_Date_OUT">
  <process:parameterType rdf:datatype="&xsd:anyURI">
    &wsones;#date
  </process:parameterType>
</process:Output>
<process:Output rdf:ID="ServerTimeHttpGet_GetDateTime_Time_OUT">
  <process:parameterType rdf:datatype="&xsd:anyURI">
    &wsones;#time
  </process:parameterType>
</process:Output>
<process:Output rdf:ID="ServerTimeSoap_GetDateTime_Status_OUT">
  <process:parameterType rdf:datatype="&xsd:anyURI">
    &wsones;#status
  </process:parameterType>
</process:Output>
```

Figure 3-5 Annotated Profile of “Server Time Synchronization Service”

With the given information, our algorithm starts creating conceptual dimensions in the Hilbert Space of the web service. The first created dimension is the “computing” dimension, since it is the main category of the web service. The ontological types and RDF identifiers of inputs and outputs are examined, and relevant data points are plotted in the “computing” dimension.

Hilbert Spaces can be decomposed into mutually orthogonal subspaces [19]. A generalization of an orthogonal decomposition of H can be made as:

$$H = W_1 \oplus W_2 \oplus \dots \oplus W_n$$

into mutually orthogonal subspaces W_1, W_2, \dots, W_n such that each coordinate $c \in H$ has a unique representation as $c = c_1 + c_2 + \dots + c_n$, with $c_i \in W_i; 1 \leq i \leq n$.

Using the orthogonal decomposition approach, a 2-dimensional subspace containing “computing” and weight dimensions extracted from the web service’s Hilbert Space can be generated, as shown in Figure 3-6.

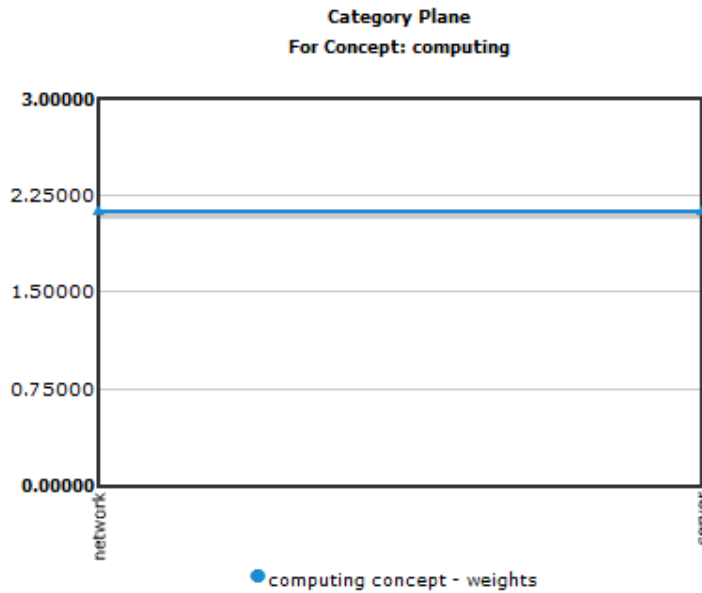


Figure 3-6 Computing and Weight Dimensions

As seen in Figure 3-6, there are only two data points in the “computing” dimension. This is an unexpected situation, since the main category of the service is “computing”. The problem is, the SUMO ontology does not contain any parent or child concepts in the “computing” category for the concepts “network” and “server”. For instance, the direct parent of the concept “server” is the concept “computer”, and it belongs to the “engineering” category, hence it is placed under that dimension, as shown in Figure 3-7.

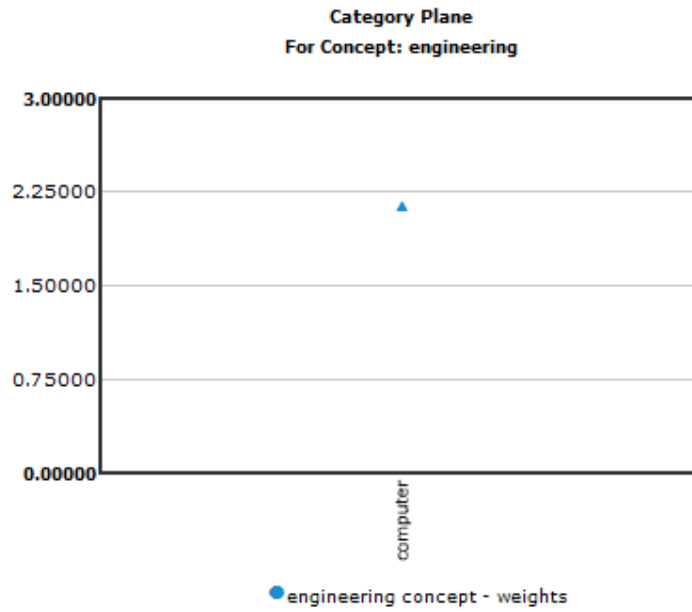


Figure 3-7 Engineering and Weight Dimensions

Apart from the inputs with conceptual “server” and “network”, there is other useful information in the description which our system can use to generate dimensions. For instance the input “calendar”, or outputs “status”, “date” and “time”. These concepts belong to the “generic” category of the SUMO ontology. The generic dimension is generated as shown in Figure 3-8.

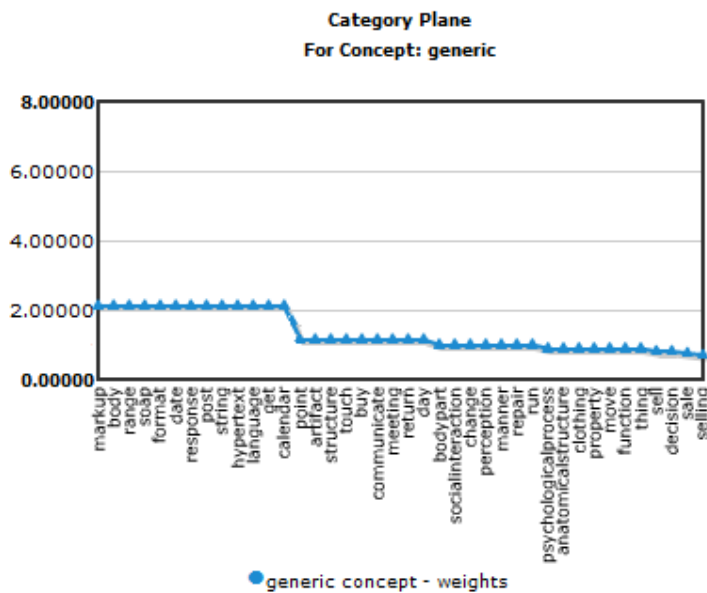


Figure 3-8 Generic and Weight Dimensions

In the “generic” dimension, the situation is a bit different from the aforementioned two dimensions. There are more data points, and the weight values for the data points are varying greatly. Since SUMO has a wide range of ontological concepts available in the “generic” category, our system can find more parent/child concepts, and place them in the dimension. This richness in ontological categories allows web service discovery clients to match the web services easily.

It can be seen that the algorithm placed the “date” concept in the generic ontology, and derived the “day” concept via a parent-child relationship. Some other concepts are derived via the processing of RDF identifiers, for instance the processing of RDF identifier “ServerTimeHttpGet_GetDateTime_Time_OUT” for concept “time”

resulted with a keyword “http” whose definition led to addition of concepts such as “hypertext” and “markup” to the “generic” dimension of the service.

Since the SUMO ontology is not computing oriented, there are very few concepts in the computing category. For this web service, this situation resulted in an unbalanced categorical Hilbert Space. While there are only 2 concepts in the “computing” dimension, there are around 40 concepts in the “generic” dimension. Even though the significance of the “computing” dimension is more than the “generic” dimension, queries targeting the “generic” dimension has a higher chance of matching the service than the queries targeting the “computing” dimension, since they are represented with more concepts; making more intersections possible.

The algorithms for conceptual plane generation, dimension significance computation and dimension similarity computation are discussed in detail in section 5.7

CHAPTER 4

SYSTEM ARCHITECTURE

This chapter outlines the architecture of the proposed web service discovery system along with the infrastructure used by the system. The system is composed of three main components: Service Discovery Client, Service Publisher Peer and Service Repository Cloud.

The implementation of our system mostly contains code developed with Java programming language. The main component of our system, the Service Repository Cloud is a J2EE application that runs on Google's cloud computing infrastructure App Engine. The created Service Repository Cloud can be deployed on the open source cloud computing stack AppScale, which can in turn be setup on popular open source cloud computing infrastructures such as Xen, Eucalyptus and KVM.

The modules of the system are explained in detail in this chapter. An outline of the described system is presented in Figure 4-1.

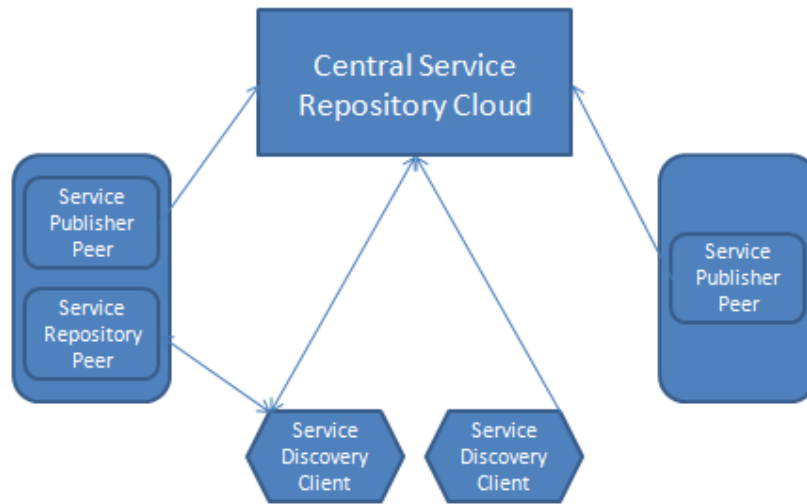


Figure 4-1 A General Outline of the Proposed System

4.1 Service Discovery Client

The Service Discovery Client is the part of the system which is built for the use of the human user agents. The component is implemented as a standalone application and provides an easy to use user-interface.

With the Service Discovery Client, the user can specify certain queries to find a certain web service, or a web service providing certain functionality. The queries are composed of one or more criteria combined with a filter operator.

The criteria can be about the structure of the web service, or they can be about the semantic properties of the web service. While the user agent can specify Input, Output, Precondition and Effect criteria in the query from a structural point of view,

Service Name, Service Category and Service Description criteria are provided to enhance the semantic nature of the query.

The filter operator can be either OR or AND. In the first case, any of the matching criteria will promote in a positive manner to the Service Match Score of a service, while the non-matching criteria will not affect the score in any way. In the latter case, the matching criteria will still promote the Service Match Score; however the non-matching criteria will affect the score negatively. The details of the Service Matchmaking are described in detail in chapter 6.

When the user agent completes query generation, the query is submitted to the Service Repository Cloud. The repository is searched for matching services, and a set of Service Match Records are returned to the user. Service Match Records contain Service Name, Publisher Name and a URL where more details about the service can be obtained. The Service Match Records also provide both ranking information (i.e. Service Match Score) and availability information (i.e. Last Online Date).

4.2 Service Publisher Peer

The Service Publisher Peer is the part of the system built for the use of the web service publishers. The component is implemented as a standalone application, which is composed of three main parts: the Service Publishing Wizard, Service Annotation Editor and Service Repository Peer.

With the Service Publisher Peer, a web service publisher can publish services of his preference to the Service Repository Cloud, and update the online/offline status of the published services periodically. With the embedded Service Repository Peer component, a service publisher can opt to function as a relay station for the Service Repository Cloud.

4.2.1 Service Publishing Wizard

In the real world, web service publishers tend to use WSDL files to describe the web services they provide. These WSDL files are generally accessible from the web, and most of the time they reside either on the same server with the provided web service or in the same network with the host server of the provided web service.

The Service Publishing Wizard has three steps of execution. In the first step the service publisher is asked to provide a directory which contains WSDL files and a Publisher Key for the Service Repository Cloud. The Publisher Key can be obtained from the Service Repository Cloud web application, by a simple sign-up operation.

The provided directory is scanned recursively for WSDL files, and a list of available service descriptors are presented to the user to select from. The user can select the list of services to be incorporated to the Service Repository Cloud.

Once the services are selected, they are converted and annotated using the Service Annotation Editor and finally they are uploaded to the Service Repository Cloud. However, the Service Publishing Wizard continues to operate, monitoring for changes in the service descriptors. If any of the service descriptors are changed, the new descriptors are converted and uploaded to the Service Repository Cloud as a new revision. The publisher is also notified of the upload to further annotate the new changes to the service, if any.

While checking for changes to the service descriptors, the Service Publishing Wizard also notifies the Service Repository Cloud of the online/offline status of the publisher; assuming the service is either on the same server or at least in the same network with itself.

4.2.2 Service Annotation Editor

The Service Annotation Editor is a simple RDF editor, which automatically converts WSDL files to OWL-S files and allows the service publisher to annotate the OWL-S

Service Profile of the service to be published with the concepts from the Service Repository Cloud's unified ontology SUMO.

During the annotation process the editor scans the Service Profile file generated for the service to find possible annotation end-points, and provides pointers to the service publisher to edit the necessary OWL-S description members.

When the publisher decides to annotate an OWL-S description member, this can only be done using a Concept Auto Complete Widget which connects to the Service Repository Cloud to find concepts similar to what is being typed by the publisher to the widget. The publisher can then select a concept from the provided list of concepts, and the annotation will be inserted to the service description.

When the annotation process is finished, the publisher is presented with a Main Category selection for the service. The selected Main Category will be further used for service categorization and query matching (for more details see section 6.3).

4.2.3 Service Repository Peer

The proposed architecture provides a peer-to-peer query architecture, in which the peers can also act like servers, by storing and serving service information. This approach is embraced to: (i) allow the publishers to provide web service discovery services to their own set of clients in a private network and (ii) form categorically focused sub-service repositories which can be used to better divide the search space.

The Service Repository Peer feature can be activated by the service publisher once the web service selection and publishing stages are completed. The Service Repository Peer will start and notify the Service Repository Cloud of its existence. At that point, the Service Repository Peer will be used in two ways: (i) explicit referrals to the Service Repository Peer as a private Service Repository Cloud by the web service publisher, (ii) referrals from the Service Repository Cloud based on the categorical alignment of the publisher and the queries received from other clients.

In the first case, the Service Repository Peer acts like the Service Repository Cloud to the contacting web service publisher, by storing web service information and maintaining the state information. In the second case, Service Repository Peer performs a search for the received query on its own Data Store and returns the results to the connecting web service publisher, which has been forwarded to by the Service Repository Cloud in the first place.

The implementation of the Service Repository Cloud takes strong precautions for availability and single point of failure issues, which are two of the main problems peer-to-peer systems try to overcome. These precautions are discussed later in section 4.4.3.

4.3 Service Repository Cloud

The Service Repository Cloud is the main component of the proposed system, and consists of a set of modules responsible for service publishing, discovery and presentation. The component allows the service publishers to sign up for the system and upload service descriptions, and update their status.

The Service Repository Cloud runs on a cloud computing architecture, and is hosted at Google App Engine. However, the Service Repository Cloud can be run on any cloud computing platform that is capable of running App Scale, which allows the Service Repository Managers to serve their own Service Repository Clouds in a cloud computing environment of their preference, instead of using the already provided architecture.

4.3.1 Service Repository Publisher

When a web service publisher publishes a web service, the OWL-S service descriptor files (i.e. Service Profile, Service Grounding, Service Process Model and Service Concept) and additional semantic information (i.e. the name of the service, the main category of the service) are transferred to the Service Repository Cloud.

In the Service Repository Cloud the given information is processed and stored in the Data Store. The stored information includes the raw description files uploaded by the publisher and the semantic information extracted by the Service Repository Publisher, which will later be used by the Service Discovery Manager when Service Discovery Clients submit queries. The details of the semantic information extracted by the Service Repository Publisher are discussed in section 5.6.

Another function of the Service Repository Publisher is to update the service descriptions when they are uploaded by the Service Publisher Peers, and keep track of separate revisions of the same web service.

4.3.2 **Service Discovery Manager**

The Service Discovery Manager is one of the most important modules in the proposed web service discovery architecture. It is responsible for receiving a query submitted by a Service Discovery Client, and respond with the most suitable set of services.

The Service Discovery Manager processes the received query, and searches the web service space in the Data Store for matching services, ranks them for similarity, and sends the responses to the Service Discovery Client. While searching for suitable services, the discovery manager first creates suitable category planes for the incoming query, and then matches the category planes with the category planes of the published web services.

The query can also be forwarded to a Service Repository Peer, if any available, when the Main Category for the query matches the Main Category of the Service Repository Peer. The forwarding must be followed by the Service Discovery Client and the query should be submitted to the given Service Repository Peer to obtain results from the given peer. This is the preferred way of Peer-to-Peer communication in the system, since the Service Discovery Client can be an agent with pre-defined time bounds for the query or Service Match Score criteria. The details of the implementation are discussed in section 6.3.

4.3.3 Service and Publisher Presentation Layer

The Service and Publisher Presentation Layer is the module that presents the publishers signed up to the system, and services they have published to the user agents over the web.

The other functionalities of the presentation layer include: (i) allowing the publishers to sign up to the system, and receive publisher keys, which are necessary to publish services using the Service Publishing Wizard, (ii) allowing the service publishers, clients and developers to download necessary software to use the project, (iii) display web services ratings and download counts, (iv) display categorical planes of the web services graphically.

The presentation layer also allows the Service Discovery Peers, Service Discovery and Publishing API users and user agents to download the OWL-S and WSDL service descriptors from the Data Store, to enable communication with the advertised services.

Each Service Repository Cloud and Service Repository Peer comes with a Service and Publisher Presentation Layer of its own. The layer provides the same services to each repository owner. In addition to a traditional peer to peer system which enables query routing and data storing decentralization, this system enables account information and interface decentralization.

4.4 Cloud Computing and Peer to Peer Infrastructure

In the near future most of the service discovery requests received by a service repository will be automated requests originating from software agents, to perform real-time, automated service compositions and invocations [29], [52]. Therefore the most important characteristics a service repository should carry are scalability and availability.

To achieve this goal, our system is built on a cloud computing architecture. Although the architecture is already available on Google's App Engine infrastructure, it can be setup on a system of provider's preference too. However, there are certain limitations of the described infrastructure that must be met in the application, and this section discusses these issues.

4.4.1 Setting Up a Cloud Computing Infrastructure

Our system uses Google App Engine as the cloud computing infrastructure, and deployments are made to a project hosted at Google App Spot [53]. However, our system can also be deployed to an App Scale installation, which can run on the hardware of the system provider's infrastructure.

To enable this, the system provider should first setup a Cloud Computing Stack to his hardware. App Scale supports Eucalyptus, Xen and KVM [54] stacks, of which all three are open source platforms. For our tests apart from Google App Engine, we chose the Xen Hypervisor Cloud Platform, which is an easy to use alternative with its Live CD and relatively large user base.

Upon a Xen Hypervisor platform the App Scale software should be setup. App Scale group provides images for the Xen and Eucalyptus infrastructure, so it is relatively easy to setup App Scale on Xen.

Once an App Scale installation is present, the system provider can select an App Scale supported database (i.e. MySQL Cluster [55]), and App Scale will transparently forward Data Store requests to the database setup.

4.4.2 Data Store Restrictions

Google App Engine is very restrictive when it comes to using the data store. There are a number of restrictions regarding the query response times, the maximum number of indexes a data store entity can have, the query strings to be executed on the data store and so on.

In our case, all SUMO concepts had to be represented in the Data Store, to enable fast response times to service publishers and service discovery agents at the same time. The SUMO team provides OWL-S formatted ontologies for both its unified ontology and the WordNet reference ontology. The unified ontology is as large as 15 MB, and WordNet reference ontology is as large as 150 MB in file size. These files can neither be uploaded to Google App Engine which has a 1 MB file size limit nor deployed to the App Store which has a 5 MB file limit [31].

Even if the files could have been uploaded, Google App Engine does not allow requests exceeding a 30 second query response time limit, a time quite less than required to parse, process and persist the concepts.

Therefore in our local system we have parsed and processed the concepts in the SUMO ontologies and uploaded the results to our local data store. Later, using Google's experimental Remote API we have exported the relational data in our data store to CSV files for bulk upload to the Google Data Store. As a last step, using the Google App Engine Bulk Uploader [56] tool, we have uploaded our data back to Google App Engine, entity by entity.

Google App Engine infrastructure enforces a 30 second per request response time cap. The restriction is there to provide a convenient level of service to all App Engine users; however it is easily met when an appropriate indexing scheme is not enforced on the data store.

There is also a 5.000 property index restriction on the number of indexes that an Entity instance can have, which is quickly met when a full-text index is built on description fields of concepts and OWL-S descriptor files of uploaded web services.

We have encountered lengthy response times in two phases: (i) service publication, and (ii) service discovery. In the first phase, most of the time is spent during the population of ancestry relationships of concepts found in the published services. To overcome this problem, we have setup the popular open source caching product Ehcache [57], to cache the widely used SUMO concepts in memory. Also we have

setup indexes on OWL-S Class and OWL-S Thing entities for parent and child queries.

In the second phase, most of the processing time was spent during service category plane traversal. We are using a technique to reduce the service space categorically, in order to start the discovery process with a far more restricted set of service category planes. The details of this technique are described in section 6.3. In addition, we have used indexable concept keys in our category planes as data points, rather than conceptual word references, to perform faster matching during Ranking Service Matches.

However, the problems with service discovery did not quite end with the processing time optimizations. The Full Text (Service Descriptor) search feature in the Service Discovery Client relies on indexing of OWL-S and WSDL service descriptors, and creates a very loaded index on the web service entities stored in the data store. As of June 2008, Google App Engine put a 5.000 indexable words restriction on each entity instance [58]. This normally does not pose a problem for a majority of the applications, but in our application all five OWL-S descriptor files and the WSDL descriptor files are stored in the same entity, allowing the 5.000 indexable properties barrier to be easily broken by both standalone and cross property indexes.

To overcome this issue, we have used the Simple Full Text Search for App Engine project [59] which allows multiple search indexes to be built on different entities for the source entity, and joining these indexes at full text search time a technique named “Relation Index”.

4.4.3 Query Processing In a Peer to Peer Network

Peer to peer networks are widely used for sharing information. Peer to peer networks usually carry binary files including audio, video and software [24] as content. The main issue with these networks is, the peers in the network are not the original creators, or owners of the content, and the content is usually standardized. Also, a client in a peer to peer file sharing network knows what he is searching for (i.e. a

video clip from a football match, a song from an artist or a software developed by a certain company) when he sends a query to the network.

However in our system, there are certain differences from the traditional peer to peer networks. First of all, a majority of the peers in the network do not actually provide content; rather they consume content; because they are clients, not publishers. Secondly, the publisher peers in the system are either the creators, or the providers of the content they provide. And lastly, the client peers in the system probably do not know what they are searching for in the sense that peers in a file sharing peer to peer network do.

Therefore, the architecture of our peer to peer system should be different from a traditional peer to peer system. First of all, our system accepts that the service discovery clients need not act as Service Repository Peers. This is an accepted practice for two reasons: (i) service discovery clients tend to use the system for small amounts of time, to retrieve a service, and then exit the network; therefore assigning them a Service Repository Peer role would result in lots of offline Service Repository Peers in the network and (ii) answering service discovery requests is a costly process, and it is a task that should be undertaken by the publishers of the services.

In our system, only Service Publisher Peers can take the Service Repository Peer role, which supports query processing in a peer to peer network. However, we do not expect an excessive number of Service Repository Peers to exist in our system, and thus allow the Service Repository Peers to conform to the same standards as our root system does. In this way, a Service Discovery Client can knowingly or unknowingly connect to a Service Repository Peer and make use of the same functionality without noticing any difference. The same case is true for API developers, who can use the web services provided by the root system through the Service Repository Peers. The three possible operation scenarios are outlined in Figure 4-2, Figure 4-3 and Figure 4-4.

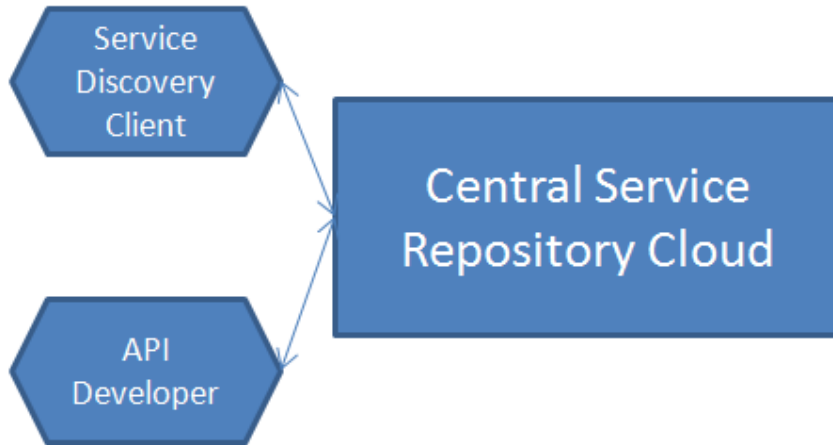


Figure 4-2 Direct Connection to Service Repository Cloud

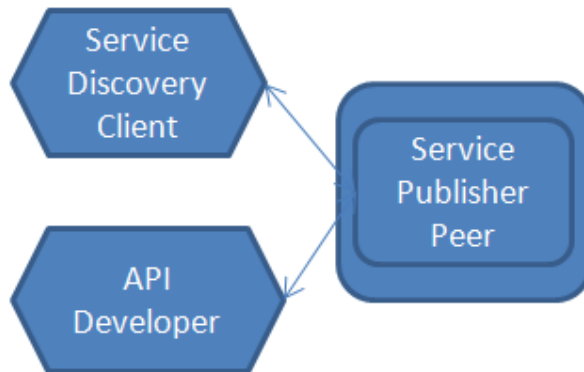


Figure 4-3 Direct Connection to Service Publisher Peer

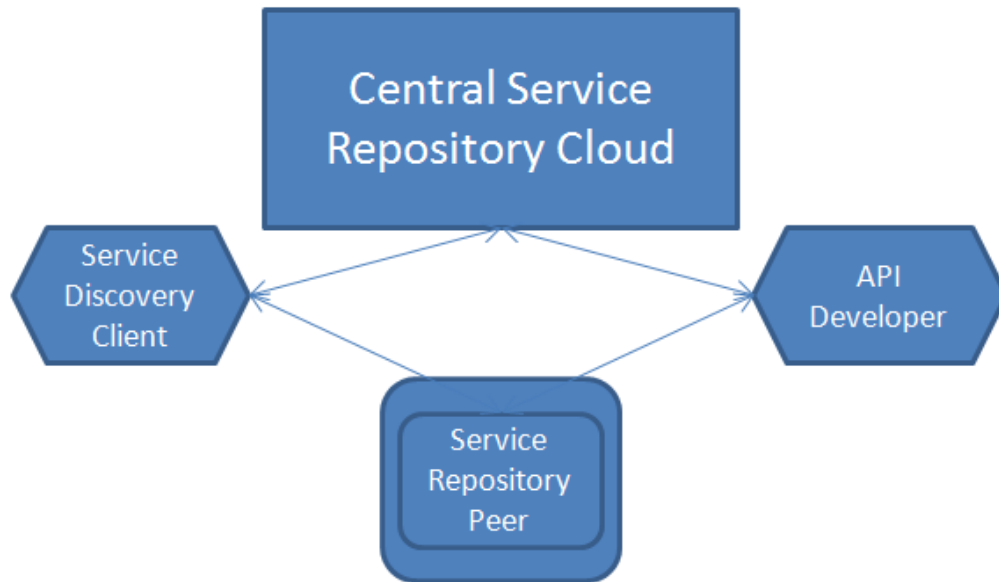


Figure 4-4 Central Repository Cloud Forwards to Service Repository Peer

The reason for such a strong super peer settlement is that, most of the service publishers publish a handful of web services and would not be willing to participate in the computation of a service discovery process taking into account both CPU and storage requirements. However, large sized service providers, or enterprises with numerous web services for their intranet would be willing to configure Service Repository Peers in their own network to make use of our system's services internally or externally. Therefore we expect participating peers to be servers with properly allocated resources of large sized organizations.

4.5 Service Discovery and Publishing API

Service oriented computing defines an architectural style whose goal is to achieve loose coupling among interacting software entities [60]. In this respect all of the

services described in sections 4.1, 4.2 and 4.3, are built as web services themselves. The service oriented approach of the architecture allows the system to be used by not only user agents with access to software developed throughout this thesis, but also solution developers who intend to use the system in their own software stack as a web service discovery solution.

The API provides services to lookup concepts over the unified ontology and to annotate the service descriptions prior to upload. Once the services are annotated, the API users can upload their services to the Service Repository Cloud, or update their services at any time.

API users can also search for services using the interfaces of the Service Discovery Manager. The Service Discovery Manager replies with proper XML formatted messages, which provide machine readable information (i.e. Service Match Score, Last Online Date, Service Rating) for the API user to programmatically decide which service to be selected. The information in the response also contains pointers to the service descriptor files in both WSDL and OWL-S formats. The API user can follow these pointers and obtain the service descriptors to either create a composite service or invoke the individual service automatically.

CHAPTER 5

SEMI-AUTOMATED SERVICE PUBLISHING

There are three types of semantic service publishing (i) manual service publishing, (ii) semi-automated service publishing and (iii) fully automated service publishing [13]. The service publishing procedure of the proposed system is a semi-automated system.

Manual service publishing methodologies allow the publisher to visit a web site, or use a software agent to manually upload previously generated semantic service descriptions [13,61]. This process is rather cumbersome for the present service publishers, because the descriptions of the services they provide are mostly in the WSDL format, which is currently the most prevalent service description language. The WSDL file at hand should be manually or with the assistance of an editor program converted to an OWL-S file, and should be annotated with ontological concepts from an available ontology. Later on the created semantic service descriptions are uploaded to a registry, with the explicit user effort. The process will recur, with the user going through the described steps for each change to the original service description.

Semi-automated service publishing techniques allow a smoother transition for the current service publishers [45]. In this technique certain tasks like WSDL to OWL-S conversion, service modification monitoring and service revision publishing are done by the software agent. However, the user agent still has to annotate the service descriptions created by the software agent, at required times. The advantage of this technique over a fully manual solution is that it takes much less time since the publisher does not have to handle the technical issues like service description conversion or service definition and upload.

Fully automated service publishing techniques try to both convert and annotate the service descriptions to achieve semantic service descriptors (i.e. annotated OWL-S files) from non-semantic service descriptors (i.e. WSDL files) [12]. These techniques mostly rely on text mining and natural language processing algorithms to understand and describe the provided services from service descriptors.

The main advantage of this approach is that it takes the service description migration cost from the shoulders of the publisher completely. The main disadvantage is, however, that the automatically generated concepts do not perform very well at discovery time. Thus the weight lifted from the service publisher's shoulders does not actually disappear; rather it lands on the client's shoulders.

Putting the performance of the query matching algorithms in the center spot, the system described in this thesis opts for the semi-automated service publishing approach. The steps and details of the approach are described in detail in this chapter.

5.1 Publisher Identification and Service Selection

The first step in automated service publishing is how to correctly identify the publishers. This is an important issue, since publishers tend to develop and serve similar web services, and similar services share similar ontological categories and concepts. The semantic information extracted from a service published by a given

publisher can be used in the semantic annotation of (or query matching for) another service from the same publisher.

UDDI, for instance, supports a variety of authentication mechanisms [4] including username/password pairs and trust relationships via operating system credentials. However in most of the peer to peer networks, there is a rather simple identification mechanism which relies on the reuse of a randomly or orderly generated series of alphanumeric characters forming a hash key.

The disadvantage of the usage of hash keys is, when the users of the system setup the application on different clients, they get a new hash key and lose the link to their previous publications. Also, several hash key generation algorithms which rely on computer characteristics or temporal data (i.e. current time, available disk space etc.) naturally generate different hash keys when they are setup on the same client again and again.

In the proposed system, a hybrid approach which enables the users to obtain a one time generated hash key via a username and password pair is used. In this way, the publisher client can be setup on numerous machines with the same hash key, while the hash key is being managed via the Service Repository Cloud with the username and password of the user.

In order to increase the credibility of the username and password validation, the system does not introduce a new user account system; rather it relies on the popular identity stack Google Accounts [62]. In this way, users can sign up to the system as service publishers using their existing Google Accounts, and receive their publisher keys which will allow them to publish services from multiple Service Publisher Peers.

Once the publisher is identified by the Service Repository Cloud, the service publisher is allowed to select services to publish. In this step, the service publisher is asked to provide a file system or network directory containing the service descriptors describing the services to be published. The given directory is

recursively searched for supported service descriptors. At this time only WSDL files are supported for service publishing.

The set of complying service descriptors are listed in a selection window, for the publisher to choose from. The publisher can select the services to be published, or unselect the services not to be published.

5.2 WSDL to OWL-S Conversion

In the present web service environment, services are prevalently described by WSDL files. For this reason the system described in this thesis targets WSDL files as service descriptor sources. The main problem regarding WSDL files as service descriptors is their lack of semantic information, which is of utmost importance for automated service discovery.

This problem can be overcome by using OWL-S files as service descriptors, which enable the expression of semantic properties for the service. To use OWL-S files as service descriptors, the WSDL files should first be converted to OWL-S syntax.

The WSDL2OWL-S [63] library created by the Atlas team at Carnegie Mellon University is used for a preliminary WSDL to OWL-S conversion. The tool was selected since it is the most stable of the libraries performing WSDL to OWL-S conversion and it supports OWL-S 1.1 specification.

The main shortcoming of the WSDL2OWL-S library is that it cannot accurately represent multiple functionalities wrapped in a single WSDL file. This is mainly because WSDL does not provide process composition information for the provided functionality. However, the tool creates the basic structure of the service described with the WSDL file; therefore it is usable in our system.

The files created by the WSDL2OWL-S library include the OWL-S Concept, Grounding, Process Model and Service Profile files. The service is described in a

single OWL-S Service file with imports pointing to the aforementioned four descriptors.

The generated four files go through a further processing step to help the state of the art OWL-S parsers to parse the generated service descriptors. In this step the descriptor URLs and ontological references are updated. The descriptor URLs are mapped to the links of the form illustrated in Figure 5-1.

File Name	File URL
Main Service Descriptor File	<code>http://servicerepositoryhost:port/services/files/{publisherkey}-{servicename}Service.owl</code>
OWL-S Service Profile File	<code>http://srhost:port/services/files/{publisherkey}-{servicename}ServiceProfile.owl</code>
OWL-S Service Grounding File	<code>http://srhost:port/services/files/{publisherkey}-{servicename}Grounding.owl</code>
OWL-S Service Process Model File	<code>http://srhost:port/services/files/{publisherkey}-{servicename}ProcessModel.owl</code>
OWL-S Service Concept File	<code>http://srhost:port/services/files/{publisherkey}-{servicename}Concept.owl</code>

Figure 5-1 Descriptor URL Mappings for OWL-S Descriptors

In Figure 5-1 the variable `{servicename}` denotes the name of the service specified by the service publisher during service annotation. The variable `{publisherkey}` is the hash key the publisher obtains from the Service Repository Cloud during signup, which is used by the Service Publisher Peer at each service upload.

Only a single ontological entity is added to the Service Profile and Process Model files, since the system proposed in this thesis uses a unified ontology thus all ontological concepts point to the same ontology.

```
<!ENTITY wsones "http://srhost:port/services/ontology/wsones.owl">
```

Figure 5-2 Service Repository Cloud Ontology Reference

5.3 Service Descriptor Annotation and Main Category Selection

Once the WSDL to OWL-S conversion is completed, the Service Profile file generated by the system is presented to the user for annotation in an easy to use editor that allows further OWL-S annotation. The user can select the nodes to annotate and use the unified ontology to annotate the given nodes.

The OWL-S annotation editor parses the Service Profile file, and identifies the annotatable nodes in the OWL-S document. The annotatable nodes currently include the Input, Output, Precondition and Effect types. The number of annotatable nodes can be increased in the future.

The default types of most of the annotatable nodes come from the XML Schema [64] type “String”, and is usually denoted with “&xsd;#String” in the OWL-S grammar. A sample translation of a simple type is shown in Figure 5-3.

WSDL Definition	<pre> <portType name="XMethodsQuerySoapPortType"> <operation name="getServiceSummariesByPublisher" parameterOrder="publisherID"> </operation> </portType> <message name="getServiceSummariesByPublisher0SoapIn"> <part name="publisherID" type="xsd:string"/> </message> </pre>
OWL-S Translation	<pre> <profile:hasInput> <process:Input rdf:ID="XmethodsQuerySoapPortType_getServiceSummariesByPublishe r_publisherID_IN"> <process:parameterType rdf:datatype="&xsd:anyURI"> &xsd:string </process:parameterType> </process:Input> </profile:hasInput> </pre>

Figure 5-3 OWL-S Translation of a Simple Type in WSDL

However there are certain exceptions to this, since the WSDL2OWL-S tool can also parse the complex types defined locally in WSDL files; however these types mostly do not carry any semantic information, and even in the cases that they do, an ontological/semantic mapping is necessary to make use of these definitions. A sample translation of a complex type can be found in Figure 5-4.

WSDL Definition	<pre> <complexType name="ServiceSummary"> <sequence> <element name="name" nillable="true" type="xsd:string"/> <element name="shortDescription" nillable="true" type="xsd:string"/> <element name="wsdlURL" nillable="true" type="xsd:string"/> <element name="publisherID" nillable="true" type="xsd:string"/> </sequence> </complexType> <complexType name="ArrayOfServiceSummary"> <complexContent> <restriction base="soapenc:Array"> <attribute ref="soapenc:arrayType" wsdl:arrayType="tns:ServiceSummary[]"/> </restriction> </complexContent> </complexType> </pre>
OWL-S Translation	<pre> <profile:hasOutput> <process:Output rdf:ID="XMethodsQuerySoapPortType_getServiceSummariesByPublisher _Result_OUT"> <process:parameterType rdf:datatype="&xsd:anyURI"> &concept;#ArrayOfServiceSummary </process:parameterType> </process:Output> </profile:hasOutput> </pre>

Figure 5-4 OWL-S Translation of a Complex Type in WSDL

Our system uses a unified ontology provided by the Suggested Upper Merged Ontology (SUMO) team. The ontology contains several mid-level ontologies and has relational links to widely used lexical database WordNet. The methodology used for importing the ontology to our Data Store is discussed in section 5.4.

The Concept Auto Complete Widget allows the service publishers to find the concepts they are searching to annotate the nodes. As they type, the widget sends queries to the Service Repository Cloud containing the characters typed. The Service Repository Cloud answers the query with a set of concepts containing the typed words, or relating to the words containing the typed words.

The publisher can then select from the list of returned words, or continue typing to refine the received results. Once the publisher is satisfied with the available concept, he can simply select the concept and assign it to the active node as an annotation.

After annotating the appropriate nodes in the Service Profile, the publisher can specify a main category for the service to be published. There is a predefined set of main categories from which the publisher can select. The main categories are derived from the mid-level ontologies contained in the SUMO and are shown in Figure 5-5.

Main Category	Mid-level SUMO Ontology
Biology	Viruses
Communications	Communications
Computing	Computing, Distributed Computing
Economy	Economy
Engineering	Engineering, Engineering Components
Finance	Finance
Geography	Countries and Regions, Geography, World Airports A-Z
Generic	Physical Elements, North American Industrial Classification System
Government	Government, Transnational Issues
Military	Military, WMD
People	People
Transportation	Transportation

Figure 5-5 Main Categories in the Service Repository Cloud

The selection of a main category for the service is crucial for the automated semantic service discovery since it helps divide the search space significantly. The details of using the main category are discussed in section 5.7.

5.4 Representing of SUMO Concepts

The ontology provided by SUMO team contains links to a separate WordNet ontology in OWL-S format, again created by the SUMO team. Our system uses both of these ontologies for service publishing and annotation purposes. However, directly using ontologies for user requests requires a tremendous amount of processing power, due to the large sizes of RDF data present in these two ontologies.

Therefore, our system imports the OWL-S formatted data to a relational model. Also, the concepts and classes in the ontologies are enriched with semantic data computed by our system to help optimize our service matching algorithms. This section explains the details of the data model present in our system.

5.4.1 Concepts, Classes and WordNet Words

There are three main types of data in the two ontologies SUMO provides. First type is a SUMO Concept, and it is an instance of a SUMO Class in the real world. A SUMO Concept is represented as shown in Figure 5-6 in the SUMO ontology.

```
<owl:Thing rdf:ID="MomaAirport">  
<rdf:type rdf:resource="#Airport"/>  
<abbreviation rdf:datatype="xsd:string">MMW</abbreviation>  
<located rdf:resource="#Mozambique" />  
</owl:Thing>
```

Figure 5-6 A Sample SUMO Concept

In the example, Moma Airport is an instance of SUMO Class “Airport”, and is located at SUMO Class “Mozambique”. SUMO Concepts contain numerous relations, “located” being one of them. These relations are carried to our data store and stored with concepts, for future use in ancestry relationships, full text search, and data point enhancement of service discovery queries. The list of relations that is saved to our data store are shown in Figure 5-7.

Relationship	Example
type	<owl:Thing rdf:ID="NetherlandsAntilles"> <rdf:type rdf:resource="#LandArea"/>
subsumingRelation	<owl:Thing rdf:ID="Above"> <subsumingRelation rdf:resource="wn#WN30-301208044"/>
equivalenceRelation	<owl:Thing rdf:ID="Mongolia"> <equivalenceRelation rdf:resource="wn#WN30-108968879"/>
instanceRelation	<owl:Thing rdf:ID="NorthSea"> <instanceRelation rdf:resource="wn#WN30-109374036"/>
located	<owl:Thing rdf:ID="MoodyAirForceBaseGAAirport"> <located rdf:resource="#UnitedStates" />
abbreviation	<owl:Thing rdf:ID="MoolawatanaSouthAustraliaAirport"> <abbreviation rdf:datatype="xsd:string">MSA</abbreviation>

Figure 5-7 SUMO Concept Structure

The SUMO Concept hierarchy also contains some domain specific relationships (i.e. atomicNumber, boilingPoint, economyType etc.); however these relationships are discarded during the imports in our system, to reach a more generic conceptual approach.

Additionally, as it can be easily noticed from the above relationship examples, while some of the relationships in the examples are pointing to SUMO Classes and SUMO Concepts within the same ontology as textual ids, some relationships point to SUMO's WordNet ontology, with WordNet 3.0 identifiers (i.e. wn#WN30-108968879).

The second type of data used by the SUMO ontology is SUMO Classes, which are united at the top most SUMO Class "Entity". The SUMO Classes are generated from the aforementioned mid-level ontologies, and they mostly represent concepts,

rather than instances of concepts. The parent child relations contained in the SUMO Classes are extremely important for our system, since most of the conceptual data point derivation is done using these relationships. The details of the SUMO Class structure can be found in Figure 5-8.

Relationship	Example
subClassOf	<code><owl:Class rdf:ID="NetworkAdapter"> <rdfs:subClassOf rdf:resource="#ComputerComponent"/></code>
equivalenceRelation	<code><owl:Class rdf:ID="Neutron"> <equivalenceRelation rdf:resource="wn#WN30-109369520"/></code>
subsumingRelation	<code><owl:Class rdf:ID="Monday"> <subsumingRelation rdf:resource="wn#WN30-115182402"/></code>
instanceRelation	<code><owl:Class rdf:ID="AgeGroup"> <instanceRelation rdf:resource="wn#WN30-108369615"/></code>
disjointWith	<code><owl:Class rdf:ID="AnaerobicExerciseDevice"> <owl:disjointWith rdf:resource="#AerobicExerciseDevice" /></code>

Figure 5-8 SUMO Class Structure

Like the SUMO Concepts, SUMO Classes also contain a variety of application and domain specific properties (i.e. externalImage, axiom, lethalDose etc.), which our system discards during OWL-S to Data Store mapping. The remaining properties are

mapped as one to many relationships, in which a class can have multiple outgoing or incoming relationship links of each type.

The third type of data the SUMO ontology uses is WordNet Words, extracted from version 3.0 of WordNet lexical database. The relations imported to our data store for WordNet words can be found in Figure 5-9.

Relationship	Example
hypernym	<hypernym rdf:resource="#WN30-107775905"/>
partHolonym	<part-holonym rdf:resource="#WN30-102644665"/>
attribute	<attribute rdf:resource="#WN30-102644665"/>
memberHolonym	<member-holonym rdf:resource="#WN30-102593863"/>
pertainym	<pertainym rdf:resource="#WN30-115139849"/>
hyponym	<hyponym rdf:resource="#WN30-108273406"/>
memberMeronym	<member-meronym rdf:resource="#WN30-109964202"/>
partMeronym	<part-meronym rdf:resource="#WN30-102644665"/>
similarTo	<similar-to rdf:resource="#WN30-301499999"/>
antonym	<antonym rdf:resource="#WN30-301500766"/>
derivationallyRelated	<derivationally-related rdf:resource="#WN30-100098385"/>
domainRegion	<domain-region rdf:resource="#WN30-108740875"/>
domainUsage	<domain-usage rdf:resource="#WN30-107075172"/>
domainTopic	<domain-topic rdf:resource="#WN30-108199025"/>
synset	<synset rdf:resource="#WN30-108199025"/>

Figure 5-9 WordNet Concept Structure

Although the entire set of WordNet word relations are imported to our system, most of the relations are not used in the service discovery procedure. As outlined in 5.7.1.1, only relationships resembling ancestry links are used in our system.

5.4.2 Concept Plane Categorization

All SUMO concepts, SUMO classes and WordNet words in our data store have a concept plane. This concept plane is used to determine which conceptual plane will be used when a data point is to be included in the conceptual space of a web service. The use of concept planes are discussed in section 5.7.

The categorization of the semantic entities is made based on the root nodes in mid-level ontologies provided, just after ontology import procedure is completed. A recursive algorithm traverses all the nodes in the data store to label each node with a category plane. The details of the algorithm are listed in Figure 5-10.

1. For each mid-level ontology
 - a. For each root category node
 1. Add the node to a category - expansion bag
 - i. While the category expansion bag is not empty
 1. Pop a node from the expansion bag
 - a. If the node is uncategorized, update its category to parent category
 - b. If the node is categorized, duplicate the node, break the original node's link to the current parent, break the duplicate node's link to any other parent and categorize the duplicate node
 2. Find all the nodes having the current node as the parent node
 - a. For SUMO Concepts and Classes use the subClassOf relation
 - b. For WordNet Words use the hypernym relation
 3. Add the retrieved nodes to the category - expansion bag

Figure 5-10 Concept Plane Categorization Algorithm

With this categorization algorithm, each semantic node in the Service Repository Cloud is assigned a root category, to be used in Service Category Plane Generation and Query Parsing and Categorization phases.

5.4.3 Concept Node Leveling

In this phase, each semantic node in the Service Repository Cloud is assigned a numeric level information, to be used in Service Match ranking. This process particularly takes place after Concept Plane Categorization, because the Concept Plane Categorization algorithm duplicates nodes that can be reached from multiple

categories, and removes the relationship links connecting the nodes to different mid level ontologies.

During this phase a recursive algorithm starts with an initial set of mid-level ontology root nodes, and traverses downwards through the Service Repository Ontology via parent-child relations to assign levels to each of the semantic nodes. The algorithm is described in Figure 5-11.

1. For each mid-level ontology
 - a. For each root category node
 1. Add the node to a category - expansion bag, set current level to 1
 - i. While the category expansion bag is not empty
 1. For each node in the expansion bag
 - a. Pop a node from the expansion bag, level the node with the current level
 - b. Find the children of the node, add them to a reserve expansion bag
 2. Contents of the reserve expansion bag are moved to the expansion bag, current level is increased by 1

Figure 5-11 Concept Node Leveling Algorithm

5.5 Service Description Upload

There are a number of entities to be uploaded regarding a service when the publisher is finished with selecting, converting and annotating a web services, including the WSDL descriptor file, OWL-S descriptor files, service name, service main category and the publisher hash key.

In this scheme, the publisher hash key identifies the owner of the web service, the OWL-S descriptor files and the service main category enable semantic discovery capabilities, and the WSDL descriptor files are required for service clients explicitly requesting WSDL technology rather than OWL-S for connection initiation.

The upload is made to the Service Repository Cloud previously registered to, and it is repeated automatically when the service's WSDL descriptor file is modified. Re-annotation is optional to the service publisher when revisions of the service are created. This is solely because a majority of the modifications to web service descriptions are regarding grounding information, rather than conceptual description of the service.

5.6 Service Description Extraction

When the service description is uploaded to the Service Repository Cloud, the system processes the service description for useful information extraction. The process information, grounding information or concept information local to the service are not extracted and stored separately. Rather, these are stored in the Data Store as OWL-S files, and they are provided to service clients upon request.

The profile information including Input, Output, Precondition and Effect (IOPE) variables are extracted and stored separately. Additionally, the service name and service description are extracted and stored in the Data Store.

OWL-S Service Profile documents contain Input, Output, Precondition and Effect information for the processes described in the service documents. The nodes describing IOPE information are defined as triples [10], including the type of the node (i.e. Input, Output, Precondition or Effect), name of the node (i.e. name of the input) and the type of the input (i.e. a publisher selected SUMO concept). The extracted information from an OWL-S file is illustrated in Figure 5-12.

OWL-S Representation	Extracted Information
<pre><process:Input rdf:ID="carPlateNumber"> <process:parameterType rdf:datatype="&xsd:anyURI"> &wsones;#platenumber </process:parameterType> </process:Input></pre>	<pre>IOPE Type: Input IOPE ID: carPlateNumber Input Type: platenumber</pre>
<pre><process:Output rdf:ID="fineAmount"> <process:parameterType rdf:datatype="&xsd:anyURI"> &concept;#price </process:parameterType> </process:Output></pre>	<pre>IOPE Type: Output IOPE ID: fineAmount Output Type: #price</pre>
<pre><process:hasPrecondition rdf:resource="#carPlateNumber "> <expr:SWRL-Condition rdf:ID="carPlateNumber"> <swrl:propertyPredicate rdf:resource="&concept;# ISOPlateNumberStandard"/> .. </process:hasPrecondition></pre>	<pre>IOPE Type: Precondition IOPE ID: carPlateNumber Precondition Type: #ISOPlateNumberStandard</pre>
<pre><process:hasEffect> <expression:SWRL-Condition rdf:ID="carFinePayment"> <expression:expressionBody rdf:parseType="Literal"> <swrl:classPredicate rdf:resource="&wsones;#payment" /> ... </expression:expressionBody> </expression:SWRL-Condition> </process:hasEffect></pre>	<pre>IOPE Type: Effect IOPE ID: carFinePayment Effect Type: #payment</pre>

Figure 5-12 IOPE Information Extracted from OWL-S Descriptors

The uploaded service description includes a service main category selected by the publisher from a list of predefined categories described in section 5.3. Along with this information, the service name and description are stored with the service. The service name and description are also extracted from the OWL-S file, and they are illustrated in Figure 5-13.

OWL-S Representation	Extracted Information
<pre data-bbox="354 751 646 884"><profile:serviceName> Car Fine Payment Service </profile:serviceName></pre>	<p data-bbox="873 804 1333 835">Service Name: Car Fine Payment Service</p>
<pre data-bbox="354 907 847 1136"><profile:textDescription> Using this web service, you can view the unpaid fines registered to the plate number of your car, and make payments for the relevant fines. </profile:textDescription></pre>	<p data-bbox="873 959 1367 1094">Service Description: Using this web service, you can view the unpaid fines registered to the plate number of your car, and make payments for the relevant fines.</p>

Figure 5-13 Service Name and Description Extracted from OWL-S Descriptors

5.7 Category Planes and Main Categories

Our system heavily depends on category planes generated in a Hilbert Space, which contain conceptual data points extracted from the service descriptions. When a service is uploaded, a Hilbert Space is generated for the service, and the space is used for service and query match score computation during web service discovery.

There are an undefined number of dimensions in the created complex space for each uploaded service, limited by the categories defining the service. In the current system there are a limited number of categories, however future work might change the situation. Even though the number of categories is limited, the number of dimensions defining a service is still unknown because planes are computed based on the information extracted from the service descriptions.

The complex space defining the web service is also a dynamic space, which is open to the addition of new dimensions and conceptual data points with the user feedback.

The computation of the Hilbert Space for the uploaded web services are discussed in section 5.7.1. Section 5.7.2 discusses the importance of each dimension in the Hilbert Space to the uploaded web service. Section 5.7.3 discusses how the distances between the category planes in the space are measured and used. Section 5.7.4 discusses how the Hilbert Space of the web service evolves within time by the help of user feedback.

5.7.1 Service Category Plane Generation

After the extraction of IOPE information and Service Name, Service Description and Service Main Category data, the Service Repository Cloud starts data cleaning and processing phase. There are two types of data that needs to be processed: (i) conceptual data (i.e. type information of IOPE and Service Main Category) and (ii) semantic, non-conceptual data (i.e. id information of IOPE, Service Name and Service Description).

5.7.1.1 Data Points Regarding Conceptual Data

Since the IOPE type information is selected from the unified ontology of the Service Repository Cloud during the service publishing phase, there is no need to search for the meanings of the type values, rather they can be located in the unified ontology directly as data points. Each ontology concept has a pre-assigned category plane; therefore each addition of a data point from a different category brings a new

category plane to the complex space being created. Intuitively, concepts within the same category plane are added to the same concept plane.

However, usually the data points accumulated by the IOPE types are not enough to accurately annotate a service, due to their low number. Most of the time a user searches for a service using a child or parent of a concept, which would result in a miss in this case. Therefore, the categorical spaces are enriched with concepts by other concepts that in one way relate to the already present concepts.

There are a number of relations between the concepts, classes and words in the SUMO library, and these relations and their mappings in our system are explained in detail in section 5.4. The primary relations used in categorical space enrichment are parent-child relationships. In the OWL-S grammar and WordNet hierarchy, these relationships are denoted with several different keywords, outlined in Figure 5-14.

Relation	Grammar	Example	Mapping
Type	OWL-S	<rdf:type rdf:resource="#Airport"/>	Instance
Sub Class Of	OWL-S	<rdfs:subClassOf rdf:resource="#AirLaunchMissile"/>	Parent
Equivalence Relation	OWL-S	<equivalenceRelation rdf:resource="wn#WN30-102693413"/>	Instance
Hypernym	WordNet	<hypernym rdf:resource="#WN30-105269901"/>	Parent
Hyponym	WordNet	<hyponym rdf:resource="#WN30-101054545"/>	Child

Figure 5-14 SUMO and WordNet Ontology Mappings

The relations shown in Figure 5-14 are followed to add more data points to the categorical concept planes, to enable a wider matching capability. However, this introduces a new problem, since in a scheme where every data point has the same value, services in the lower and upper categories will have very similar match scores for a certain query. This happens because our system not only follows upward links (i.e. parent relationship) in the hierarchy, but also it follows the downward links (i.e. child relationship) and parallel links (i.e. instance relationships).

Therefore the data points in the categorical planes need an importance value to further distinguish the difference between services. Taking this issue into account, the generated categorical planes are designed to have concept IDs in the X axis, and weight values in the Y axis. A sample category plane is shown in Figure 5-15.

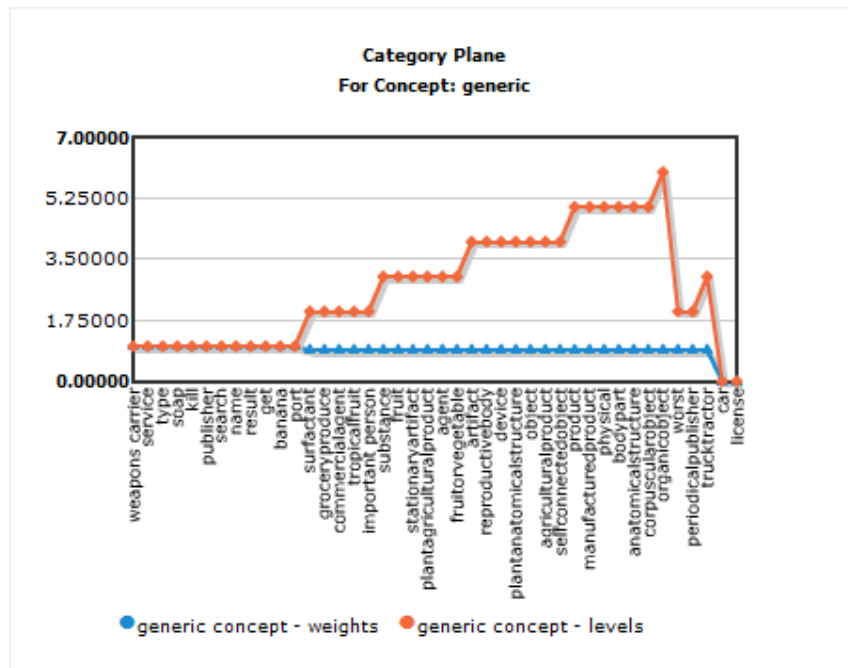


Figure 5-15 A Sample Category Plane Generated by Service Repository Cloud

The computation of the weight values for data points is extremely important for the accuracy of query responses. For our system, the most heavy data point is the original data point received from the type identifier of an IOPE variable. The weight decreases gradually as both the parent and child relationships are followed. This means that, the direct parent of a concept has a higher weight than the parent of a direct parent, and the same goes for the child relations too. Figure 5-16 outlines a sample parent child relationship, for the concept “Canal System” In the given example, the concept “Canal System” carries the maximum weight, and the concept “Entity” carries the minimum weight as shown in Figure 5-17.

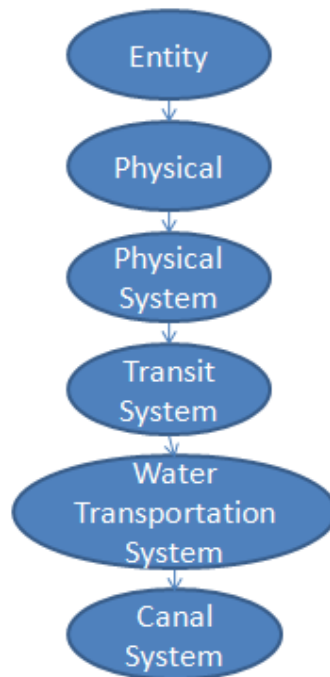


Figure 5-16 A Sample Parent Child Relationship

weight(Canal System) > weight(Water Transportation System) > weight(Transit System) > weight(Physical System) > weight(Physical) > weight(Entity)

Figure 5-17 Weights for a Sample Parent Child Relationship

The described weighing scheme performs well in conditions where the original concepts do not have overlapping ancestors or successors. When ancestors or successors overlap, a misrepresentation may occur for two reasons: (i) even though a concept has a high weight since it is an original data point, the weight of it might be lowered by a child when links are being followed and (ii) a concept lying on the ancestry pathway of more than one original data point might have the same weight as a concept lying on the ancestry pathway of a single data point.

For instance, a web service descriptor can include both “Water Transportation System” and “Canal System” concepts as original data points. For the given service the weight of “Water Transportation System” will be high since it is an original data point. However, since “Water Transportation System” concept is also the parent of an original data point (i.e. “Canal System”), its weight might be decreased.

To prevent these problems, the plane generation algorithm is implemented in an incremental fashion as shown in Figure 5-18.

1. All the original data points from IOPE types are added to the categorical plane they belong to
2. For each category plane
 - a. For every original data point (N) on the category plane
 - i. Parent concepts are followed until the root concept is reached
 - ii. Child concepts are followed until the child relationship has an empty value
 - iii. For every parent and child concept (M) obtained
 1. If the data point at hand (M) is not present on the current category plane, it is added to the category plane, and assigned a weight value based on its distance (i.e. number of links followed upwards or downwards) to the original data point
 2. If the data point at hand (M) is already present on the current category plane as a data point (K)
 - a. If the data point present on the plane (K) is an original data point, its weight value is incremented based on the distance (i.e. number of links followed upwards or downwards) between the data point at hand (M) and the original data point (N), given its weight is not more than Maximum Original Data Point Weight
 - b. If the data point present on the plane (K) is placed via an ancestry relationship, its weight value is incremented based on the distance (i.e. number of links followed upwards or downwards) between the data point at hand (M) and the original data point (N), given its weight is not more than Maximum Ancestry Data Point Weight

Figure 5-18 Service Repository Cloud Plane Generation Algorithm

As seen in Figure 5-18, data points are categorized, and certain weight limitations are applied to the data points to create a balanced and feasible plane generation algorithm. The definitions of the weight limitations are given in Figure 5-19. The exemplary values for the weight limitations are provided Appendix A.

Original Data Point	A data point which represents a concept in the service description.
Ancestry Data Point	A data point which is obtained via following parent-child relationships of an original data point.
Maximum Original Data Point Weight	The maximum weight that can be assigned to a data point which originates from the service's description.
Maximum Ancestry Data Point Weight	The maximum weight that can be assigned to a data point, which has been obtained via following the parents or children of an original data point.

Figure 5-19 Weight Limitation Descriptions

5.7.1.2 Data Points Regarding Non-Conceptual Data

Categorical planes are filled with numerous data points of varying weights when the data points regarding the conceptual data are processed. However, there are still a number of semantic information items waiting to be processed, which might be of use in the service discovery process. These items include IOPE ID, Service Name and Service Description.

Most of the automated service publishing tools make use of these information bits and pieces to label and categorize the web services. Even though the information contained in these non-conceptual items are not entirely capable of performing

accurate service discovery, completed with conceptual data they can increase the number of query hits for relevant services.

There are two important issues when adding non-conceptual data derived data points to a category plane: (i) the weight values of the data points and (ii) the categories of the data points. When there are a lot of words in the description, this might scatter the categorical alignment of the complex space of the service. Similarly, treating non-conceptual data derived data points as valuable as conceptual data points might decrease our chances of distinguishing similar web services from each other. For instance, most of the service descriptions include words with little semantic value such as “this”, “a”, “the”, “with” etc.

Therefore data point selection and weight distribution should be done even more carefully in the non-conceptual data set. For this purpose our system takes certain precautions in its non-conceptual data point selection algorithm as described in Figure 5-20.

1. The three non-conceptual data sources (i.e. IOPE ID, Service Name and Service Description) are cleaned and tokenized to words, and added to a word bag
2. The word bag and the Data Store, including WordNet links are cross checked to see if there are any records for the given words
 - a. If there are records for the word, word is added to a whitelist
 - b. If there are no records for the word in the system, word is added to a blacklist
3. For each word in the white list
 - a. Parent concepts are followed until the root concept is reached
 - b. Child concepts are followed until the child relationship has an empty value
 - c. The followed concepts are added to a concept whitelist, and assigned weight values based on their distance from the original non-conceptual data points
4. Concepts in the concept whitelist are sorted based on their weights in decreasing order
5. For each concept in the concept whitelist
 - a. If there is a categorical plane present for the concept
 - i. If the current concept does not exist on the plane, the concept is added to the plane, given that the number of non-conceptual data points do not surpass the number of conceptual data points on the plane ¹
 - ii. If the current concept exists on the plane, the concepts weight value is increased, given that its weight value does not surpass the weight cap it is subject to ²
 - b. If there is no categorical plane present for the concept, the category plane is generated and the concept is added to the plane, but the plane is marked as Non-Conceptual Data Only. This data will later be used in the Dimension Significance Computation.

Figure 5-20 Non-Conceptual Data Point Selection

¹ If there are n conceptual data points on a given conceptual plane, there can be as many as n non-conceptual data points on that plane.

² The weight cap for an original conceptual data point is Maximum Original Data Point Weight, where as the weight cap for a data point added via an ancestry relationship is Maximum Ancestry Data Point Weight.

5.7.2 Dimension Significance

There are multiple dimensions in the complex space computed for a web service, and all of these dimensions contribute to the service discovery procedure when queries are received by the Service Repository Cloud. However, not all dimensions are as important as each other, since there are certain categories that are fundamental to the service, and some categories that are simply there because of the non-conceptual data extraction system.

Therefore, our system computes a dimension significance score for each dimension, and includes this score in the discovery process to make sure that the data points generated in the categories that are important to the web service has more impact in a service discovery. More details are discussed in section 6.3. The dimension significance score is computed with the algorithm outlined in Figure 5-21.

1. For each category plane
 - a. For each data point in the category plane,
 - i. Increase the dimension significance of the category plane with IOPE Significance Amount³ if the data point equals to the type of an Input, Output, Precondition or Effect
 - ii. Increase the dimension significance of the category plane with Service Name Significance Amount if the data point concept is one of the words in the Service Name
 - b. If the category plane's category is equal to Service Main Category, increase the dimension significance of the category plane with Service Main Category Significance Amount

Figure 5-21 Dimension Significance Computation Algorithm

³ Exemplary IOPE Significance Amount, Service Name Significance Amount and Service Main Category Significance Amount values are provided in Appendix A

5.7.3 Category Plane Similarity Measures

There are multiple dimensions and category planes in the complex category space generated for the web services. However, an explicit relationship is not present between these categories. The distances between planes cannot be intuitively computed since the data points are composed of words and weights; and their alignment on the X axis is solely used to represent them better visually.

This introduces a problem since inter-category plane relationships play an important role on web service discovery at most times. When a complex space generated for the query does not have any corresponding dimension in the service space, still a selection has to be made among the services to decide which service is more similar to the query. This requires a notion of inter-category relationships.

In most of the service discovery approaches, the distance between any two concepts is measured via the tree distance in the ontology disregarding the categorical location of the concept. The same can be done in our system, by either creating a tree structure in the Data Store, or giving each concept node a level information starting from the categorical plane's root node.

However, this approach does not actually take similarity between categories into account. In this case, a node X with level 2 in the Economy domain will have the same similarity to a node Y with level 2 in the Finance domain and a node Z with level 2 in the Health domain whereas X should clearly be closer to Y and farther to Z.

In order to overcome these problems, during the initialization of the system ontology, our system computes a categorical plane distance matrix where a similarity score is computed between each category plane and stores it permanently. The computed similarity score is a symmetric score, i.e. for planes X and Y, the similarity function S is defined as: $S(X, Y) = S(Y, X)$.

The algorithm for the computation of the category plane similarity score is similar to Google's Page Rank [65] algorithm, in which links between two entities bond the entities together. There are two types of links for a certain concept: (i) incoming links and (ii) outgoing links. A link is an incoming link for a node if the source node is different, and a link is an outgoing for a node if the source node is itself. If a concept A contains a certain relationship to a concept B, the relationship is an outgoing link for concept A and an incoming link for concept B. The details of the Category Plane Similarity Measure algorithm are shown in Figure 5-22.

1. For each main category
 - a. All the concepts are retrieved, and all the non-empty relations (see Figure 5-14) of the concepts are counted
 - b. For each outgoing relation in the main category, the outgoing similarity score between the source main category and the target main category is incremented by the fraction of $1/(\text{total outgoing relations})$
2. For each main category
 - a. All the concepts are retrieved, and all the concepts from and to the concepts in the main category are counted
 - b. The incoming similarity score of the source main category (i.e. the outgoing similarity score of the target main category) is divided by the total number of incoming relations
3. For each category similarity
 - a. The incoming and outgoing similarity scores are added

Figure 5-22 Category Plane Similarity Measure Algorithm

With the described algorithm, a similarity score between category planes can be computed, which allow us to calculate distances between inter-category plane data

points. However, to accurately measure the data point difference, plane distance is not adequate, and we still need to evaluate the distances between the individual concepts in a tree-distance kind measure.

In our system, we do not form a tree to represent the unified ontology, but we provide level information for all the concepts in our categories. In a traditional service discovery environment this might not be suitable, because services are annotated with words, and there are multiple categories a word can belong to, since a word can have multiple senses. But in our architecture, the annotation is made directly via concepts, and separate concepts are provided for each sense of the concept, therefore the parent of a concept necessarily resides on the same categorical plane with the concept itself. So as long as the concepts are not in the same plane, their distance can be measured by adding their distance to their conceptual parents (i.e. categorical root concept).

Therefore, with the ontology initialization process, our system recursively labels each concept in our ontology with level information for future use in service discovery. More details on how the level information and plane similarity is used in service ranking can be found in section 6.3.

5.7.4 Space Modification with User Feedback

In the real world, content publishers rarely provide enough informative data to locate a provided service or product semantically. Actually, the lack of semantic data persuaded most of the search engine developers to look for alternative ways to semanticize a given content.

There are several techniques commonly used to enrich the semanticity of a given entity with user feedback, including but not limited to: (i) user comments, (ii) content tagging and labeling, (iii) referrals and trackbacks and (iv) content viewing and leaving counts and durations [66].

In our system, the most important metric that can contribute to a service's semanticity is user feedback. However, we believe that service discovery clients will not be willing to provide ratings, tagging and labeling services or commenting about publishers since they will be trying to finish their discovery process as soon as possible. The situation gets even worse when the service discovery client is not a user, but a software agent using the Service Discovery and Publishing API.

Therefore, our system makes use of query and service download data following the query to modify the service category space. When a query is followed with a download this contributes to the category space as increased data point weights, new data points or even new category planes.

The contribution is achieved by adding the data points in the categorical planes of the complex space generated for the query by the Service Repository Cloud to the original category space of the service that has been chosen by the service discovery client based following the query.

In environments where the end users of a search system can contribute to the search results, certain users try to manipulate the search results to obtain better rankings. Common manipulation techniques include duplicate voting for a result, multiple selection of a given resource and linking to one's own service. In our system, a precaution is taken to lower the harm that can be done by these type of users. To prevent misuse by the service discovery clients, the weights of the data points received from the query are decreased by a User Feedback Weight factor. The total weight that can be contributed by the user feedback is also limited to a User Feedback Cap amount, to prevent services from moving away from the original conceptual location the service publisher intended to place the service to. Exemplary values used in our system for the User Feedback Weight and User Feedback Cap are provided in Appendix A.

CHAPTER 6

AUTOMATED SEMANTIC SERVICE DISCOVERY

In our system both user agents and software agents can perform service discovery, and service discovery can be made in a fully automated manner. The agent's responsibility is to generate a query in the format that Service Repository Cloud understands, and post the generated query string using the query web service. An agent can create the query using the Service Discovery Client or the Service Discovery and Publishing API.

The Service Repository Cloud receives, parses and categorizes the query. Similar to categorical space generation for the uploaded web services, a conceptual space is generated for the received query too. Web services are searched in the web service repository based on the query category and they are ranked based on the query's and services' concept space. A Service Match Record is created for each of the evaluated services, and a sorted list of Service Match Records is returned to the agent as a machine interpretable web service response.

The agent can decide based on the Match Score, service details or service OWL-S files referred in the Service Match Record of the returned services to download service descriptor files in OWL-S or WSDL files, and follow the links in the Service

Match Record accordingly. This chapter elaborates on the details of the steps of the automated semantic service discovery of our system.

6.1 Query Generation

The query for a service discovery can be generated by: (i) a user-agent or (ii) a software-agent. The data to be provided in a query by each agent is the same: a number of query criterion and a filter operator. The generated query is transmitted to the Service Repository Cloud using the query web service by both of the agents, and is a text of the form shown in Figure 6-1.

Filter-Operator(criteria₁=value₁|criteria₂=value₂...|criteria_n=value_n)

Figure 6-1 Service Discovery Query Representation

The steps for query generation are outlined below for each actor.

6.1.1 Query Generation by a User Agent

The Service Discovery Client provides a user interface to generate queries consisting of multiple criteria. The client provides only equality operator for each criterion, since our system works on a positively incremental match score computation algorithm.

A criterion can be about the Input, Output, Precondition, Effect, Service Name or Service Descriptor Text (Full Text) of the service. For the first four service properties, the user interface mandates values generated using the Concept Auto

Complete Widget, which allows the user to specify a concept in the unified ontology of the Service Repository Cloud. For the latter two properties, the widget is still provided, however the user can use words not present in the repository as well.

The criteria generated by the user should be joined with a filter operator (i.e. AND or OR), and the user can select this filter operator from a combobox. The details of the functionalities of these filter operators are discussed in detail in section 6.1.4.

After the specification of the query criteria, and selection of the filter operator, the user-agent can submit the query to the Service Repository Cloud by clicking a button on the user interface of Service Discovery Client application. An example query generated by a user agent is shown in Figure 6-2.

```
AND(Input=Price|Input=Movie Name|Output=Ticket)
```

Figure 6-2 Example Query Generated by a User Agent

6.1.2 Query Generation by a Software Agent

Similar to a user agent, a software agent specifies a query with a set of criteria and a single filter operator. The difference is that, the software agent is supposed to use the Ontological Concept Service to find the concepts in the system ontology, during query criterion generation. This is necessary because for IOPE related criterion, the target values should be present in the Service Repository Cloud's unified ontology.

Since the Service Discovery Client also uses the Ontological Concept Service to retrieve a set of concepts based on the user input, the Service Discovery and

Publishing API provides methods to get a set of concepts given a word. However, the selection of the proper concept from the list of the received concepts -which might or might not contain the input word-, is left to the software agent.

6.1.3 OWL-S Ontology Concepts

Ontological concepts are used in criteria as target values, since the services are defined with conceptual spaces filled with ontological concepts and service matching is made via ontological concepts. The agents can make a selection from a set of ontological concepts returned by the Service Repository Cloud in response to a partially or entirely typed word.

There are several algorithms for returning lexically similar words based on an input word; however these algorithms [67], [68] rely heavily on SQL Like operator to find substring matches, which Google App Engine's Data Store does not support. The reason for such a restriction is the high processing cost of such queries.

Therefore for the sake of practicality and speed, a Unicode character based index is built on the words, and a set of closest words to the input word in terms of an index value is returned to the agent. For query "Abc", results "Aaa, Aab, Abd, Abf, Acd" might be returned. The returned concepts are not necessarily semantically similar to the query.

6.1.4 AND/OR Filter Operators

The criteria entered by the user, or generated by the software agent are joined with a filter operator, which can be AND or OR. Default filter operator is OR, but the selection is left to the agent. The filter operators AND and OR are used widely in many search applications, and their functionality is trivial. In our case the common functionality of the filter operator AND is quite eliminatory, and ends with very few, if not zero, results for service discovery. The main reason behind this is that, service clients and publishers rarely describe a concept with the same words, even when they actually mean the same concept.

To overcome this problem, we have introduced new meanings to these filter operators, which help the AND condition match more specific services than the OR condition and still have a large result set with services similar to the query, even if the resulting services are not entirely matching with the query criteria.

For the OR condition, any of the matching criterion promotes in a positive manner to the Service Match Score of a service, while the non-matching criteria will not affect the score in any way. For the AND condition, the matching criteria will still promote the Service Match Score; however the non-matching data points in the criteria will affect the score negatively with respect to their weights.

6.2 Query Parsing and Categorization

The queries generated by the agents include multiple criteria and a filter operator. The criteria in a query can be of different importance levels. Therefore query parsing is an important step in service discovery. A main category is assigned to the query by processing IOPE information in the query criteria. Query categorization is also an important step, because it allows the conceptual space generated for the query to be a more balanced space with respect to the service conceptual spaces.

The Service Repository Cloud first processes the given query to find the individual criteria, and then sort the criteria in the order of importance. The importance of the query criteria is ordered as shown in Figure 6-3.

Input = Output > Precondition = Effect > Service Name > Full Text (Service Descriptor)
--

Figure 6-3 Query Criteria Importance Order

The query filter operator is also extracted and stored by the Query Parsing and Categorization Engine, before the service ranking takes place.

In order to create a main category for the query, the criteria containing the Input, Output, Precondition and Effect variables as the query target are examined, and the category of the conceptual data points rooting from these are counted. For every original data point rooting from an IOPE criterion, the corresponding category gains points. At the end, the category with the most criterion points is selected to be the main category of the query, and the significance value of the category plane corresponding to the main category is increased to Query Main Category Significance Amount. Appendix A contains the exemplary value used for Query Main Category Significance Amount in our algorithm.

For the significance calculation of the remaining category planes in the conceptual space of the query, Dimension Significance Computation algorithm described in section 5.7.2 is reused.

The details of the query categorization algorithm are as given in Figure 6-4.

1. For all the category planes
 - a. For all the data points in the category plane
 - i. If the point is an original data point, Main Category Candidate's significance value is increased by point's weight
 - ii. If the point is a data point added to the plane via an ancestry relationship, Main Category Candidate's significance value is increased by half of the point's weight
2. Main Category Candidates are sorted and the highest ranking category is selected

Figure 6-4 Query Categorization Algorithm

6.3 Categorical Service Matching

Categorical service matching depends on the previously created data, by the Service Repository Cloud and it is a rather simple process. The first data necessary for the matching algorithm is created when a web service is published to the Service Repository Cloud: a permanent conceptual space is created for the given service, and stored for future use in service discovery. The second data necessary for the matching algorithm is created when the query is sent to the system: a temporary conceptual space is created for the given query.

These conceptual spaces, dimension significance values and category plane similarities are used by the service match ranking algorithm to obtain the most similar services, and return a limited set of Service Match Records to the requesting agent.

The service match ranking algorithm of our system has two modes of operation based on the filter operator specified by the query (i.e. OR or AND). In the first mode, data point misses between the query and service spaces are neutral to the service match score; whereas in the second mode, data point misses are penalized.

The details of the service match ranking algorithm are outlined in Figure 6-5.

1. Service Search Space is reduced to services whose conceptual spaces contain categorical planes present in query's conceptual space
 - a. If the number of retrieved services is higher than the Maximum Number of Services to be Ranked (see Appendix A), further refine the services by sorting them based on the maximum number of intersecting planes
2. For each service in the Service Search Space
 - a. For each category plane in the conceptual space of the service
 - i. For each category plane in the conceptual space of the query
 1. If the category of the plane originating from the service is equal to the category of the plane originating from the query
 - a. All intersecting data points between the planes are found
 - b. The weights of the intersecting data points in both of the planes are multiplied
 - c. All the multiplied weights are summed
 - d. The obtained weight value is multiplied by the significance values of the query and service category planes
 2. If the category of the plane originating from the service is different from the category of the plane originating from the query
 - a. For each data point in the query category plane
 - i. Distance between the query data point and the points in the service category plane are calculated by adding predefined concept levels
 - ii. Weights of the data points are multiplied for each data point pair

Figure 6-5 Service Match Ranking Algorithm (Continued on Next Page)

- iii. The obtained results are divided by the conceptual distances between data point pairs
 - iv. The resulting numbers are summed up for the source query data point
 - v. The resulting sum is multiplied by the Category Plane Significance Score of the source and target category planes
 - b. The importance values obtained for each source value are multiplied by significance values of each category plane, and summed up
 3. The obtained final score value in case 1 or case 2 is added to the Service Match Score
3. The services are sorted based on their Service Match Scores, and a page of Service Match Records are returned to the agent.

Figure 6-5 Service Match Ranking Algorithm

6.4 Query Forwarding and Query Responses

When certain conditions are satisfied, the Service Repository Cloud might forward the service discovery request of an agent to a Service Repository Peer. A query forward might mean one of the following three: (i) there is not a reasonable number of available services in the cloud to make an accurate service ranking, (ii) there is an adequate number of services in the cloud, however none of the services are suitable enough for the received query or (iii) there is a Service Repository Peer specialized to the main category of the query, which might contribute better results to the service discovery.

In these cases, the Service Repository Cloud returns a forward message, bundled with the Service Match Record list to the query owner agent. The agent is free to follow or discard the forward. In our implementation, Service Discovery Client

follows the forwards by default, and enhances the Service Match Record list with responses from the Service Repository Peers.

The Service Repository Cloud forwards an agent to a Service Repository Peer if and only if it has received an online status update from this peer at least in one of the previous five status update rounds. This is done to ensure that when a query is forwarded to a Service Repository Peer, the given peer is online and available. However, a Service Publisher Peer can still go offline or discontinue its Service Repository Peer during the query forward. Therefore, a synchronous request can result in the Service Discovery Client, or the software using Service Publishing and Discovery API to be locked until a connection timeout occurs. To prevent these locks, the requests to the Service Repository Peers are made in an asynchronous manner.

The service discovery seems to be finished in the Service Repository side when a query is answered with a list of Service Match Records or a query forward message, but the agent still has to pick a service and execute it. To enable an easy to use service selection and execution experience, the Service Discovery Client formats the list of Service Match Records displaying valuable information like match score, online/offline information and service detail links in the user interface.

However, the Service Match Records contain more information, to enable easier interaction with software-agents. The format of a Service Match Record is described in Figure 6-6.

Field Name	Field Description
Service Name	A string, provided by the service publisher titling the service
Service Main Category	One of the main categories supported by Service Repository Cloud
Service Match Score	The match score computed for the agent query
Last Online Date	A date time value displaying the last time when the service publisher has provided a status update
Service Key	The unique identifier of the service, especially useful for API users who can programmatically call service related functionality from Service Repository Cloud
Publisher Name	Name of the publisher, helpful for both user and software agents to distinguish a specific service (i.e. an “official” service)
Service OWL-S Descriptor	Downloadable link to the OWL-S descriptor of the service, which unites all four OWL-S descriptors of the service
Service OWL-S Concept Descriptor	Downloadable link to the OWL-S Concept descriptor of the service
Service OWL-S Grounding Descriptor	Downloadable link to the OWL-S Grounding descriptor of the service
Service OWL-S Process Model Descriptor	Downloadable link to the OWL-S Process Model descriptor of the service
Service OWL-S Service Descriptor	Downloadable link to the OWL-S Service descriptor of the service
Service WSDL Descriptor	Downloadable link to the WSDL descriptor of the service

Figure 6-6 Contents of a Service Match Record

Once a discovery agent decides to select a service, it can visit a Service Detail page, which contains both machine processable information like links to the OWL-S and WSDL descriptors of the service and human readable information like categorical graphic, user comments and ratings. A sample categorical graphic for a sample web service in our system in the “Communication” category is shown in Figure 6-7.

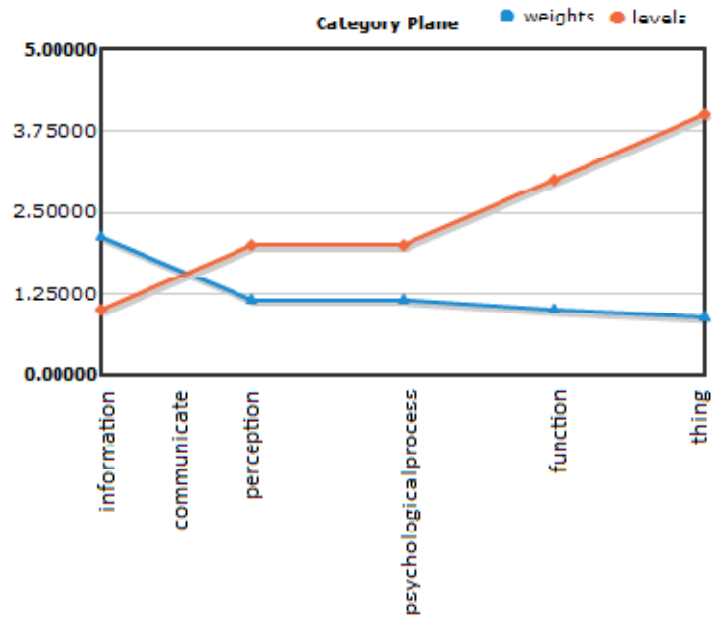


Figure 6-7 Categorical Graphic for a Sample Web Service

These descriptor links are not the actual links to the files; rather they are pointers to the Service Repository Cloud to serve the desired file with a HTTP GET request to the Service Descriptor Restlet Service.

CHAPTER 7

TESTS AND PERFORMANCE ANALYSIS

In this chapter the performance of the system is discussed. Section 7.1 outlines the details of the system's performance, while section 7.2 outlines the details of the service discovery algorithm's performance. In section 7.3, the performance of the system is reviewed.

7.1 System Performance

Our performance tests are made on the Google App Engine cloud. The resources provided by the cloud to our application are as shown in Figure 7-1.

Resource	Capacity
Application Processing Power	6.50 CPU Hours per Day
Data Store Processing Power	60 CPU Hours per Day
Data Storage Capacity	1 GB Total
Outgoing Bandwidth	1 GB per Day
Incoming Bandwidth	1 GB per Day

Figure 7-1 Google App Engine Resource List

In our system, we used SUMO's v1.75 ontology. A total of 12724 SUMO Concepts, 4600 SUMO Classes and 116652 WordNet Concepts have been migrated to our Data Store from the SUMO ontology, by spending 6 CPU hours in total. Figure 7-2 shows the entity distribution in our Data Store, after the ontology migration.

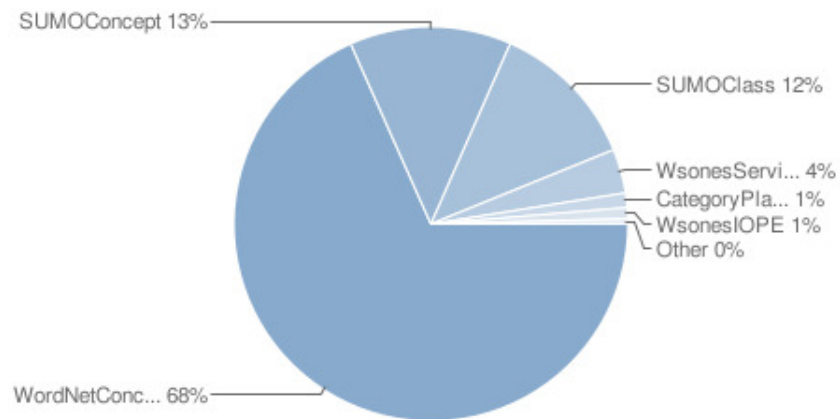


Figure 7-2 Entity Distribution in Data Store

Most of the data in our Data Store is textual information, and the metadata about the entities in the store (i.e. query indexes, search indexes). Figure 7-3 shows the usage of storage space by data type.

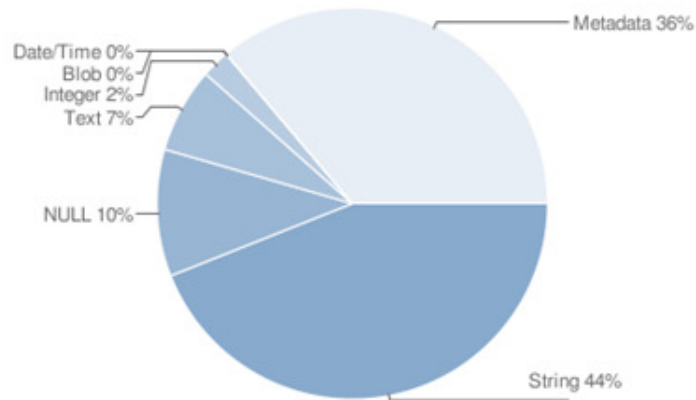


Figure 7-3 Storage Space by Property Type

7.1.1 Concept Request for Annotation

Our annotation process requests for concepts starting with a word provided by the user or the software agent. Since the request is time critical, proper indexes on SUMO Concept, SUMO Class and Word Net Concept labels are built. The results for conducted tests are shown in Figure 7-4.

Parameter	Value	Result
Request Count	1000	
Minimum Processing Time	180ms	
Maximum Processing Time	2734ms	
Average Processing Time	800ms	
Average Response Time	4000ms	

Figure 7-4 Concept Request Performance

7.1.2 Web Service Publishing

Web service publishing is one of the most important services that our system provides. The service has multiple steps, including network delays for service descriptor transfer, OWL-S parsing and semantic information extraction and Hilbert Space generation. The results for conducted tests are shown in Figure 7-5.

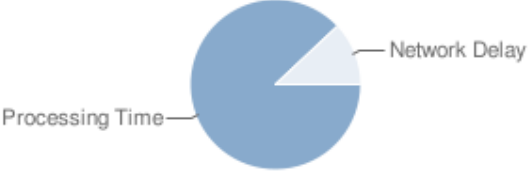
Parameter	Value	Result
Publishing Count	188	 <p>A pie chart illustrating the composition of the average response time. The chart is divided into two segments: a large blue segment representing 'Processing Time' and a smaller light blue segment representing 'Network Delay'. The 'Processing Time' segment is significantly larger, indicating it is the dominant component of the total response time.</p>
Minimum Processing Time	15000ms	
Maximum Processing Time	30000ms	
Average Processing Time	26500ms	
Average Response Time	31000ms	
Average Response Time	31000ms	

Figure 7-5 Web Service Publishing Performance

The processing time for individual web service publishing operations can also be broken down as shown in Figure 7-6.

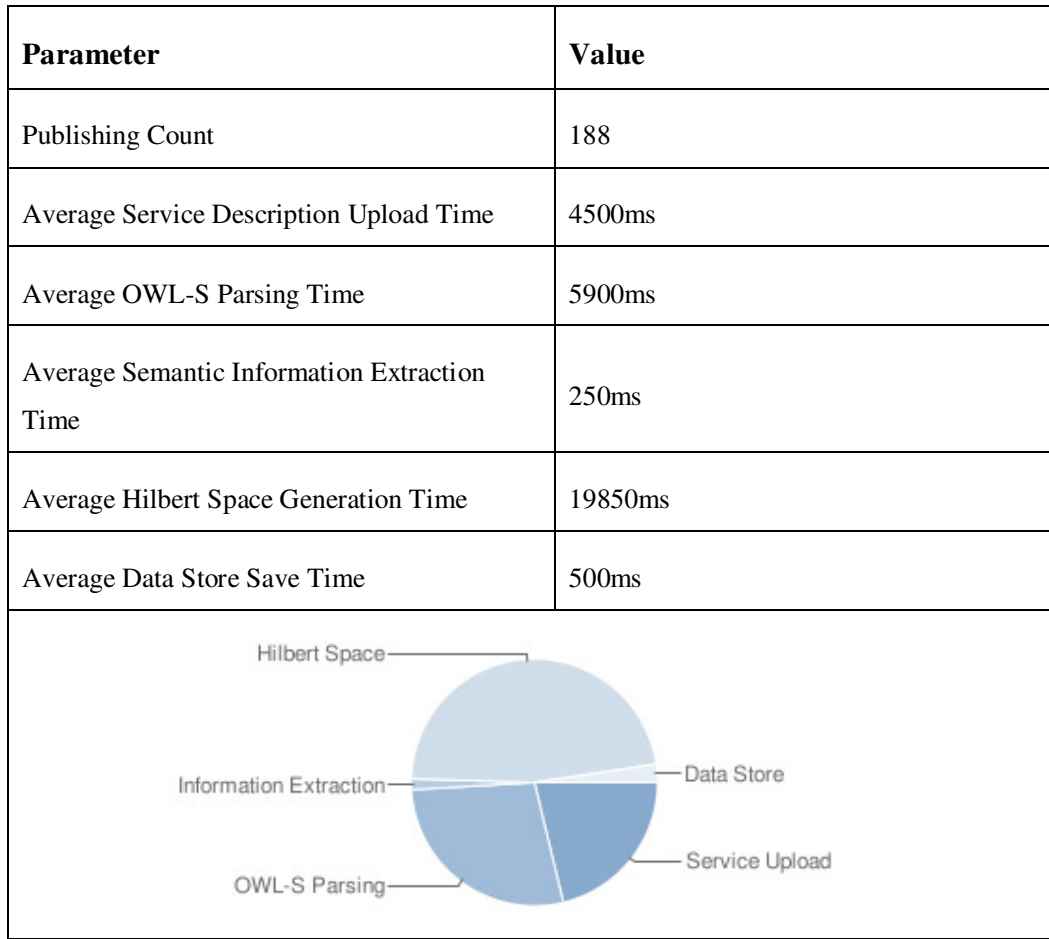


Figure 7-6 Web Service Publishing Detailed Performance

7.1.3 Web Service Discovery

Web service discovery is the most important service provided by our system. The service primarily focuses on comparing Hilbert Spaces. The comparisons are mathematical computations, and relatively very fast operations. The results for conducted tests are shown in Figure 7-7.


Parameter	Value	Result
Query Count	1200	
Minimum Processing Time	1200ms	
Maximum Processing Time	8500ms	
Average Processing Time	2200ms	
Average Response Time	3100ms	

Figure 7-7 Web Service Discovery Performance

7.2 Discovery Performance

We have uploaded 188 distinct web services to our system, obtained from the QWS data set [69]. The services were manually annotated, using concepts from the Service Repository Cloud ontology. Our system created the Hilbert Spaces for the published service as shown in Figure 7-8.

Total Web Service Count	188
Total Number of Dimensions in Hilbert Spaces	1132
Average Number of Conceptual Dimensions in a Hilbert Space	6.02
Total Number of Data Points in Hilbert Spaces	4702
Average Number of Data Points in a Hilbert Space	25.01

Figure 7-8 Hilbert Space Statistics for Published Web Services

Three types of queries were created for system tests.

- The first group includes queries targeting specific web services with known information about the service. The criteria for these queries were prepared manually, making use of the ontology concepts used to annotate the web service.
- The second group includes queries aimed at web services in specific category planes. The criteria for these queries were prepared manually, making use of the ontology concepts in the same category as the web service. Concepts specified in semantic web service descriptions were not used in the criteria. However, parent and child concepts of these concepts were used.
- The third group includes queries aimed at finding web services based on functionality, completely disregarding system ontology and annotations used for publishing web services.

The queries are created for the web service profile shown in Figure 7-9.

<p>Service Profile</p>	<pre> <process:Input> <process:parameterType rdf:datatype="&xsd:anyURI"> &wsones;#anagram </process:parameterType> </process:Input> <process:Output> <process:parameterType rdf:datatype="&xsd:anyURI"> &wsones;#randomization </process:parameterType> </process:Output> </pre>
-------------------------------	--

Figure 7-9 Sample Web Service Profile

A sample Group 1 query for the sample web service is shown in Figure 7-10.

Query	<i>AND(Input=anagram Output=randomization)</i>																												
Actual Result	<table border="1"> <thead> <tr> <th>Service Name</th> <th>Service Revision</th> <th>Service Main Category</th> <th>Service Match Score</th> </tr> </thead> <tbody> <tr> <td>Anagram</td> <td>1</td> <td>communication</td> <td>6.4426403</td> </tr> <tr> <td>News</td> <td>1</td> <td>communication</td> <td>4.0</td> </tr> <tr> <td>Credentials</td> <td>1</td> <td>computing</td> <td>2.2</td> </tr> <tr> <td>ProductArchive</td> <td>1</td> <td>engineering</td> <td>2.2</td> </tr> <tr> <td>SOAP</td> <td>1</td> <td>computing</td> <td>2.2</td> </tr> <tr> <td>JobManager</td> <td>1</td> <td>computing</td> <td>2.2</td> </tr> </tbody> </table>	Service Name	Service Revision	Service Main Category	Service Match Score	Anagram	1	communication	6.4426403	News	1	communication	4.0	Credentials	1	computing	2.2	ProductArchive	1	engineering	2.2	SOAP	1	computing	2.2	JobManager	1	computing	2.2
Service Name	Service Revision	Service Main Category	Service Match Score																										
Anagram	1	communication	6.4426403																										
News	1	communication	4.0																										
Credentials	1	computing	2.2																										
ProductArchive	1	engineering	2.2																										
SOAP	1	computing	2.2																										
JobManager	1	computing	2.2																										

Figure 7-10 Group 1 Query Sample

A sample Group 2 query for the sample web service is shown in Figure 7-11.

Query	<i>AND(Input=linguisticexpression Output=randomize)</i>																								
Actual Result	<table border="1"> <thead> <tr> <th>Service Name</th> <th>Service Revision</th> <th>Service Main Category</th> <th>Service Match Score</th> </tr> </thead> <tbody> <tr> <td>Anagram</td> <td>1</td> <td>communication</td> <td>4.89867</td> </tr> <tr> <td>News</td> <td>1</td> <td>communication</td> <td>4.0</td> </tr> <tr> <td>BibleSearch</td> <td>1</td> <td>people</td> <td>2.8666668</td> </tr> <tr> <td>ProductArchive</td> <td>1</td> <td>engineering</td> <td>2.2</td> </tr> <tr> <td>SoftwareDevelopment</td> <td>1</td> <td>computing</td> <td>2.7773504</td> </tr> </tbody> </table>	Service Name	Service Revision	Service Main Category	Service Match Score	Anagram	1	communication	4.89867	News	1	communication	4.0	BibleSearch	1	people	2.8666668	ProductArchive	1	engineering	2.2	SoftwareDevelopment	1	computing	2.7773504
Service Name	Service Revision	Service Main Category	Service Match Score																						
Anagram	1	communication	4.89867																						
News	1	communication	4.0																						
BibleSearch	1	people	2.8666668																						
ProductArchive	1	engineering	2.2																						
SoftwareDevelopment	1	computing	2.7773504																						

Figure 7-11 Group 2 Query Sample

A sample Group 3 query for the sample web service is shown in Figure 7-12.

Query	AND(<i>Input=literature</i> <i>Output=words</i>)																															
Actual Result	<table border="1"> <thead> <tr> <th>Service Name</th> <th>Service Revision</th> <th>Service Main Category</th> <th>Service Match Score</th> </tr> </thead> <tbody> <tr> <td>Anagram</td> <td>1</td> <td>communication</td> <td>4.3213205</td> </tr> <tr> <td>News</td> <td>1</td> <td>communication</td> <td>4.0</td> </tr> <tr> <td>Credentials</td> <td>1</td> <td>computing</td> <td>2.2</td> </tr> <tr> <td>ProductArchive</td> <td>1</td> <td>engineering</td> <td>2.2</td> </tr> <tr> <td>SOAP</td> <td>1</td> <td>computing</td> <td>2.2</td> </tr> <tr> <td>JobManager</td> <td>1</td> <td>computing</td> <td>2.2</td> </tr> </tbody> </table>				Service Name	Service Revision	Service Main Category	Service Match Score	Anagram	1	communication	4.3213205	News	1	communication	4.0	Credentials	1	computing	2.2	ProductArchive	1	engineering	2.2	SOAP	1	computing	2.2	JobManager	1	computing	2.2
Service Name	Service Revision	Service Main Category	Service Match Score																													
Anagram	1	communication	4.3213205																													
News	1	communication	4.0																													
Credentials	1	computing	2.2																													
ProductArchive	1	engineering	2.2																													
SOAP	1	computing	2.2																													
JobManager	1	computing	2.2																													

Figure 7-12 Group 3 Query Sample

300 of each type of queries were created, summing up to a total number of 1200 queries during the tests. The algorithm is awarded with points for each successful query response. Each 1st place obtained by a target web service is awarded with 2 points and each 1st page (2nd-5th place) obtained by a target web service is awarded with 1 points. If the target web service is not on the 1st page, this is considered as a miss and the algorithm is awarded 0 points.

The summary results for Group 1 queries can be seen in Figure 7-13.

Criteria	Result	Points
Number of 1st Places	221	442
Number of 1st Pages	63	63
Number of Misses	16	0
	Total Points	505/600
	Success Ratio	84.16%

Figure 7-13 Summary Results for 300 Group 1 Queries

The summary results for Group 2 queries can be seen in Figure 7-14.

Criteria	Result	Points
Number of 1st Places	103	206
Number of 1st Pages	152	152
Number of Misses	45	0
	Total Points	358/600
	Success Ratio	59.66%

Figure 7-14 Summary Results for 300 Group 2 Queries

The summary results for Group 3 queries can be seen in Figure 7-15.

Criteria	Result	Points
Number of 1st Places	87	174
Number of 1st Pages	119	119
Number of Misses	94	0
	Total Points	293/600
	Success Ratio	48.83%

Figure 7-15 Summary Results for 300 Group 3 Queries

The aggregated summary results for all three types of queries can be seen in Figure 7-16.

Criteria	Result	Points
Number of 1st Places	411	822
Number of 1st Pages	334	334
Number of Misses	155	0
	Total Points	1156/1800
	Success Ratio	64.22%

Figure 7-16 Aggregated Summary Results for All Types of Queries

7.3 Performance Review

Our algorithm performs relatively well when conceptual or categorical relatedness is provided in the query (i.e. Group 1 and Group 2 queries). Even though our success ratios seem low, this is mostly due to our point awarding system. For instance, in Group 1 and Group 2 queries, only 61 of 600 queries are missed, leading to an 89.83% 1st page viewing ratio.

Most of the misses recorded by our system (e.g. 60% of all the misses) are due to Group 3 queries. Group 3 queries, by definition disregard conceptual and categorical similarities, and are not quite suitable for our system, since our similarity algorithms are completely based on conceptual similarity.

However, the performance recorded throughout our tests is still below our expectations. We have detected three main reasons for this situation:

- Ontological controversy between web services in QWS data set and SUMO
- High volume of uncategorized semantic data in SUMO
- Missing categories in SUMO

There's an ontological controversy between web services in QWS data set and SUMO. While SUMO focuses in category planes described in Figure 5-5, QWS data set mainly focuses on Biology, Internet Technologies and Media. Due to the ontological controversy, for each service uploaded a very small number of conceptual dimensions and data points could be created, as described in Figure 7-8. Since our system primarily depends on data points in the generated Hilbert Spaces, this controversy degrades our performance.

There is a high volume of uncategorized semantic data in SUMO. As shown in Figure 7-2, WordNet Concepts account to 68% of the semantic data in the ontology. Even though some of the WordNet Concepts have categories due to their parent-child relationships with SUMO Classes and SUMO Concepts, some are not explicitly categorized by the SUMO team. These uncategorized WordNet Concepts are placed in a "Generic" dimension and cannot efficiently contribute to our web service discovery routine.

SUMO is missing many categories and it does not have category granularity. Many of the web services in the QWS data set belong to categories not present in the SUMO ontology, and they are represented with "Generic" dimensions. Additionally, categories like "Computing" and "People" are very large categories and they can be further categorized to sub-categories. Missing of these categories and sub-categories undermine our inter-dimensional similarity computation metrics.

CHAPTER 8

CONCLUSION AND FUTURE WORK

Web services play an important role in the world of information technology, due to the popularity of architectural approaches requiring a higher degree of automation. Web services also provide many useful features to their users, including increased interoperability, platform independence, programming language neutrality and composability. Since there are a wide range of web services available on the web, and many more are under development, it is very hard for human beings to locate the web services matching their functional and semantic requirements. Once these services are located, they can be executed, or further be composed with other web services. The described web service discovery and composition processes can be automated by using standard ontologies, semantic descriptor formats and service discovery and publishing systems using these standards.

This thesis proposes a software system which allows web service publishers to convert their WSDL web service descriptions to OWL-S web service descriptions, annotate their web service descriptions, and publish the descriptions to a semi-decentralized peer-to-peer network using a client application or a software API. The software system also allows the web service clients to query the network using a client application or a software API to find services matching a given criteria. All of

the web service publishing and web service discovery operations are performed via a single, unified ontology.

Most remarkable features of the proposed “Service Oriented Peer to Peer Web Service Discovery Mechanism with Categorization” system can be summarized as follows:

- It is highly scalable and robust.

The architecture uses a semi-decentralized peer to peer network. The central software is setup on a cloud computing architecture, and peers can also be setup on local clouds. Only explicitly requesting peers can be marked as super peers, which respond to service discovery requests. The super peers are continuously monitored by the central system for their reliability and uptime. Highly available, distributed Data Stores are used for storing data. Services are grouped categorically and conceptually during service publishing to reduce response times for service discovery queries.

- Web service similarities are computed with a novel approach, the Hilbert Space approach, based on categories and concepts.

A novel similarity computation mechanism is introduced with this system. The semantic web services are represented as a multi-dimensional Hilbert Space. The Hilbert Space for the web service is created during web service publishing and it is updated with the user feedback via service discovery queries. The pre-computed Hilbert Space descriptions are used in the service discovery process. Hilbert Spaces model both conceptual and categorical relations successfully. The approach also provides functionality to align concepts in different categories. All of the computation is performed via a single ontology, so concept coherence is attained.

- Web service publishing is done semi-automatically, using a single, unified ontology.

The system allows the service publishers to convert, annotate and publish their existing web services. Existing WSDL documents are accepted by the system, and conversion to OWL-S is done automatically. Publishers can use the provided editor to annotate their OWL-S documents. The annotation is done using a single, unified ontology, which allows system-wide concept coherence. Services are monitored continuously for changes, and automatically republished to the system when necessary. The availability of web service publishers are also tracked by the system.

- Automated, semantic web service discovery with web service ranking feature.

The system allows service users to discover web services automatically. When the unified ontology is used to generate queries, result set for the query is returned via the relevant web service. The result set contains machine processable information like match score, service name, service online/offline status and hyperlinks to service descriptors. The queries can be sent to the system by using the built-in client application or provided service discovery API. Making use of the provided API, developers can use the system transparently to create service composition algorithms.

- Web service semanticity is enhanced via web service discovery queries.

The system supports transparent user feedback, and uses the feedback to enhance the semanticity of the web services in its repository. With prolonged usage of the system, the Hilbert Spaces of the services will change as clients select a given service. With this feature, even though a service may not be annotated very well, or it may be annotated in a misleading way, the service will be positioned to its intended place via the user feedback.

- Top-down service oriented architecture.

The system is completely service oriented. All the functionality provided by the system, including user signup operations, web service publishing, web service

updating, web service discovery queries and relational concept queries are implemented as restful web services. Even though the system provides an API for programmatic access, with this feature it becomes completely language and platform neutral.

This thesis has the mentioned significant contributions to the automated web service discovery and publishing problems. However, some future work is still present. Most important future work is working with a rich, fully categorized ontology. Even though SUMO is a large ontology, it can represent only a minor fraction of the world we live in. This results in scattered concepts and missing categories in the Hilbert Spaces of the web services. When a wider ontology is available, the approach will reach to its full potential.

Another future work is to compute multiple categories for concepts. Our approach divides concepts with multiple categories to independent category planes. This way inter-dimensional concept similarity can only be computed with plane similarity measures. Instead, each concept can be represented as a single data point in the Hilbert Space, with non-zero coordinates in multiple category dimensions. This will allow us to see the web services as multi-dimensional complex solids in a Hilbert Space. With this advancement, service discovery requests can simply be seen as the intersection computation for complex multi-dimensional objects.

Another future work can be semi-automated web service annotation, during service publishing. Our system allows the publishers to browse our ontology using a widget in the annotation editor; however it does not provide suggestions to the publisher by scanning the web service descriptor at hand. Such a system can automatically suggest semantic annotations for the OWL-S service descriptor, and reduce the workload of the web service publisher.

A final future work can be allowing transparent structural and semantic composition for provided queries. With this advancement, our system can transparently compose web services in the repository to create a web service that matches the complex solid

described by the query Hilbert Space. The composition can simply be modeled as the unification of solids to create a bigger multi-dimensional object.

REFERENCES

- [1] W3C Working Group. (Last visited on 01.01.2010) Web Services Architecture. <http://www.w3.org/TR/ws-arch/>
- [2] OASIS SOA Reference Model Technical Committee. (Last visited on 02.01.2010) OASIS SOA Reference Model. <http://www.oasis-open.org/committees/soa-rm/>
- [3] T. Koskela, J. Julkunen, J. Korhonen, M. Liu, and M. Ylianttila, "Leveraging Collaboration of Peer-to-Peer and Web Services," in *Mobile Ubiquitous Computing, Systems, Services and Technologies*, 2008, pp. 496-501.
- [4] OASIS UDDI Specification Technical Committee. (Last visited on 01.01.2010) OASIS UDDI Specification. <http://www.oasis-open.org/committees/uddi-spec/>
- [5] OASIS Group. (Last visited on 01.01.2010) OASIS Group. <http://www.oasis-open.org/>
- [6] Microsoft. (Last visited on 01.01.2010) Windows Server 2003 UDDI Services. <http://www.microsoft.com/windowsserver2003/technologies/idm/uddi/default.aspx>
- [7] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. (Last visited on 01.01.2010) Web Services Description Language 1.1 Reference. <http://www.w3.org/TR/wsdl>
- [8] W3C. (Last visited on 01.01.2010) OWL-S: Semantic Markup for Web Services. <http://www.w3.org/Submission/OWL-S>

- [9] Michael Pantazoglou and Aphrodite Tsalgatidou, "A P2P Platform for Socially Intelligent Web Service Publication and Discovery," in *The Third International Multi-Conference on Computing in the Global Information Technology*, 2008, pp. 271-276.
- [10] W3C. (Last visited on 01.01.2010) Resource Description Framework. <http://www.w3.org/RDF/>
- [11] S.N. Srirama, M. Jarke, Hongyan Zhu, and W. Prinz, "Scalable Mobile Web Service Discovery in Peer to Peer Networks," in *Third International Conference on Internet and Web Applications and Services*, 2008, pp. 668-674.
- [12] Dimitrios Skoutas, Dimitris Sacharidis, Verena Kantere, and Timos Sellis, "Efficient Semantic Web Service Discovery in Centralized and P2P Environments," in *Lecture Notes in Computer Science.: Springer Berlin / Heidelberg*, 2008, pp. 583-598.
- [13] Kunal Verma et al., "METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services," *Information Technology and Management*, vol. 6, no. 1, pp. 17-39, January 2005.
- [14] DAML. (Last visited on 01.01.2010) DAML Semantic Web Services. <http://www.daml.org/services>
- [15] Adam Pease. (Last visited on 01.01.2010) Suggested Upper Merged Ontology. <http://www.ontologyportal.org/>
- [16] Changtao Qu and W. Nejdl, "Interacting the Edutella/JXTA peer-to-peer network with Web services," in *Proceedings of 2004 International Symposium on Applications and the Internet*, 2004, pp. 67-73.

- [17] W3C XML Protocol Working Group. (Last visited on 02.01.2010) SOAP Version 1.2. <http://www.w3.org/TR/soap/>
- [18] Deborah L. McGuinness and Frank van Harmelen. (Last visited on 01.01.2010) OWL Web Ontology Language. <http://www.w3.org/TR/owl-features/>
- [19] David Hilbert, John von Neumann, and Lothar Nordheim, "Über die Grundlagen der Quantenmechanik," *Mathematische Annalen*, vol. 98, no. 1, pp. 1-30, March 1928.
- [20] Mathematica. (Last visited on 01.01.2010) Wolfram MathWorld. <http://mathworld.wolfram.com/HilbertCube.html>
- [21] Ian Niles and Adam Pease, "Towards a standard upper ontology," in *Proceedings of the international conference on Formal Ontology in Information Systems*, 2001, pp. 2-9.
- [22] Standard Upper Ontology Working Group. (Last visited on 01.01.2010) Standard Upper Ontology. <http://suo.ieee.org/>
- [23] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599-616, June 2009.
- [24] Rüdiger Schollmeier, "A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications," in *Proceedings of the First International Conference on Peer-to-Peer Computing*, 2002.
- [25] Napster. (Last visited on 01.01.2010) Napster. <http://www.napster.com>

- [26] eMule. (Last visited on 02.01.2010) Official eMule Homepage.
<http://www.emule-project.net>
- [27] Freenet. (Last visited on 02.01.2010) Freenet P2P Foundation.
<http://www.p2pfoundation.net/Freenet>
- [28] Matei Ripeanu, "Peer to Peer Architecture Case Study: Gnutella Network," in *Proceedings of the First International Conference on Peer-to-Peer Computing*, 2002.
- [29] Daniel Nurmi et al., "The Eucalyptus Open-Source Cloud-Computing System," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009, pp. 124-131.
- [30] Microsoft. (Last visited on 01.01.2010) Windows Azure.
<http://www.microsoft.com/windowsazure/>
- [31] Google. (Last visited on 21.01.2010) Google App Engine.
<http://code.google.com/appengine/>
- [32] Rackspace. (Last visited on 20.01.2010) Rackspace Cloud.
<http://www.rackspacecloud.com/>
- [33] Amazon. (Last visited on 20.01.2010) Amazon Elastic Computing Cloud.
<http://aws.amazon.com/ec2/>
- [34] Roy Fielding and Richard N. Taylor, "Principled Design of the Modern Web Architecture," *ACM Transactions on Internet Technology*, vol. 2, no. 2, pp. 115-150, May 2002.
- [35] Leonard Richardson and Sam Ruby, *RESTful web services*, 1st ed.: O'Reilly, 2007.

- [36] Restlet. (Last visited on 20.01.2010) Restlet: A RESTful Web Framework for Java. <http://www.restlet.org/>
- [37] AppScale. (Last visited on 21.01.2010) AppScale. <http://code.google.com/p/appscale/>
- [38] Salesforce. (Last visited on 21.01.2010) Salesforce PaaS. <http://www.salesforce.com/paas/>
- [39] Xen. (Last visited on 20.01.2010) Xen HyperVisor. <http://www.xen.org/>
- [40] Katarzyna Keahey, Mauricio Tsugawa, Andrea Matsunaga, and Jose Fortes, "Sky Computing," *IEEE Internet Computing*, vol. 13, no. 5, pp. 43-51, September 2009.
- [41] Jing Zhong and Hong Ying, "A Semantic Web Based Peer-to-Peer Service Registry Network," in *First International Conference on Semantics, Knowledge and Grid*, 2005, pp. 122-122.
- [42] Sun Microsystems. (Last visited on 22.01.2010) Juxtapose. <https://jxta.dev.java.net/>
- [43] Anna Averbakh, Daniel Krause, and Dimitrios Skoutas, "Exploiting User Feedback to Improve Semantic Web Service Discovery," in *Lecture Notes in Computer Science.*: Springer Berlin / Heidelberg, 2009, pp. 33-48.
- [44] Sparck Jones Karen, "A statistical interpretation of term specificity and its application in retrieval," *Journal of Documentation*, vol. 60, no. 5, pp. 493 - 502, 2004.
- [45] Minghui Wu, Fanwei Zhu, Jia Lv, Tao Jiang, and Jing Ying, "Improve Semantic Web Services Discovery through Similarity Search in Metric

- Space," in *2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering*, 2009.
- [46] Thomas Cover and Peter E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21-27, January 1967.
- [47] Christian Platzer and Schahram Dustdar, "A vector space search engine for Web services," in *Third IEEE European Conference on Web Services*, November, 2005, pp. 14-16.
- [48] Marco Crasso, Alejandro Zunino, and Marcelo Campo, "Easy web service discovery: A query-by-example approach," *Science of Computer Programming*, vol. 71, no. 2, pp. 144-164, April 2008.
- [49] Huei Chuang, D. L. Lee, and K. Seamons, "Document ranking and the vector-space model," in *IEEE Software*, 1997, pp. 67-75.
- [50] E. Garcia. (Last visited on 21.01.2010) Cosine Similarity and Term Weight Tutorial. <http://www.miislita.com/information-retrieval-tutorial/cosine-similarity-tutorial.html>
- [51] Angus E. Taylor, *Introduction to functional analysis.*: Wiley, 1958.
- [52] Jinghai Rao and Xiaomeng Su, "A Survey of Automated Web Service Composition Methods," in *Lecture Notes in Computer Science.*: Springer Berlin / Heidelberg, 2005, pp. 43-54.
- [53] Google. (Last visited on 30.12.2009) Google AppSpot. <http://www.appspot.com/>
- [54] KVM Group. (Last visited on 30.12.2009) Kernel Based Virtual Machine. <http://www.linux-kvm.org/>

- [55] MySQL. (Last visited on 01.01.2010) MySQL Cluster.
<http://www.mysql.com/products/database/cluster/>
- [56] Google. (Last visited on 02.01.2010) Google App Engine Tools.
<http://code.google.com/appengine/docs/python/tools/uploadingdata.html>
- [57] TerraCotta. (Last visited on 20.01.2010) Ehcache Distributed Cache.
<http://www.ehcache.org/>
- [58] Google. (Last visited on 21.01.2010) Limitation on Indexed Properties at Google App Engine. http://groups.google.com/group/google-appengine/browse_thread/thread/d5f4dcb7d00ed4c6
- [59] Bill Katz. (Last visited on 01.01.2010) Simple Full Text Search for App Engine. <http://www.billkatz.com/2009/6/Simple-Full-Text-Search-for-App-Engine>
- [60] M. Amoretti, F. Zanichelli, and G. Conte, "Enabling Peer-To-Peer Web Service Architectures with JXTA-SOAP," in *In Proceedings of IADIS International Conference e-Society 2008*, 2008.
- [61] K. Gottschalk, S. Graham, H. Kreger, and J. Snell, "Introduction to web services architecture," *IBM Systems Journal*, vol. 41, no. 2, pp. 170-177, April 2002.
- [62] Google. (Last visited on 20.01.2010) Google Accounts API.
<http://code.google.com/apis/accounts/>
- [63] CMU Atlas Project Group. (Last visited on 01.01.2010) WSDL2OWL-S.
<http://www.daml.ri.cmu.edu/wsdl2owls/>
- [64] W3C XML Schema Working Group. (Last visited on 01.01.2010) XML

Schema 1.1 Specification. <http://www.w3.org/XML/Schema>

- [65] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," Stanford InfoLab, 1999.
- [66] Eugene Agichtein, Carlos Castillo, Debora Donato, Aristides Gionis, and Gilad Mishne, "Finding high-quality content in social media," in *Proceedings of the international conference on Web search and web data mining*, 2008, pp. 183-194.
- [67] Martin F. Porter, "An algorithm for suffix stripping," *Program: electronic library and information systems*, vol. 40, no. 3, pp. 211-218, 2006.
- [68] John Ulmschneider and James Doszkocs, "A practical stemming algorithm for online search assistance," *Online Information Review*, vol. 7, no. 4, pp. 301-318, 1983.
- [69] E. Al-Masri and Q. H. Mahmoud, "QoS-based Discovery and Ranking of Web Services," in *IEEE 16th International Conference on Computer Communications and Networks (ICCCN)*, 2007, pp. 529-534.
- [70] George A. Miller, Christiane Fellbaum, Randee Teng, and Helen Langone. (Last visited on 01.01.2010) WordNet. <http://wordnet.princeton.edu/>
- [71] Nullsoft. (Last visited on 01.01.2010) Nullsoft Scriptable Install System. <http://nsis.sourceforge.net/>
- [72] Dem Pilafian. (Last visited on 20.01.2010) Bare Bones Browser Launch for Java. <http://www.centerkey.com/java/browser/>
- [73] FusionCharts. (Last visited on 01.01.2010) FusionCharts v3. <http://www.fusioncharts.com/>

APPENDIX A ALGORITHM CONSTANTS

Maximum Original Data Point Weight	1.2
Maximum Ancestry Data Point Weight	0.9
Service Name Significance Amount	0.6
IOPE Significance Amount	0.8
Query Main Category Significance Amount	0.9
User Feedback Cap	0.2
User Feedback Weight	0.001
Maximum Number of Services to be Ranked	200