



Norwegian University of  
Science and Technology

# Service Discovery for Future Mobile Services

**Atif Bhatti**

**Imran Aslam Choudhary**

Master of Telematics - Communication Networks and  
Networked Services (2 year)

Submission date: Van Thanh Do, ITEM

Supervisor:



# Problem Description

In current mobile systems, services are limited and known by both the user and the mobile device. It is hence quite simple for a user roaming onto a location to make use of the services available at this location. In the near future, mobile devices will be capable of connecting to multiple heterogeneous mobile network systems simultaneously such as 3G, WLAN, WIMAX, UWB, etc. and have access to multiple services that are fully, partially equivalent or completely different in terms of functionality, performance, quality, security, price, etc. The challenge is then to discover and recognize these services in order to offer them to the user. It is also crucial that the service discovery and matching are accomplished in an acceptable amount of time. The first goal of this project is to study, and evaluate existing service discovery systems, protocols and procedures. Secondly, the project is aiming at designing a service discovery for future mobile services. The project consists of the following tasks:

- o Study of current service discovery systems
- o Requirements of future mobile services
- o Evaluation of current discovery systems
- o Specification of future service discovery systems
- o Design of a service discovery for future mobile services

Assignment given: 20. January 2010

Supervisor: Van Thanh Do, ITEM



# Dedication

---

We would like to dedicate this Master Thesis to our beloved country ‘‘Pakistan’’ which has given us identification and confidence to stand proud in this world.

We are also thankful to our families without whom support this must have never been possible. Thank you for all the unconditional love, guidance, and support that you have always given us. Thank you for everything. It will take another thesis to express how special we feel to have you as part of our lives.



# Abstract

---

The pervasive computing environment for heterogeneous network is on a continuous rise. The ability to interact and control network devices with different functionalities within office and home environment could be very beneficial to a lot of users. The service discovery in computers and mobile devices enabled them to interact with one another through wireless and heterogeneous wired networks. Services advertise their existence in a dynamic way and devices are designed with this capability to discover these services and its properties automatically. These devices are though based on different technologies but are still able to communicate and discover one another based on existing service discovery architectures. It is notable that a significant number of networked devices are now mobile and these mobile devices make service discovery more challenging.

In future mobile multi-domain multi-language environments, a service can be anything and introduced by anybody. Consequently, same or equivalent services may have different names and services with same name or type may be completely different. Existing service discovery systems are incapable of handling these situations.

We have implemented a service discovery system which supports semantics to service descriptions. It allows any user to act as a service provider and introduce any service at any time. The service provider can define any service as equivalent to any existing service and in any language as wanted. In addition, it is capable to find services that are not exact matches of the requested ones. More semantics are introduced through attributes like `EquivalenceClass`, `ParentType` and `Keywords`. The test conducted on this system in real time proves that the system is efficient and can be applied in real life.

# Preface

---

This report serves as a Thesis in Network and Quality of Service, in the final semester of the Master's Program in Computer Network and Network Services at Norwegian University of Science and Technology, NTNU. The assignment was given by Telenor and the project has been carried out at Department of Telematics.

This project has been a challenging task in the complex but very exciting field of Service Discovery. Working on a research topic with such a vast scope has been really challenging and involves lots of brainstorming.



# Acknowledgements

---

We would like to thank our supervisor, Nor Shahniza Kamal Bashah for her valuable input in every weekly meeting. Her guidance, enthusiasm and knowledge have contributed so much in this research work. We would like to express our gratitude to her.

Finally we would like to thank our Professor, Dr Do van Thanh, who always has been a source of inspiration during this project. We are really thankful to him for his continuous guidance and thoughtful ideas.

Trondheim, June 15, 2010

Atif Bhatti

Imran Aslam Choudhary

# Abbreviations

---

<b>3G</b>	Third Generation (Mobile Communication System)
<b>DA</b>	Directory Agent
<b>DAML-S</b>	DARPA Agent Mark-up Language for Services
<b>DHCP</b>	Dynamic Host Configuration Protocol
<b>DNS</b>	Domain Name System
<b>GENA</b>	General Event Notification Architecture
<b>GPRS</b>	General Packet Radio Service
<b>GSM</b>	Global System for Mobile
<b>HTTP</b>	Hyper Text Transport Protocol
<b>IANA</b>	Internet Assigned Numbers Authority
<b>IP</b>	Internet Protocol
<b>ISO</b>	International Standards Organization
<b>LUS</b>	Look up Service
<b>NAT</b>	Network Address Translation
<b>NIC</b>	Network Interface Card
<b>OWL</b>	Ontology Web Language
<b>OS</b>	Operating System
<b>P2P</b>	Peer to Peer
<b>PDA</b>	Personal Digital Assistant
<b>RDF</b>	Resource Description Framework
<b>RMI</b>	Remote Method Invocation
<b>SA</b>	Service Agent
<b>SLM</b>	Salutation Manager
<b>SLP</b>	Service Location Protocol
<b>SMS</b>	Short Message Service
<b>SOAP</b>	Simple Object Access Protocol
<b>SQL</b>	Structured Query Language
<b>SSDP</b>	Simple Service Discovery Protocol
<b>SSL</b>	Secure Socket Layer
<b>TCP/IP</b>	Transmission Control Protocol/Internet Protocol
<b>TM</b>	Transport Manager
<b>UA</b>	User Agent
<b>UDDI</b>	Universal Description, Discovery and Integration
<b>UDP</b>	User Datagram Protocol
<b>UIC</b>	UPnP Implementers Corporation
<b>UPnP</b>	Universal Plug and Play
<b>URI</b>	Uniform Resource Identifier
<b>URL</b>	Uniform Resource Locator
<b>UWB</b>	Ultra-Wideband
<b>VOIP</b>	Voice over Internet Protocol
<b>VPN</b>	Virtual Private Network

<b>WLAN</b>	Wireless Local Area Network
<b>WIMAX</b>	Worldwide Interoperability for Microwave Access
<b>WSDL</b>	Web Services Description Language
<b>XML</b>	Extensible Markup Language

# Definitions

---

<b>Service</b>	a mechanism enabling the end-user's access to one or more capabilities
<b>Network service</b>	service offered to the user by a network system
<b>Service availability</b>	the time when the service can be accessed
<b>Service continuity</b>	the ability for a user to maintain an ongoing service during mobility across domains, networks, and devices
<b>Service discovery</b>	the process of finding services that match the requirements of the service requestor
<b>Service advertisement</b>	the procedure to announce the service to potential users/consumers
<b>Service search</b>	the procedure to search and find the desired services
<b>Service lookup/service request</b>	the mediation of a request for a service
<b>Service matching</b>	the process of comparing the service request against the available service advertisements and determining which service best satisfies the request
<b>Equivalent service</b>	The service which has the same attributes or functionality is considered as equivalent service.
<b>Parent type</b>	The service which is being defined as a parent service due to the similarities are being inherited by the service the process of comparing the service request against the available service advertisements and determining which service best satisfies the request.

# Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>1</b>
1.1	Motivation .....	2
1.2	Problem Statement .....	4
1.3	Methodology .....	5
1.4	Organisation of Thesis .....	8
<b>2</b>	<b>Background .....</b>	<b>9</b>
2.1	Existing Service Discovery Architectures.....	10
2.1.1	Universal Plug and Play.....	10
2.1.2	JINI .....	11
2.1.3	Service Location Protocol (SLP) .....	12
2.1.4	Salutation .....	12
2.2.	Web Services.....	13
2.2.1.	Simple Object Access Protocol.....	14
2.2.2.	Web Service Description Language.....	16
2.2.3.	UDDI.....	18
<b>3</b>	<b>Requirements of Future Mobile Services .....</b>	<b>21</b>
3.1.	Requirement for Future Mobile Services .....	22
3.1.1.	A service in the future can be anything .....	22
3.1.2.	A service can be introduced by anybody at any time .....	22
3.1.3.	In a mobile environment, services must be discovered very fast: .....	24
3.1.4.	It is crucial not to misunderstand or confuse one service with another one .....	25
3.1.5.	All the services available in an area must be discovered.....	26
3.1.6.	It is also essential to verify that a service is offering what it announces .....	26
3.1.7.	It is also essential to be able to conclude that a service is trustful.....	26
3.1.8.	The user must be able to move everywhere in the world .....	26
3.1.9.	There are many services and service discovery systems it is important to ensure interoperability.....	27
3.2.	Evaluation of existing Service Discovery Architectures.....	29
3.2.1)	Summarizing the evaluation.....	30

<b>4</b>	<b>Design.....</b>	<b>32</b>
4.1.	Service Registration: .....	34
4.2.	Discovery of Service: .....	37
4.3.	Select a Service .....	39
4.4.	Find A Service:.....	41
<b>5</b>	<b>Implementation.....</b>	<b>44</b>
5.1.	Overview of the system architecture.....	45
5.1.1.	Web Service Client .....	45
5.1.2.	Development Environment .....	46
5.1.3.	Support for ontologies: .....	47
5.2.	Class Overview .....	47
5.2.1.	Base Class: .....	48
5.2.2.	DALService .....	48
5.2.3.	DALServiceType .....	48
5.2.4.	DALKeyword .....	48
5.2.5.	Discover .....	50
5.2.6.	OwlHelper.....	50
5.2.7.	Service.....	50
5.2.8.	ServiceType .....	50
5.3.	Database Overview .....	51
5.3.1.	Service.....	52
5.3.2.	ServiceType .....	52
5.3.4.	ServiceKeyword.....	53
5.3.5.	Keyword.....	53
5.4.	Main methods.....	53
5.4.1.	Register Service .....	53
5.4.2.	Lookup by Service Type:.....	55
5.4.3.	Lookup by Keyword: .....	57
5.4.4.	Get Service by Service name: .....	58
<b>6</b>	<b>Testing.....</b>	<b>60</b>
6.1	Use Scenarios .....	61

6.1.1.	Anybody can Introduce a service at anytime.....	61
6.1.2.	Similar services in different languages.....	63
6.1.3.	Same services with Different Name .....	65
6.1.4.	Different Services with Same Name.....	66
6.1.5.	Partially Equivalent Service.....	66
6.2.	Scalability.....	69
6.2.1.	Service Repository with 50 services.....	70
6.2.2.	Service Repository with 100 services.....	74
6.2.3.	Service Repository with 250 services.....	78
6.2.4.	Service Repository with 500 services.....	82
6.2.5.	Service Repository with 1000 services.....	86
6.3	System Overall Behavior .....	90
<b>7</b>	<b>Conclusion .....</b>	<b>92</b>
7.1	Major Contribution of this thesis .....	93
7.2	Summary of thesis.....	94
7.3	Future Work .....	94
<b>Appendix A</b>	<b>.....</b>	<b>96</b>
A.1	Class Description.....	96
A.2	Database table overview.....	103
<b>Appendix B. Publication</b>	<b>.....</b>	<b>106</b>
<b>Bibliography</b>	<b>.....</b>	<b>114</b>

# Table of Figures

---

Figure 1-1 Existing Network Scenarios .....	2
Figure 1-2 Current Scenarios .....	3
Figure 1-3 Future Scenario .....	4
Figure 1-4 Common view of design science research process.....	5
Figure 2-1 Web Service Architecture .....	14
Figure 2-2 UDDI Service Cloud.....	19
Figure 3-1 Ambiguity in words.....	22
Figure 3-2 Relations between services .....	23
Figure 3-3 Language semantics .....	24
Figure 3-4 Fast discovery of service during handoff .....	25
Figure 3-5 Service interoperability .....	27
Figure 3-6 Service discovery interoperability.....	28
Figure 4-1 case diagram of Future service discovery system .....	33
Figure 4-2 Service type description template .....	35
Figure 4-3 Sequence diagram for “Registration of Service” .....	36
Figure 4-4 Collaboration Diagram for Registration of Service .....	37
Figure 4-5 Sequence Diagram for “Discovering services” .....	38
Figure 4-6 Collaboration Diagram for “Discovering Services” .....	39
Figure 4-7 Sequence Diagram for “Service Request” .....	40
Figure 4-8 Collaboration Diagram for Service Type.....	41
Figure 4-9 Sequence Diagram for “Finding a Service” .....	42
Figure 4-10 Collaboration Diagram for “Finding a Service” .....	43
Figure 5-1 Future service discovery system architecture .....	45
Figure 5-2 Class Diagram .....	49
Figure 5-3 Entity Relationship Diagram.....	51
Figure 5-4 Main Loop followed for Registration of Services.....	55
Figure 5-5 Main loops for Service Type lookup.....	56
Figure 5-6 Loop followed for Keyword lookup .....	58
Figure 5-7 Loop for Service Name Lookup.....	59
Figure 6-1 User interface to register a new Service Type .....	61
Figure 6-2 User interface to register a new Service SubType .....	62
Figure 6-3 User Registration a new service with newly registered Service Type .....	62
Figure 6-4 User successfully registering the service received.....	63
Figure 6-5 Service in different languages equivalent to Taxi.....	64
Figure 6-6 Service in different languages equivalent to Taxch .....	64
Figure 6-7 Same services with different name .....	65



Figure 6-8 Different service with same name.....	66
Figure 6-9 Parent type and Subtype relation for telephony .....	67
Figure 6-11 Retrieval of service by service name .....	68
Figure 6-10 Retrieval of services in telephony .....	67

# Table of Tables

---

Table 3-1 Summarizing evaluation of existing architectures ..... 30

# Table of charts

---

- Chart 6-1 Retrieving all services when number of services registered are 50..... 70
- Chart 6-2 Retrieving services by service name when number of services registered are 50..... 71
- Chart 6-3 Retrieving services with service type lookup where services registered are 50..... 72
- Chart 6-4 Retrieving services by keywords when number of services registered are 50 ..... 73
- Chart 6-5 Retrieving all services when number of services registered are 100..... 74
- Chart 6-6 Retrieving services by service name when number of services registered are 100..... 75
- Chart 6-7 Retrieving services by service Type when number of services registered are 100 ..... 76
- Chart 6-8 Retrieving services by Keyword lookup when number of services registered are 100 77
- Chart 6-9 Retrieving all services when number of services registered are 250..... 78
- Chart 6-10 Retrieving services with service name when number of services registered are 250. 79
- Chart 6-11 Retrieving services by Service Type with no. of services 250 ..... 80
- Chart 6-12 Retrieving services by keyword when number of services registered are 250..... 81
- Chart 6-13 Retrieving all services when number of services registered are 500..... 82
- Chart 6-14 Retrieving services by name when number of services registered are 500 ..... 83
- Chart 6-15 Retrieving services with service type having 500 services ..... 84
- Chart 6-16 Retrieving services by keyword when number of services registered are 500..... 85
- Chart 6-17 Retrieving all services when number of services registered are 1000..... 86
- Chart 6-18 Retrieving services by name when number of services registered are 1000 ..... 87
- Chart 6-19 Retrieving services with service type having 1000 services ..... 88
- Chart 6-20 Retrieving services by keywords when services registered are 1000..... 89
- Chart 6-21 System overall behaviour .....90

# Chapter 1

## **1 Introduction**

*“If you go out 10 years, computers are not going to be these rectangular objects we carry around. They are going to be extremely tiny. They are going to be everywhere. There is going to be pervasive computing. It is going to be embedded in the environment, in our clothing. It is going to be self-organizing.”*

**Ray Kurzweil, 2007**

In this modern era, computer devices and network services are playing an important role to accomplish our daily tasks. Ranging from classical services as printers, scanners, fax machines to others common services such as radio, television, air condition, and DVD player, etc. can all be expressed as “ubiquitous computing”. This term presented by Mark Weiser in (1), means that computers are present everywhere in our daily environment.

Users should be able to choose and make use of the services that are available to them. Ideally, they would like to obtain access to the right services immediately, without requiring them to reconfigure their device. This function should not be noticeable by the user. They should be able to interact and manage all the devices and services whenever required without any difficulty.

In future, a number of different services will be available but discovering the appropriate service is a major challenge. For example if a user at a train station is searching for a service “Train Booking”, but he ended up discovering a service with name “book” which is a service for lending a book. This is a simple example that highlights the problem but such a discovery mistake could be very devastating and disastrous in case of emergency.

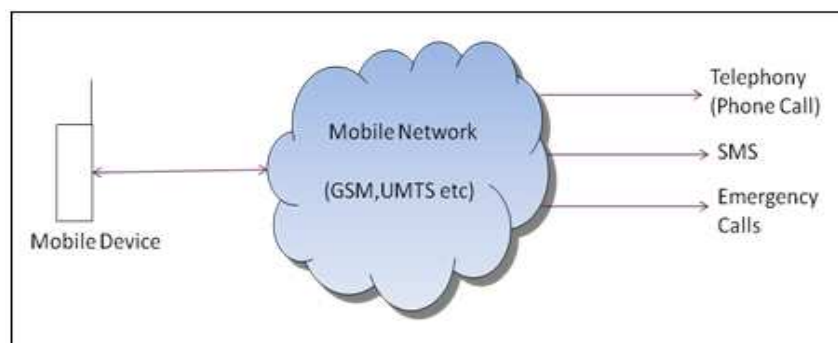
Service discovery also plays an essential role in ad hoc communications (2). The mobile phone, Pocket PC and laptop could form an ad hoc network. In such network without administrative control, the device must be self organizing. For example, the laptop may offer a translation

service to mobile phone and the phone may offer internet access service via its General Packet Radio Service (GPRS) Interface. Since this network is dynamic, there is a need of dynamic and automatic service discovery functionality. Moreover the voice and data services are expensive on the conventional communication systems such as 3G, GSM etc. Voice over IP (VoIP) is getting popular for its affordable service as compared to conventional telephony service. It is becoming popular for the Service Provider to provide access to their users from different access technologies in order to promote their services. As an example T-Mobile, a worldwide telecom operator has announced that it will introduce Unlicensed Mobile Access (UMA) in 2006. It will provide “GSM over WiFi”, thus filling coverage gaps and possibly allowing for lower traffic charges (3) . This will have certain challenges toward service continuation. The multiple connections will enable access to a wider variety of services that need to be discovered, properly understood and used.

Unfortunately, the existing service discovery systems are limited to their own domains and specific devices and not able to discover these diverse services that were introduced in an unorganized way. Therefore, a more efficient service discovery solution capable of interoperating-cover different network technologies and platforms is needed.

## 1.1 Motivation

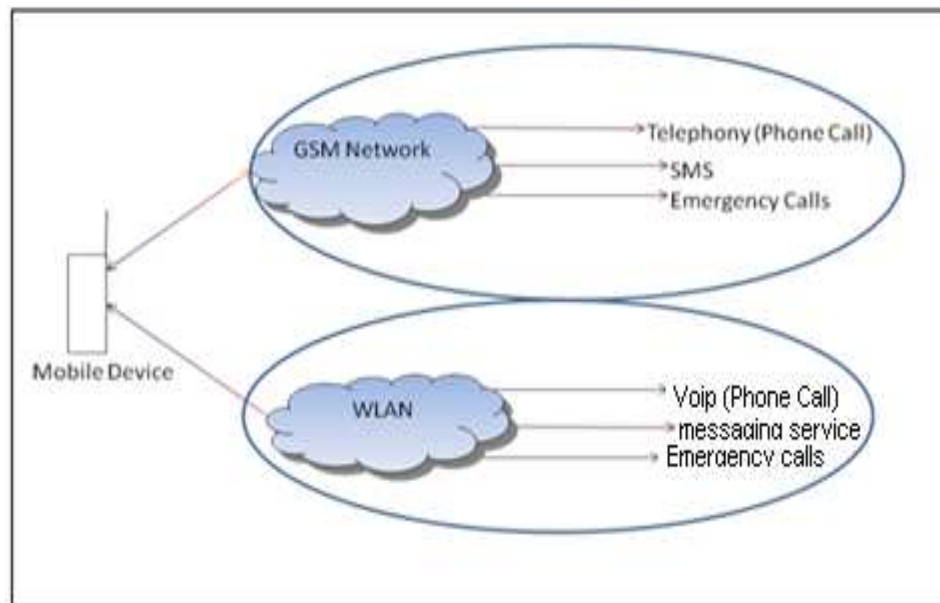
The networks of today are going through an evolution phase with the main focus shifting toward making the life easier for the end user. The technology enhancements are therefore done keeping in view of the above requirement. The main focus of this document is for service discovery for future mobile services.



*Figure 1-1 Existing Network Scenarios*

Traditional mobile networks as illustrated in Figure 1-1 have limited number of specified services which can be listed down as Telephony, SMS service, emergency calls, etc. But if we closely look at the current mobile devices that are available today we will realize that they are capable of connecting to more than one network (4). This will allow the mobile device to access different network services, such as a PDA can connect to a GSM network and meanwhile connect to a WLAN as well by using the built in Network Interface Card (NIC) (5).

Suppose a device currently connected to a GSM network is using a telephony service, during the call, the mobile device detects a WLAN network. Optionally the device can choose to connect with WLAN to use cheaper Voice over IP (VOIP) service that has been defined equivalent at the time of registration to telephony service of GSM (it is worth noting that only functionality and not quality is considered by the user in this case). The challenge is how the device can discover any equivalence between services? This could be more misleading if the service name is not appropriate or having ambiguity. The scenario is specified in Figure 1-2 where the device is not able to detect any equivalence service until or unless they have the same name or same type.

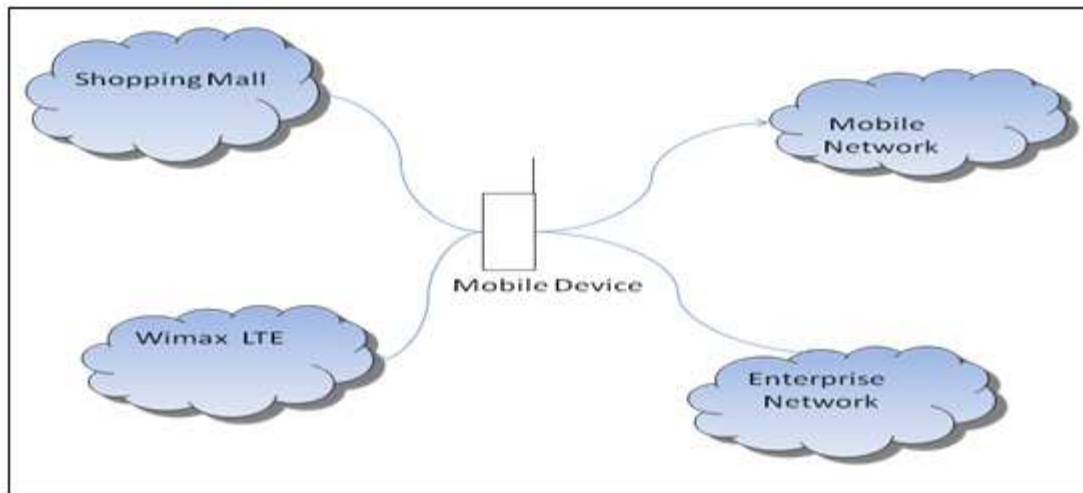


*Figure 1-2 Current Scenarios*

In current mobile systems, the number of services are limited and known by both the user and the mobile device. It is hence quite simple for a user roaming onto a location to make use of the services available at this location. In the near future as illustrated in figure 1-3, mobile devices

will be capable of connecting to multiple heterogeneous mobile network systems simultaneously such as 3G, WLAN, WIMAX, UWB, etc. and have access to multiple services that are fully or partially equivalent, completely different in terms of functionality, performance, quality, security, price, etc. The challenge is then to discover and recognize these services in order to offer them to the user. It is also crucial that the service discovery and matching are accomplished in acceptable amount of time.

The first goal of this thesis is to design and implement the newly presented service discovery system and its procedure. Secondly, it is aiming at testing and evaluating the proposed service discovery architecture for future mobile services.



*Figure 1-3 Future Scenario*

## **1.2 Problem Statement**

Addressing the described situation the main problem statements of this work are as follows:

- How can the services be discovered rapidly when the mobile user is roaming into an area?
- What are the requirements for the future service discovery?
- How can the future service discovery method allow the introduction of new services by anybody at any time?
- How can the service discovery be performed semantically?
- How can we discover same service with different names efficiently?

- How can different services with same name be discovered without confusion?
- How can services be defined as partially equivalent?
- How can services be introduced in multiple languages?

### 1.3 Methodology

The research methodology used for this work is based on the design science research process (6). The process is not a research method on its own but a formalized combination of existing methods.

Figure 1-4 illustrates the common view of the design science research process.

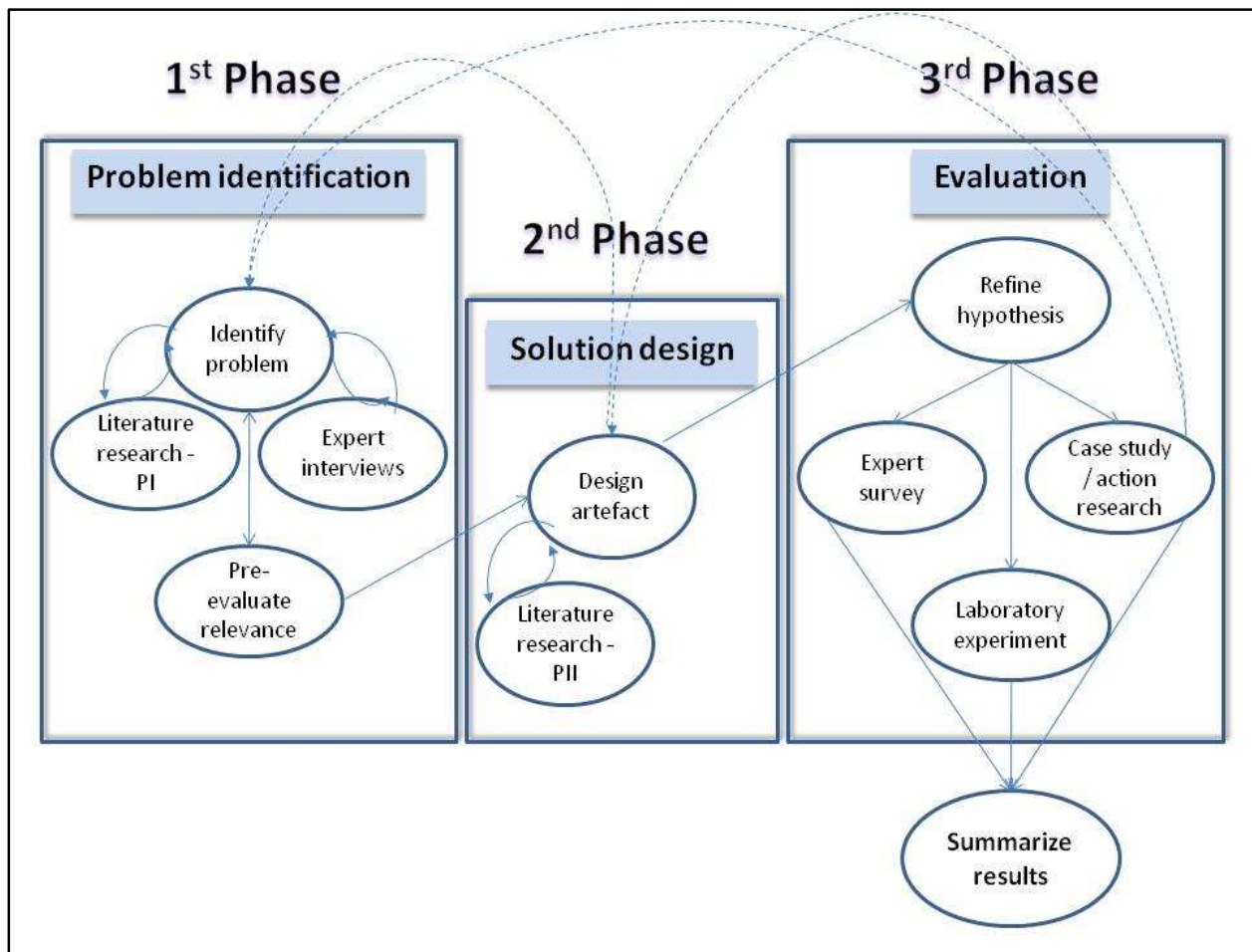


Figure 1-4 Common view of design science research process



According to this process, the research methodology is divided into three phases which are:

First Phase: Problem identification

This phase is more related to Analysis stage in a classic software design life cycle. It specifies research questions and verifies its practical relevance. From there, the research question is defined. There are four steps involved which are:

- i. Identify problem: This is to assure its relevance and understanding. Literature research or expert interviews can be used in order to identify the problem.
- ii. Literature research – Part I: It is needed to review the state-of-the-art concerning the identified problem or to analyse possible obstacles and difficulties for its solution.
- iii. Expert interviews: It is used to identify relevant and addressed problems.
- iv. Pre-evaluation relevance: Based on the problem identified from the literature research or expert interviews the pre-evaluation relevance is defined. The step involves creating a general research hypothesis in the form of a utility theory, postulating a link between the solution space and the problem space.

This phase offers a solid and important foundation for the further research process.

Second Phase: Solution design

The second phase consists of two steps which are:

- i. Design artefact: During this stage, the problem can be restated. The existing solutions and state-of-the-art have to be taken into account.
- ii. Literature research – Part II: This is a more depth study on the existing solutions and their state-of-the-art. It is important to keep track of ongoing current activities to be able to react on changes in research findings.

Third Phase: Evaluation

The steps involved in this stage are:

- i. Refine hypothesis: Should be refine by “smaller” hypothesis with a more constricted but more precise scope.
- ii. Case study/action research: It shows applicability in practice.

- iii. Expert survey: By showing general interest.
- iv. Laboratory experiments or simulations: Which is used to compare different approaches.

Steps (ii), (iii) and (iv) are the alternatives of conducting the evaluation based on types of output or expected outcome of the research process.

At the end of the research process, results are summarized and published. As the nature of action research, the iteration process (for e.g. back to “design artefact” or “identify problem”) are relevant.

The different stages in each phases have not been followed strictly through this work, since several different problem areas have been investigated, and several different artefacts have been developed. The steps involved in this research are:

*i. Identify problem and Literature research – Part I: First Phase*

In this phase first the existing service discovery systems are studied and evaluated. Based on the limitations identified, some research questions of the research are defined.

*ii. Design artefact and Literature research – Part II: Second Phase*

In the second phase, the requirements of future mobile services are proposed which is based on the findings from the initial stage (Problem identification phase). An in depth study on the existing service discovery systems and their state-of-the-art as well as current research activities involved are also conducted in order to be able to react on changes in research findings. The outputs of this phase are conceptual models of future service discovery system and descriptions of requirements for future mobile services.

*iii. Case study and Laboratory experiments – Third Phase*

In this phase, the future service discovery system is implemented based on the conceptual model of the design artefact (Solution design phase). Some case studies are given to show the applicability in practice and the laboratory experiments are conducted for testing and evaluation purpose.

## 1.4 Organisation of Thesis

This report is organized as follows:

**Chapter 1** begins with an introduction of service discovery technology and its limitation in modern heterogeneous networks. It also briefly introduced the reasons for proposing this research.

**Chapter 2** explains existing well known service discovery protocols briefly and web services particularly in detail.

**Chapter 3** presents requirement for the future service discovery protocol and evaluation of current service discovery architectures based on requirements proposed.

**Chapter 4** proposes the design of future mobile service discovery architecture by specifying use cases for different scenarios that are derived in order to fulfil the requirements.

**Chapter 5** provides all the details that have been covered in order to implement the proposed Architecture.

**Chapter 6** presents Testing of the implemented solution which is later on followed by the evaluation.

**Chapter 7** concludes this report.

# Chapter 2

## 2 Background

*“You see, wire telegraph is a kind of a very, very long cat. You pull his tail in New York and his head is meowing in Los Angeles. Do you understand this? And radio operates exactly the same way: you send signals here, they receive them there. The only difference is that there is no cat”*

**Albert Einstein, when asked to describe radio.**

As the title of the project suggests, the contribution of this research is to specify service discovery for future mobile services. This chapter will present existing well known service discovery protocols. In order to have a basic understanding of existing well known service discovery architectures we will explain each protocol in detail. Web services will be discussed in detail in second section of this chapter as it will be further used for implementation in this project whereas the architectures that will be discussed briefly includes

- 1) Universal Plug and Play
- 2) Jini
- 3) Service Location Protocol
- 4) Salutation

## **2.1 Existing Service Discovery Architectures**

This section will present existing well known service discovery protocols. In order to have a basic understanding of existing well known service discovery architectures we will explain each protocol briefly. As the future implementation of the proposed architectures will be relevant to web services architecture therefore it will be discussed in complete details. The protocols that will be discussed briefly are as follows

### **2.1.1 Universal Plug and Play**

Universal Plug and Play (UPnP) are a set of networking protocols which are designed to support zero-configuration (7), "invisible" networking, and automatic discovery for a breadth of device categories from a wide range of vendors. The UPnP architecture allows peer to peer networking of computers, networking in home appliances, and wireless devices. It is a distributed, open architecture protocol based on established standards such as TCP/IP, UDP, HTTP, XML, and SOAP.

When a new UPnP device is connected within a network for the first time it will search for DHCP server by using its Dynamic Host Configuration Protocol (DHCP) (8) client which is already embedded in it. If the DHCP server is available the device will use the IP address that was assigned to it. If there is no DHCP Server available the device will use Auto IP to choose an address from a set of reserved private addresses at the Network. The device can then move easily between managed and unmanaged networks. After the addressing phase the discovery phase will take place and that will be handled by Simple Service Discovery Protocol (SSDP) (9).

Universal Plug and Play (UPnP) is a widely accepted standard for automatically detecting devices and services in a Local Area Network (LAN). However UPnP does not provide any mechanisms for authentication. The protocol stack being used in UPnP includes IP as its lowest layer which is a very big constraint as it is limited to HTTP over UDP over IP. UPnP does not support the naming of devices as in DNS server as it always allocate IP address. IP multicast does not scale very well on big networks.

### 2.1.2 JINI

Jini pronounced “Genie”, is introduced by Sun systems in 1999. The basic idea behind the invention of this technology has been to provide the flexibility in the network (10). Different resources can be shared and used across the network as they are available locally. JINI architecture consists of major three components as JINI Client, a Service Locator, and JINI services (11). Discovery in JINI is very simple and consists of the following six steps:

- **Discover:** The device is plugged in and discovery occurs when the Service Provider looks for a Lookup Service (LUS) to register with Lookup by multicast. The Client uses unicast if know the lookup service otherwise it uses multicast.
- **Join:** Once the LUS is found, it returns a service register object to the service, which is used to register the service in the lookup.
- **Discover:** When a Client wants to use a specific service it searches for the service by either unicasting discovery if the LUS location is known or by multicasting discovery.
- **Lookup:** When Client reaches to LUS it gets the service register object, which Client uses to lookup particular service by LUS catalog and searches based on the type, name or description of service.
- **Receive:** LUS will return java proxy which contains the specification on how to connect directly to the service. If the service object consists of two programs, one proxy on the client and another controlling program on hardware device, the communication between them might use Remote Method Invocation (RMI).
- **Use:** Now the Network Client interacts directly with the network service via service proxy.

JINI is efficient as it is easier to add or remove services, relocation of services, fast discovery, and the services are available immediately and found automatically. It requires Java which needs 48 KB of memory. For future network there is no issue of memory. Jini has so many advantages but it is not scalable for future mobile where numbers of nodes are too high. Jini application can be written in any language but it has to be wrapped with Java which requires JVM to be present.

### **2.1.3 Service Location Protocol (SLP)**

Service Location Protocol was introduced by IETF in 1997 and was later updated with SLP version 2 (12). SLP provides a scalable framework for providing hosts with access to information about the existence, location and configuration of networked services (12). An SLP agent is a software entity that processes SLP protocol messages and acts in three different roles which can be listed down as User Agent (UA), Service Agent (SA) and Directory Agent (DA). The SLP User Agent is responsible for looking out for the location of services; Service Agent advertises the location of service whereas the Directory Agent can be considered as a caching entity. It is an optional entity that is used to provide scalability and also acts as a centralized repository for service location information.

SLP eliminates need of prior information to access and use a service. The user supplies the required type of service and its attributes describing the service whereas SLP retrieves the service for the user. SLP provide the dynamic configuration for services in Local Area Network where dynamic changes occur rapidly. The devices use SLP to announce services on Local Area Network. A service must have a URL for locating the service. In addition it can have a number of different key-values pair called attributes.

SLP is a simple protocol for advertisement of services in Intranet. The entities and operation of SLP is really simple. It has a really secure architecture. However it is not scalable over the Internet. There is no mechanism to deal with for replay attacks. If Directory Agent fails due to any reason the whole network and service are no longer available to communicate with each other.

### **2.1.4 Salutation**

Salutation is a service discovery architecture that has been developed by Salutation Consortium (13). The goal of Salutation is to solve the service discovery problem and making it possible with wide range of appliances and equipment within an environment that has widespread connectivity and mobility. As the devices are of various kind with different functionalities, these devices are required to be processor, operating system and communication protocol independent. The Salutation architecture enables it applications, services and devices to describe and advertise their capabilities to other applications, services and devices. Application, services and devices

can also search other applications, services or devices for a particular capability, and to request and establish interoperable sessions with them (14).

Major Components of Salutation can be listed down as: Salutation Manager (SLM) and Transport Manager (TM). The core of the architecture is the Salutation Manager. It contains a Registry where it keeps the record of the services available. A Client can register or unregister itself from the nearest Salutation Manager available. The Salutation Manager discovers the other Salutation Managers by matching types and attributes as specified by local Salutation Manager. The Salutation Managers communicate amongst one another by using Salutation Manager Protocol. As all the services are registered with their local Salutation Manager so in order to know about them, a unique feature capability exchange is needed. After discovering the required services the Client can then request the Salutation Manager to keep the track of availability of service by checking periodically. The Transport Manager is responsible for providing reliable communication channels, regardless of what the underlying network transports are.

Salutation provides us with certain advantages. Salutation is independent on the network technology and may run over multiple infrastructures, such as over TCP/IP and IrDA (15). It is not limited to HTTP over UDP over IP. There are no specified programming languages to be followed for Salutation unlike Jini which has Java as its pre requisite.

## **2.2. Web Services**

Web services introduce a new trend of reusing application modules; they are self-contained and self describing application components which can be used as a part of other applications (16) . An XML Web service can be used internally by a single application or exposed externally over the Internet for use by any number of applications accessible through a standard interface, an XML Web service allows heterogeneous systems to work together as a single web of computation.

A web service is platform independent as it uses the XML language and HTTP that can be used with any device. The HTTP protocol is mostly common in the current devices for example (Palmtop, Pocket PC, 3G cell phones). The feature that makes web services so special is the fact



that HTTP is using port 80, which is open, at most firewalls (17). Web services allow heterogeneous systems to work together as a single web of computations.

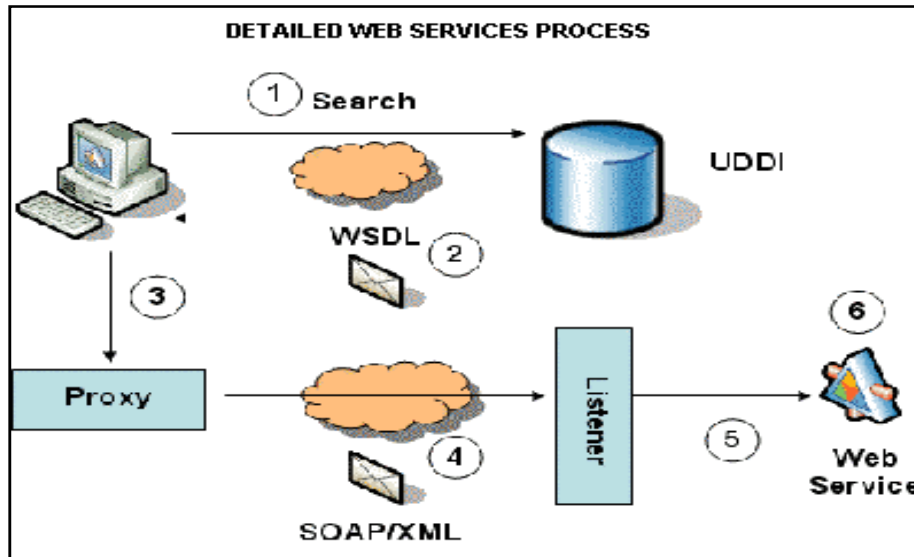


Figure 2-1 Web Service Architecture

### The Web service elements

As shown in Figure 2-1, the web service concept is based on the three elements. We will discuss in details each of the elements. The details are discussed as under

#### **2.2.1. Simple Object Access Protocol**

Simple Object Access Protocol (SOAP) is basically a communication protocol that is being used to access a Web service. SOAP basically allows a program that is running in one kind of operating system (such as Windows XP) to communicate with same or another kind of operating system (such as Linux) by using the HTTP and XML as the mechanisms to exchange information in a decentralized distributed environment (18). SOAP also specifies how the called program can return a response. SOAP as other protocols have certain advantages and disadvantages which are discuss as under

#### Advantages:

- SOAP is simple, extensible and platform independent.
- SOAP is language independent.

- Unlike previous remote execution technologies SOAP over HTTP provide extensibility and flexibility making it as platform independent and firewall-friendly.
- SOAP allows use of different transport protocols. HTTP is the standard transport protocol used in the stack, but different protocols for example RSS, SMTP etc are also usable.

#### Disadvantages:

Soap has various disadvantages which its competitor highlights and are discussed below as under

- SOAP can be slower than the competing middleware technologies because of the XML architecture.
- The interacting parties role are fixed as only one party (the client) can call the service of the others when HTTP is used as the transport protocol and WS addressing or ESB are not in use.

#### SOAP Elements:

A SOAP message is an ordinary XML document and consists of certain elements which are defined as follows.

- **Envelope:** It is a mandatory part of a SOAP message. It specifies the start and end point of the message.
- **Header:** It is an optional part of a SOAP message. It contains any additional attributes of the message which can be used in processing of the message.
- **Body:** It is the mandatory part of a SOAP message. It contains the XML data regarding the message which is being sent.
- **Fault:** It is an optional part of the message. It contains information about the errors that might occur while processing the message.

A SOAP message containing a SOAP header block and a SOAP body is written down to illustrate the elements (19)

```
<env:Envelopexmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Header>
<n:alertcontrolxmlns:n="http://example.org/alertcontrol">
<n:priority>1</n:priority>
```

```
<n:expires>2001-06-22T14:00:00-05:00</n:expires>
</n:alertcontrol>
</env:Header>
<env:Body>
<m:alert xmlns:m="http://example.org/alert">
<m:msg>Pick up Mary at school at 2pm</m:msg>
</m:alert>
</env:Body>
</env:Envelope>
```

### Conclusion:

SOAP is a lightweight protocol that can replace more complicated, distributed object technologies for many applications. However, SOAP's use of Web servers to tunnel through firewalls may limit its usefulness because it potentially opens corporations to external access. SOAP may increasingly find itself in the sights of security personnel as it will become more ubiquitous.

### **2.2.2. Web Service Description Language**

WSDL stands for Web Services Description Language and is a standard format for describing a web service. It is an XML based protocol that is used for exchanging information in a distributed environment. It is the language that UDDI uses. WSDL was developed jointly by Microsoft and IBM. WSDL is pronounced as 'wiz-dull' and spelled out as 'W-S-D-L'. W3C defines the standard as "an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate." (20).

WSDL is often used in combination with SOAP and XML schema to provide web services over the Internet. A client program connecting to a web service can read the WSDL to determine what functions are available on the server. Any special data types used are embedded in the WSDL file in the form of XML Schema (21). The client can then use SOAP to actually call one of the functions listed in the WSDL.

Elements of WSDL:

Web Services can be broken down into three specific elements by WSDL which can be reused once defined or can be combined. A WSDL document can have various elements, but they are contained within these three main elements. These three major elements of WSDL can be defined separately as

- Types
- Operations
- Binding

The main structure of a WSDL document looks like this (22)

```
<definitions>
```

```
<types>
```

```
definition of types.....
```

```
</types>
```

```
<message>
```

```
definition of a message....
```

```
</message>
```

```
<portType>
```

```
<operation>
```

```
definition of a operation.....
```

```
</operation>
```

```
</portType>
```

```
<binding>
```

```
definition of a binding....
```

```
</binding>
```

```
<service>
```

```
definition of a service....
```

```
</service>
```

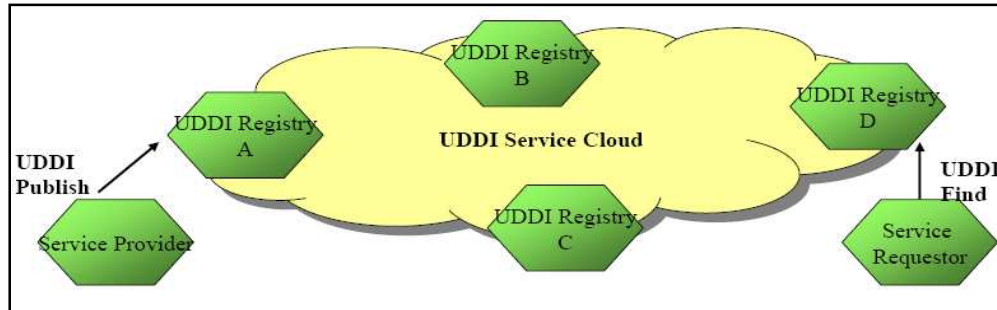
```
</definitions>
```

Sub Elements of WSDL (23):

- **Definition:** It is always the root element of a WSDL document. It specifies the name of the web service and declares multiple namespaces used in remaining of the document. It is also the container for the remaining of the service elements specifies in the document.
- **Data types:** The data types are declared for later use in the messages.
- **Message:** A message is an abstract definition of the data presented in the entire document or as an argument that can be later returned as a result of message invocation.
- **Operation:** Operation is an abstract definition for a message, such as naming a method, or business process, that will accept and process the message
- **Port type:** It is a set of operations that is mapped to one or more end points, defining the collection of operations for a binding.
- **Binding:** Protocol and data formats for the operations and messages which are specifically defined for a particular port type.
- **Port:** The target address for the service communication, which is a combination of network address and the binding.
- **Service:** Services map the binding to the port and include any extensibility definitions.

**2.2.3. UDDI**

The universal description, discovery and integration (UDDI) defines a method to publish and discover information about web services (24). To contact a business for ordering something, it is required to find information about that business: street address, telephone number, website, or web service address. It can obtain the information directly from a business representative, perhaps in the form of business card, handwritten note, or e-mail. It can also look up a business name in a telephone directory and obtain the address and telephone number. Similarly, the information necessary for a program running on a computer to talk to a program running on another computer over the web must be published. Although UDDI is like a white pages or yellow pages for web services, it also enables developers to interact with UDDI at both design time and runtime. In short, UDDI resources can be considered part of the web services architecture and infrastructure (25). The “web service” provides a business specific functionality through internet connection, for the purpose of providing a way for another company or software program to use the service.



*Figure 2-2 UDDI Service Cloud*

Figure 2-2 shows a UDDI service cloud, which consists of several collaborating UDDI registries. Web services are playing an important role in a distributed business environment. For example, company “x” providing a service for the payment through the internet, any business client can use the service of “x” for the secure transaction.

In the beginning, it seems to be very simple to manage the process of web service discovery. But the reality is different, because there are a number of different organizations each providing different services, this leads difficulty in the discovery of a service.

The UDDI is implemented in a common XML format to avoid the interoperability issues; because many companies started to define ways to allow their internal applications to interact with the business systems at other companies using the emerging web infrastructure. Each company invents a unique approach based on experience designers, available technologies and project budgets. The XML solve the integration problem between different companies and provide a single registry for all the services. (26)

The UDDI has two main parts: registration and discovery. The registration part means that businesses can post information to UDDI that other businesses can search for and discover, which is the other part. Businesses and individuals interact with UDDI by using SOAP API’s or one of the user interfaces provided by the operators or other web services vendors. UDDI operators post WSDL descriptions of their web services for registration and discovery. UDDI provides separate WSDL file for registration and discovery services, using its own XML document format.

- A UDDI registration consists further into three components:
  - White pages - contain address, contact details, and known identifiers for Web services providers.
  - Yellow pages - have industrial categorization of Web services based on standard taxonomies.
  - Green pages - contain technical information about services.

**Conclusion**

Web service is a great concept which allows the construction of applications using components distributed across heterogeneous networks and domains. However, the Web service discovery based on UDDI is falling because of inconsistencies between the information stored by the UDDI registries and the Web services really available on the Web. The readers can refer to (27) for discussing shortcomings of UDDI and the properties of an ideal. Discovery of a Web service is critical for privacy issue which is a security concern for the use of Web services (28).

# Chapter 3

## **3 Requirements of Future Mobile Services**

*“Why can’t computers in real life work like they do on Star Trek? (. . . ) They don’t have to do all that careful handwork involving cables and IP addresses and logins and passwords on Star Trek—it all just works. Is it just special effects, or are we missing something?”*

**Paul Vixie**

In future mobile environment, mobile devices will be able to perform handover between access networks to improve connectivity. To ensure service continuity, services in the new network must be discovered rapidly. However, due to the plurality and diversity of the services introduced by different players at different times and places it is quite challenging to identify services rapidly and new service discovery system is required. As the previous section contain an overview of different service discovery methods, this part is focusing on the requirements of future services, which are used to deduce the future service discovery requirement. Both requirements will be defined and explained thoroughly for clear understanding.

The chapter is divided in two sections as follows

3.1) Requirement for future mobile services

3.2) Evaluation of existing service discovery architectures based on requirement for future mobile services



### 3.1. Requirement for Future Mobile Services

#### 3.1.1. A service in the future can be anything

⇒ *Future service discovery must be capable of handling different services with same names without confusion.*

The future service can be anything and can be published by anyone, this leads to a number of challenges. The consequence is the situation where the same name or word can have several meanings and denotes different services. Ambiguity and confusion are hence introduced which can result to different interpretation (29). For example if a user search for “Book” this may lead to an ambiguity between the Literature book and ticket booking. Figure 3-1 shows the problem where user searching for the “Book” might get erroneous result. Therefore, the future service discovery must be capable of handling different services with same name without confusion.

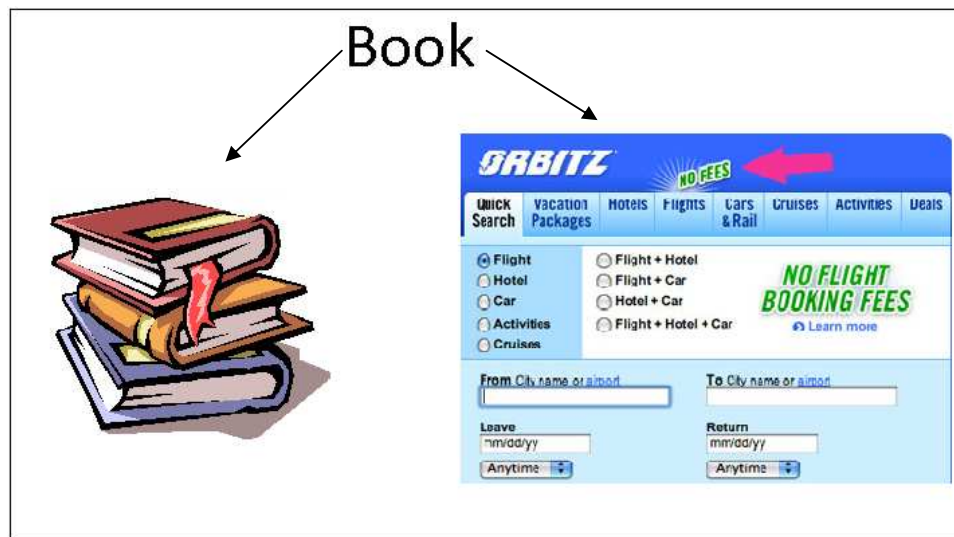


Figure 3-1 Ambiguity in words

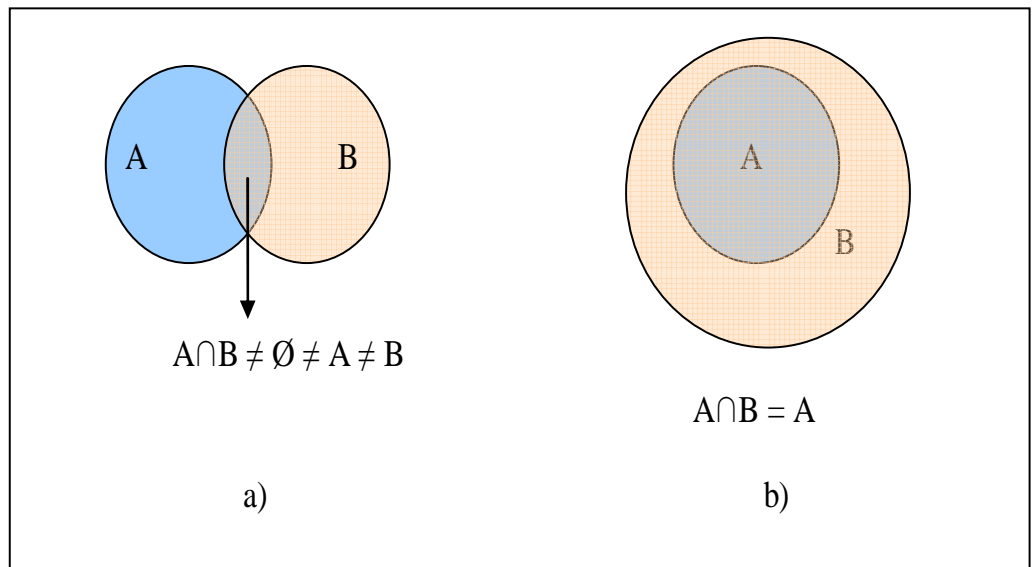
#### 3.1.2. A service can be introduced by anybody at any time

*Future service discovery,*

- ⇒ *must be capable of having services with multiple names in multiple languages.*
- ⇒ *must allow the introduction of any service anytime by anybody*

The future service discovery must allow introducing any service any time by anybody, the result of this flexibility may raise the following consequences.

- Not all services can be standardized as in current service discovery systems where a service is well specified and has a uniquely defined name.
- Another consequence is that the same service can be given different names by different service providers
- Since there is no regulation about a service definition a service may be close to another one but not 100% similar. There are two cases as follows:
  - A service A contains similar elements with service B but have also different elements as shown in Figure 3-2 a). The intersection of A and B is not empty and different from both A and B.



*Figure 3-2 Relations between services*

- A service A is a subset of service B since all the elements of A are also elements of B.
- The service may have several names in different languages but all of them refer to the same thing. The goal is to work globally without any language ambiguity. One of the users from China should be able to use a service that is provided in English and the same should be the case for an Englishman who want to use a Chinese or a Japanese service without having language as a hurdle. The future service therefore should lay special emphasize on the language compatibility.
- A service name may have several names in different languages as shown in Figure 3-3.

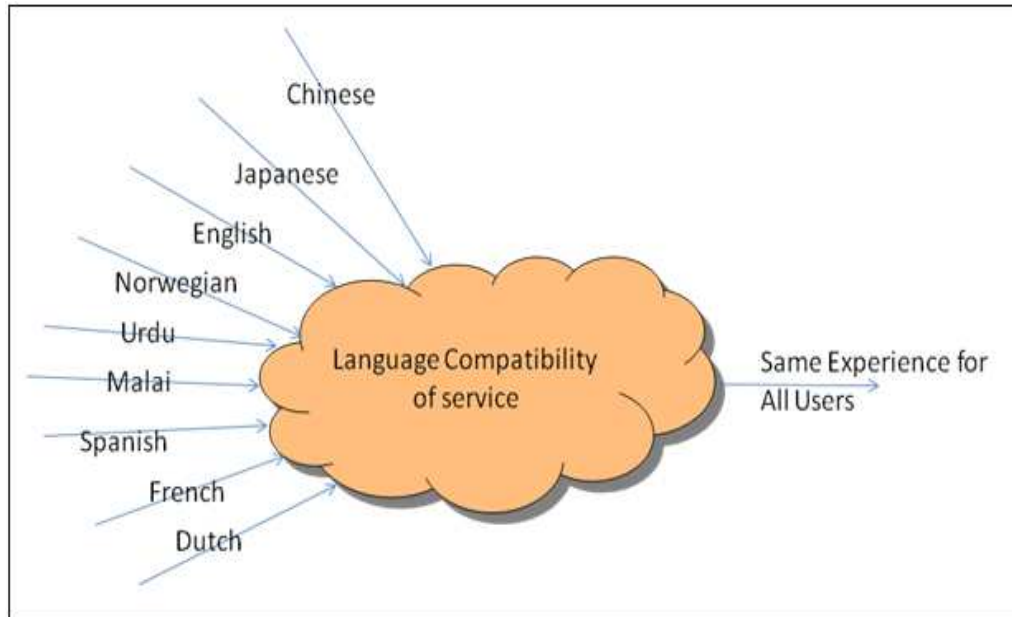


Figure 3-3 Language semantics

### 3.1.3. **In a mobile environment, services must be discovered very fast:**

⇒ Requirement for future service discovery, it must be very efficient.

The future service discovery must be very efficient. Efficient is a relative word for future service discovery. Efficient means to be fast in two ways.

- In future the user equipment can perform handoff between networks while using any service such as telephony, online shopping, banking etc. The goal of future service discovery is to be fast enough to discover the service before the handover (30). This can be more illustrated from the following example. A user "A" using internet banking service from (GPRS) GSM network, the device of user A is capable to connect with WLAN as well. When user A enter in his office its device found a network with higher bandwidth and connect with WLAN while using the internet banking the device discover the service on WLAN and handover to WLAN while continuing the internet banking service as shown in Figure 3-4.

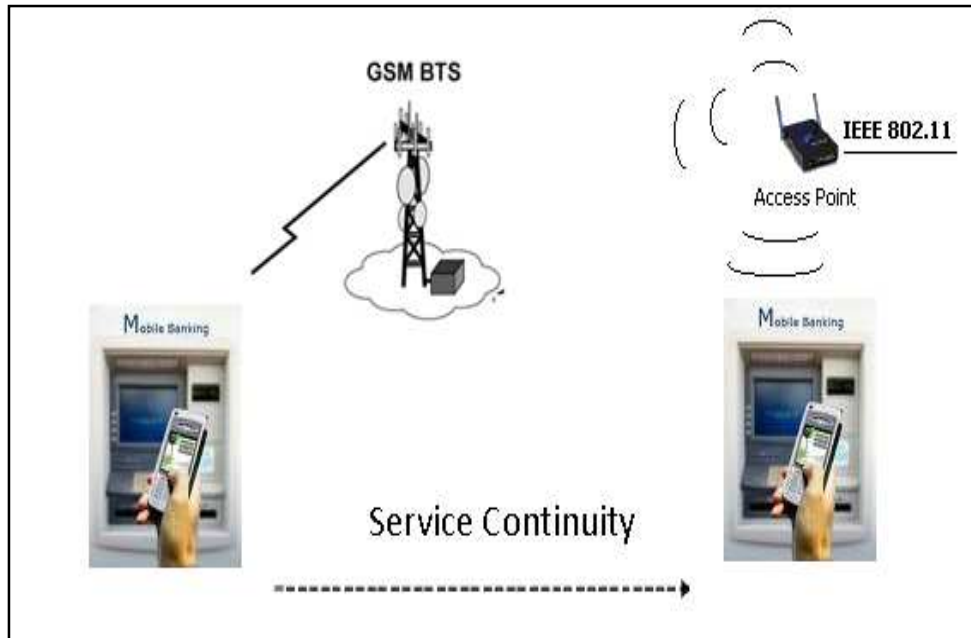


Figure 3-4 Fast discovery of service during handoff

### 3.1.4. It is crucial not to misunderstand or confuse one service with another one

⇒ *The service discovery must be error-free*

It is crucial not to misunderstand about a service or to confuse one service with another one. The service discovery must be simple and unique in order to error free. The service discovery is based on the name of the service, whenever a user uses a service it checks the available services and has an idea about the service from its name. Sometimes the service function can be misunderstood because of the short service names. The structure for the service has to be simple and unique. The end user should be able to easily use the service without any ambiguity that can create the confusion for him regarding the service type (31). The service names should be long enough to define a service properly. For example “London\_Victoria\_train\_schedule” instead of “Train\_schedule” only the first make the sense to understand what a service will provide. So if the user does not provide complete information such as “Train\_schedule” which is not sufficient to locate a specific service, the service discovery should request to add more detail in order to discover the service without any error.

**3.1.5. All the services available in an area must be discovered**

⇒ *The service discovery must be sound*

The future service discovery must be sound in order to discover all the available services. When the user find a service the service discovery should return a valid answer instead of terminating the service discovery with a message “Not Found”. If the service discovery does not find any appropriate it must request the user to add more key words in order to discover the service. To achieve this, the future service discovery must be sound.

**3.1.6. It is also essential to verify that a service is offering what it announces**

⇒ *It must be possible to extend the service discovery with verification functions.*

Before using any service it is very important to have a little knowledge about the service or description which describes the service. As in web service UDDI contains the description. In UPnP, after the discovery of a service the step 2 description describe the service. Same with Jini Lookup Service (LUS) have knowledge of the service but this description cannot directly see by the user they are more at the protocol level and user is unable to see the description. Therefore the future service discovery must verify the service to ensure what service is providing.

**3.1.7. It is also essential to be able to conclude that a service is trustful**

⇒ *It must be possible to extend the service discovery with security functions to validate a service.*

It is also essential to make sure that a service is trustful. A service has to registered with a central database, server or repository. This entity is responsible to ensure that a service is trustful. So whenever a user will be accessing the service, he will trust a service because of the entity verifying it. It will be based on mutual trust.

**3.1.8. The user must be able to move everywhere in the world**

⇒ *The future service discovery must be ubiquitously and location aware.*

For each user there is usually a set of services that he/she is using frequently such as for example weather, taxi, hotel, bus, cinema, etc. However, quite often these services are changing according to the location of the user. For example, when the user is in Paris he/she will initiate the service taxi and in this context he/she means “taxi in Paris” and not “taxi in Trondheim”. When the user arrives to Paris and switch on

his/her mobile phone and the service discovery will try to find the services weather, taxi, hotel, buss, cinema in Paris. If they exist all the information will be queried and all the necessary preparations will be made. When the user wants to request for a taxi then he/she will get the Paris taxi.

**3.1.9. There are many services and service discovery systems it is important to ensure interoperability**

⇒ *Service Discovery must be capable of discovering current existing services.*

The service discovery for future network would have a number of different features such as fast, secure, reliable and with many other specialities, but the future service discovery should be capable to discover the existing services already deployed and available for use. This is a challenge for the future service discovery mechanism to be interoperable with respect to the service discovery for both the future services as well as the existing services. For example, if an existing service was launched to use for finding a location in a city, then this service should be discovered by the future service discovery. As shown in Figure 3-5 the future service discovery can discover both the existing as well as the future services.

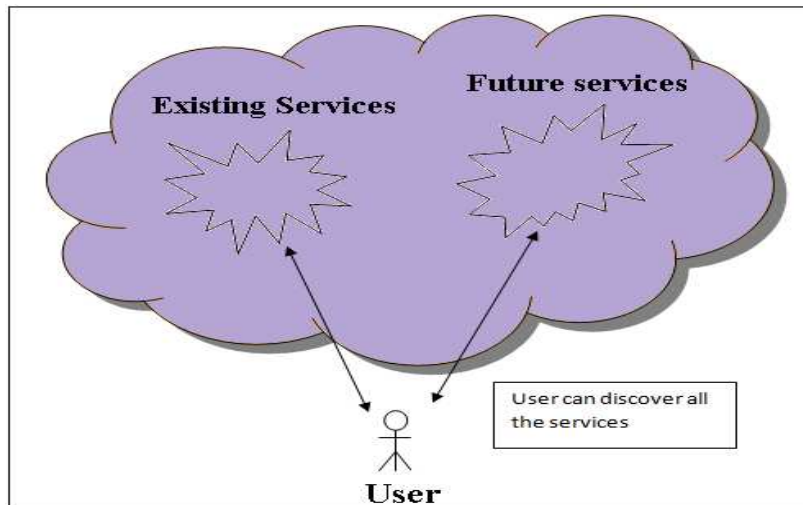
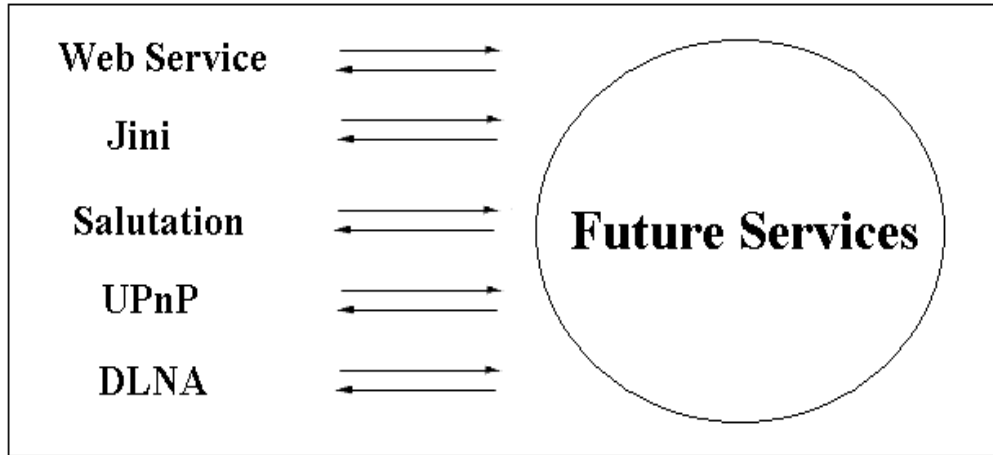


Figure 3-5 Service interoperability

⇒ *The service discovery must be capable of operating with existing service discovery systems.*

The future service should be interoperable with the current services. One of the main concerns for the future services is that the users/programmers have to design the future services from scratch but the future services must be able to communicate with the current service discovery systems in order to interact with the current services. This includes all the current service mechanisms such as Web services, Salutation, Jini, and UPnP to communicate and interoperable with future service discovery.



*Figure 3-6 Service discovery interoperability*

### 3.2. Evaluation of existing Service Discovery Architectures

This part will focus on the evaluation of current well known service discovery architectures. We will now evaluate current existing service discovery architectures using the requirements previously identified in part 1 of this chapter. Table 3-1 evaluates the existing service discovery systems by sequence mentioned below

1. Universal Plug and Play
2. Jini
3. Service Location Protocol
4. Salutation
5. Web services



**3.2.1) Summarizing the evaluation***Table 3-1 Summarizing evaluation of existing architectures*

	UpnP	Jini	SLP	Salutation	Web services
Capable of handling different services with same names without confusion	No	Partially fulfilling if 128 bit service id is remembered	No	3.2.1 No	No it is handled manually
Capable of services with multiple names in multiple languages	No	Yes by attributes of a service	Partially supporting	No	No
Allow the introduction of any service anytime by anybody	No.	Partially With registering with LUS	Partially Supporting for local domain but not globally	No Salutation manager responsible for registration	No UDDI is authority for publish a service
Efficient	Partially fulfilled With cache its fast but when primary proxy fail it's not efficient	Partially If unicast address is known	Partially Fast when directory agent is not used	NO	No
Error-free	Yes	Yes By Leasing	Yes	Yes	Yes UDDI contains the description
Sound	Partially fulfilled	No	No	Partially With salutation manager	Partially

Possible to extend service discovery with verification functions	Partial fulfilled	No	Yes	No	No
Possible to extend the service discovery with security functions to validate a service	No Internal mechanism	No	Yes But vulnerable to replay attack	Yes Identification and Password scheme	Partially SSL only provide 1-1 Not for multiple entities
Capable of functioning ubiquitously	No	No	Partially if operating with DA and Scopes	No	No
Capable of operating with existing service discovery system	Partially With JINI	Yes With UPnP, Web Service and SLP	Partially With JINI	No	Partially With JINI

# Chapter 4

## 4 Design

*“To write good software you must simultaneously keep two opposing ideas in your head. You have to be able to think how hard can it be? With one half of your brain while thinking it will never work with the other”*

**Paul Graham**

There are few important steps involved in service discovery which are to be kept in mind when proposing a new model. The steps are as follows:

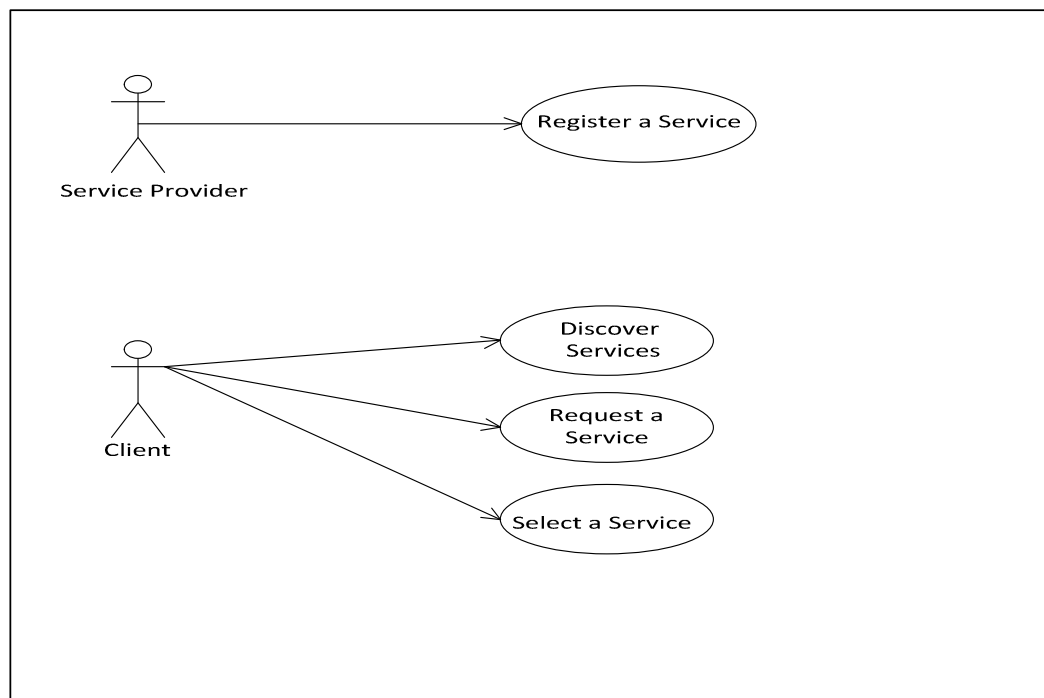
- Service registration:
  - What type of information should be supplied to the Service Broker to make it sufficient enough to do registration?
- Service discovery:
  - On the way how the discovery is organized. It is a whole system consisting of many servers, databases and functionality to discover the services.
  - There are few challenges such as:
    - The service name – since the service can be from anything and it can be introduced by anybody at any time, the open and flexible future service discovery might result to different types of service names been introduced and it is not standardized. This will result as a challenge for the service broker on how the service matching will be performed.
- Service matching:
  - Where would the service matching functionality held; should it be in the Client side or at the Network side?
    - If the service matching is done in the Client side, it is needed to consider the Client's terminal ability as it will require more powerful processor
    - If the service matching is done at the Network side, it could take longer time

From discussion above two major features are identified:

- Service Discovery – where the Client sends a message to the Service Broker asking what are the services available
  - If it is a powerful Client, it can do the matching by itself
- Service Lookup/Service Request – where the Client asking for the specific service
  - In this feature, both the service discovery and service matching are done by the Service Broker and it could take longer time.

There are number of systems which use syntactic matching for locating the contents such as “Gnutella” (32) and direct locating based on Distributed Hash Table Chord (33), they are widely used for file sharing, in the case of services it could be used for discovery of services. The chord has been proved to be efficient. But none of the methods support semantic lookup. The proposed system will be a combination of syntactic and semantic design. To have a more clear picture of the system design and considering the challenges discussed above we will use the unified modelling language (UML) (34) use case diagrams.

Four cases are specifically identified to represent the high level requirements of the future mobile services. A Service provider can register a service whereas a client can discover a list of available services, request a specific service and then select a service. The cases are illustrated in Figure 4-1 and are discusses in detail below



*Figure 4-1 case diagram of Future service discovery system*

#### **4.1. Service Registration:**

In traditional old systems the services that are present in the system are having a standardized name which is specifically assigned to it e.g telephony with a dedicated service number which uniquely identifies it. Hence a direct syntax matching is done but as we know that in future the mobile will be capable of connecting to heterogeneous network and therefore different services will exist which will be entirely new to the client . So in order to be discovered the services will have to advertise it. The Service Provider will have to provide certain details while introducing a new service which will make it easier for the user to identify the desired services.

##### **Service name:**

Service name is the first item of information that has to be provided by the Service provider while introducing a service. As specified in the requirements previously there is no restriction as anyone can introduce any service at any time, there is no specific naming convention to be followed which has been the case previously for existing service discovery architectures. Moreover the service name can be introduced in multiple languages which cannot be done earlier in other existing service discovery architectures.

##### **Service type:**

Service type is one of the necessary information required while registering a service. It is important in order to identify a service functionality e.g a service name of Micheal Jackson will have a service type as Music. In existing service discovery system the service type are already standardized and no new service type can be introduced, hence negating our requirements 1 and 2 which state that a service can be introduced by anyone . Moreover our system type allows anyone to introduce service type and in any language. There can be different cases while introducing a new service type which are discussed as follows

- **Brand-new service type:** without relation with any existing service type: A service type description in XML (eXtensibleMarkup Language) containing the service type Name, Keywords, ParentType, StateVariables, etc. as shown in Figure 4-2 below. In addition an URI (Uniform Resource Identifier) has to be assigned to this service type description. This service type description will be used later in the service matching. Since the service type is a brand-new type that was not derived by any existing service type ParentType and ParentType has to be set to nil. The service type does not have

any alias since there exists no equivalent service type. The `EquivalentClass` is then left empty.

- **Equivalent service type:** The service type has a different name and may be another implementation but is equivalent to an existing service type. The service provider has to add the name of the existing service type and also all the known service type in different languages.
- **Subtype of an existing service type:** The service type has all the functions and features of an existing service type but has also additional ones. The parent type is hence indicated.

#### Service type description template

A service type description has the following items:

- *Name:* The name can be in any language and less than 64 characters.
- *Keywords:* Some words that can be used in the first round discovery
- *ParentType:* The name of service type that the current type is derived from.
- *ParentTypeURI:* The URI of the parent type
- *EquivalenceClass:* All alias in any language are given here
- *StateVariables.* The *state variables* determine the states of the services. They are left empty in the service type description
- *Actions:* Actions are the methods that can be called by clients or other services
  - Each action has a *name* and a set of *parameters*
    - Each parameter has a *type*, *allowable values* (for enumerated types), and *direction* (in or out)
- *Events:* Enable clients to subscribe to the occurrence of a particular event

Figure 4-2 Service type description template

After defining the require fields we will take a look at the execution of the events by drawing a sequence diagram for registration of service as specified in Figure 4-3. The steps are as follows

- 1) Service provider will request the service broker to register a service by providing necessary information i-e Service name, Service Type, keywords etc.
- 2) The service broker in order to register a service has to make sure that the service type already exist or is it a brand new service type so it will pass the service type information to service matcher.
- 3) The service matcher will check whether the service type already exists or do we need to add a new service type. If the service type already exists it will just pass the information to

service broker after the successful lookup. If it's a new service type, it will be added in our system and then the information will be passed over to the service broker respectively.

4) Once the service type exists, the service broker will send the necessary information to the service info base (repository) in order to register the service. The service name will be used to check if any service with the same name and same parent type cannot be added twice whereas a service with same name and completely different functionality can be added with different parent type.

5) The service info base will update the service broker that the service has been successfully registered.

6) The service broker will update the service provider that the service has been successfully added and is now ready to be used.

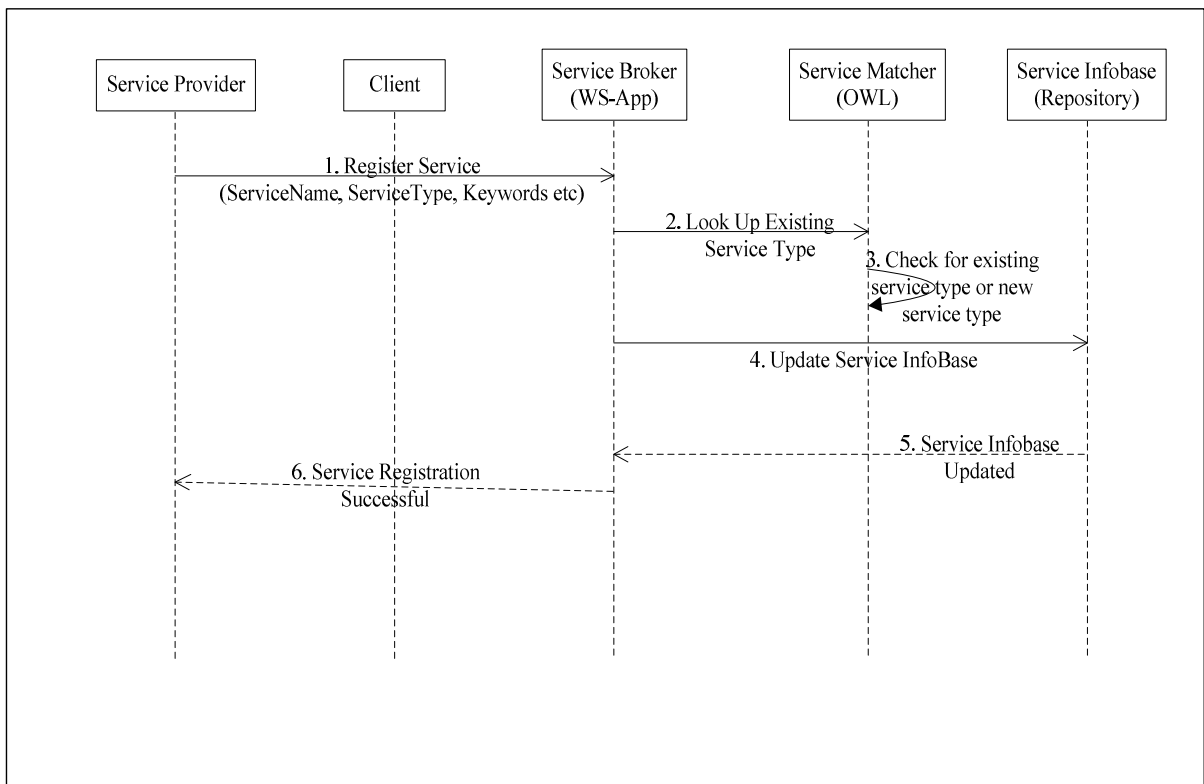


Figure 4-3 Sequence diagram for "Registration of Service"

The collaboration diagram for registration of a service can be found in figure 4-4 illustrated below

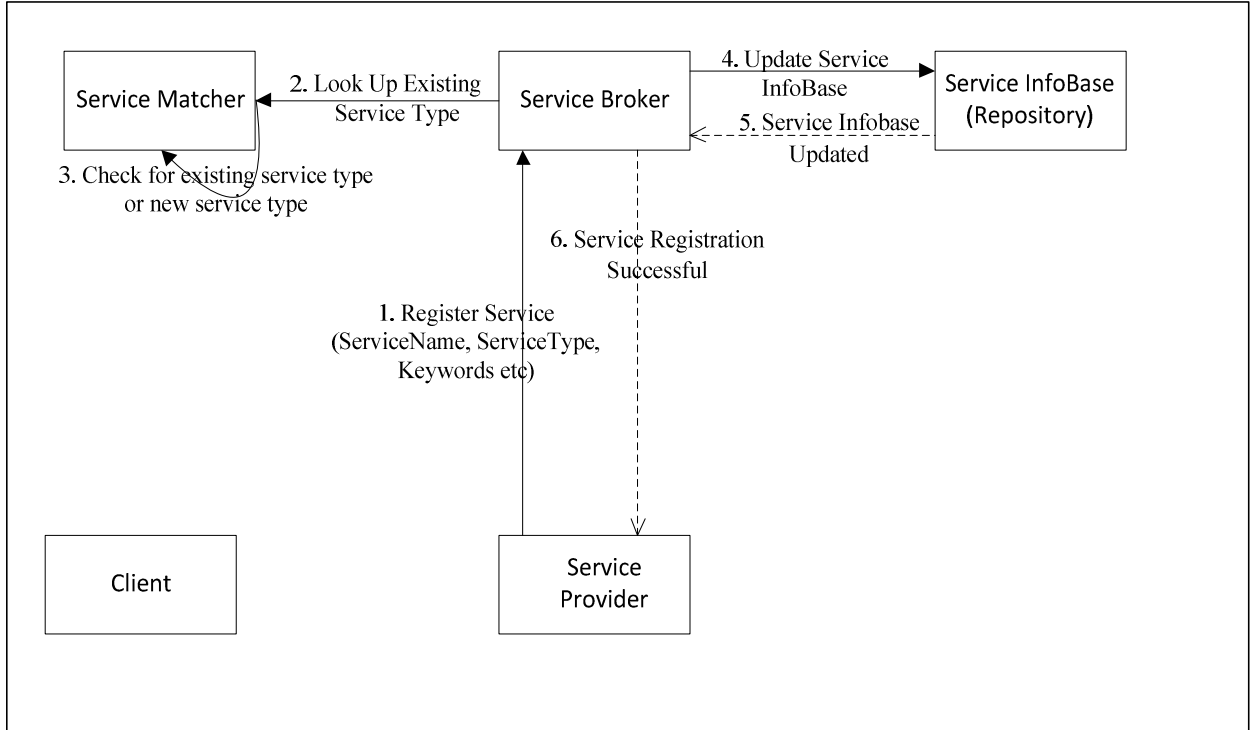


Figure 4-4 Collaboration Diagram for Registration of Service

## 4.2. Discovery of Service:

After the services are registered the most important part for the client is to discover the services. This can be achieved by advertising the services. There can be two possible ways of doing it either by broadcasting or the user can retrieve all services by requesting the service broker. As the web service architecture is been used in our application and it is not capable of broadcasting periodically until it is being invoked by any process or a client. The services can therefore be discovered by the client by simply sending a request to the service broker. The process is explained in collaboration and sequence diagram below and the respective steps are defined accordingly.



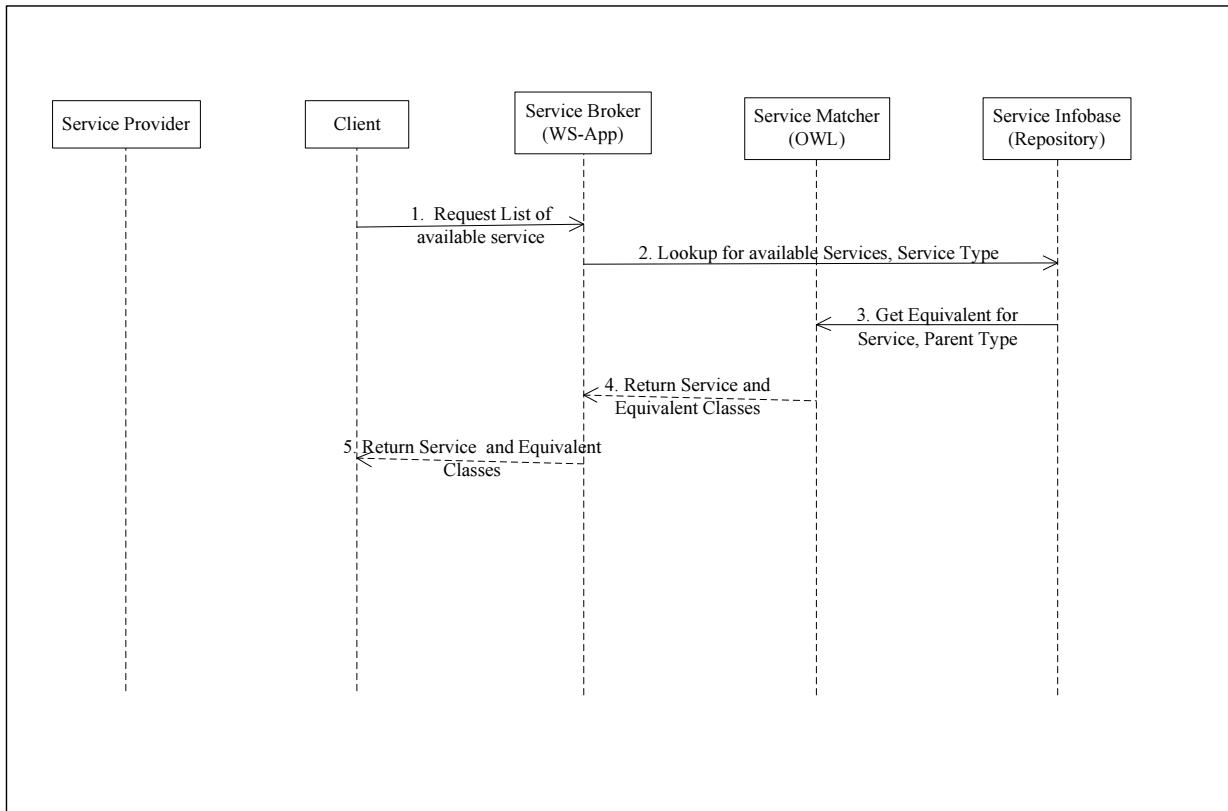


Figure 4-5 Sequence Diagram for “Discovering services”

The event execution is as follows

- 1) Client request list of available services for specific service types i-e communication, Music etc.
- 2) The service broker on receiving the request performs a look up for the available services in the repository. The services are saved along with their parent id which makes it easier for the user to select the appropriate service.
- 3) On getting the requested services from the repository the service matcher will look for any equivalent class defined for the service as well its parent type to provide the client which might be of interest to him.
- 4) Once retrieving the desired services and its equivalent classes the services will be returned back to the service broker.
- 5) The service broker will return the list of available services along with their equivalent type that can be either logical or any service name defined in different language that is providing the same functionality.

The collaboration diagram for discovering a service is shown below in Figure 4-6

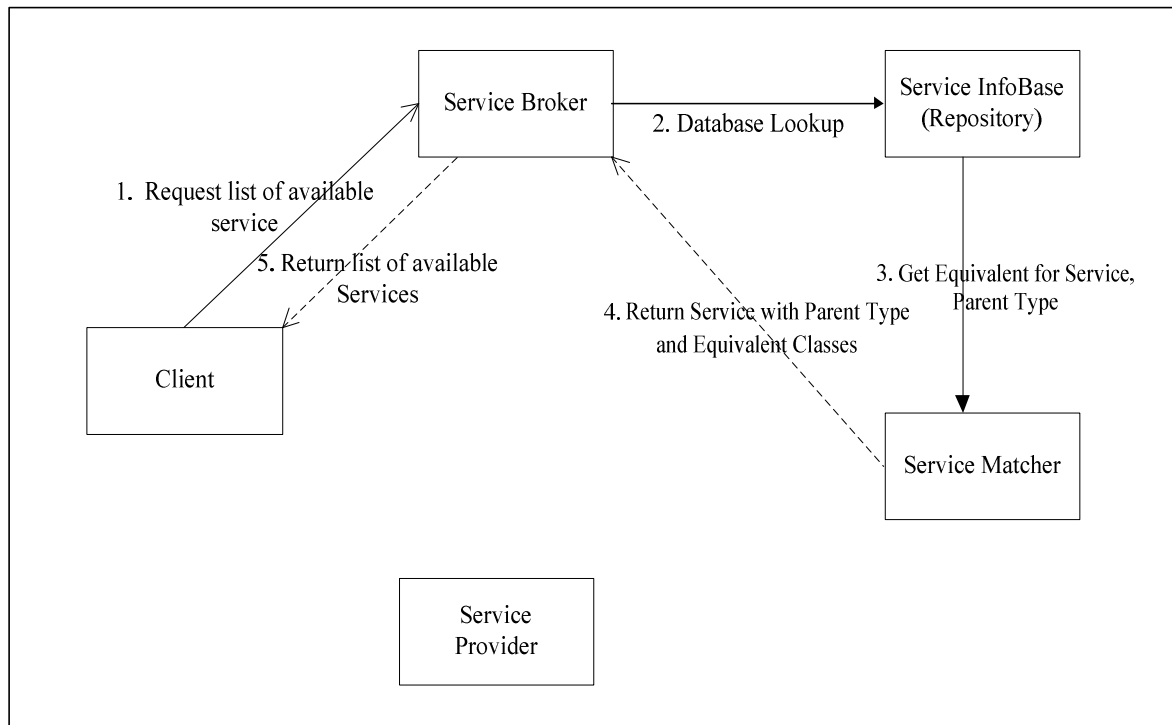
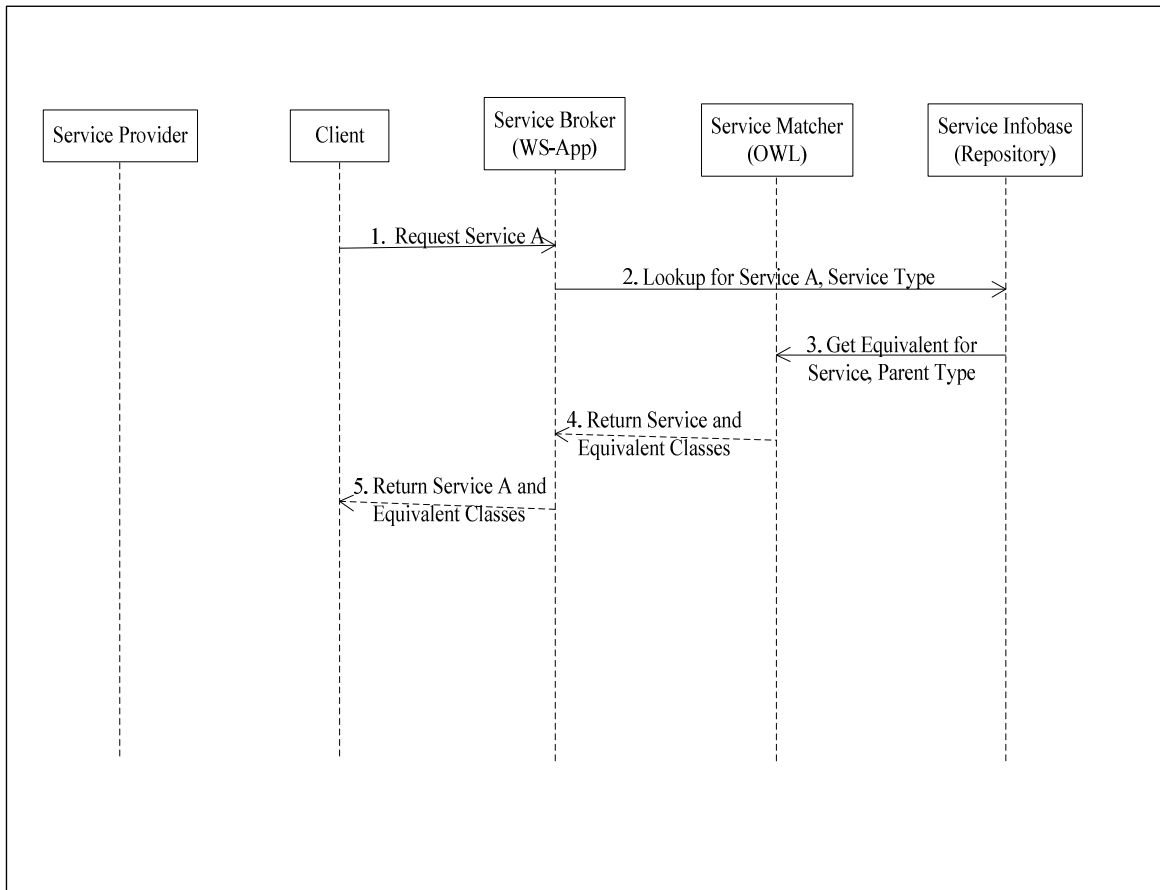


Figure 4-6 Collaboration Diagram for “Discovering Services”

### 4.3. Select a Service

Once the list of available services are discovered a client need to select the appropriate service as per his requirement. This can be done by the client going through all services that were retrieved and then making a decision manually on which service to invoke. However this architecture provides the client with an option to select a service by specifying precisely the service name and service type. It is known as service search or service lookup. This greatly increases the efficiency of the system as the network automatically search and find the service for the client hence relieving him from this burden. However this approach holds a major drawback as the client is unable to discover newly introduced services which may be of interest to him. Service lookup events are explained in sequence diagram below

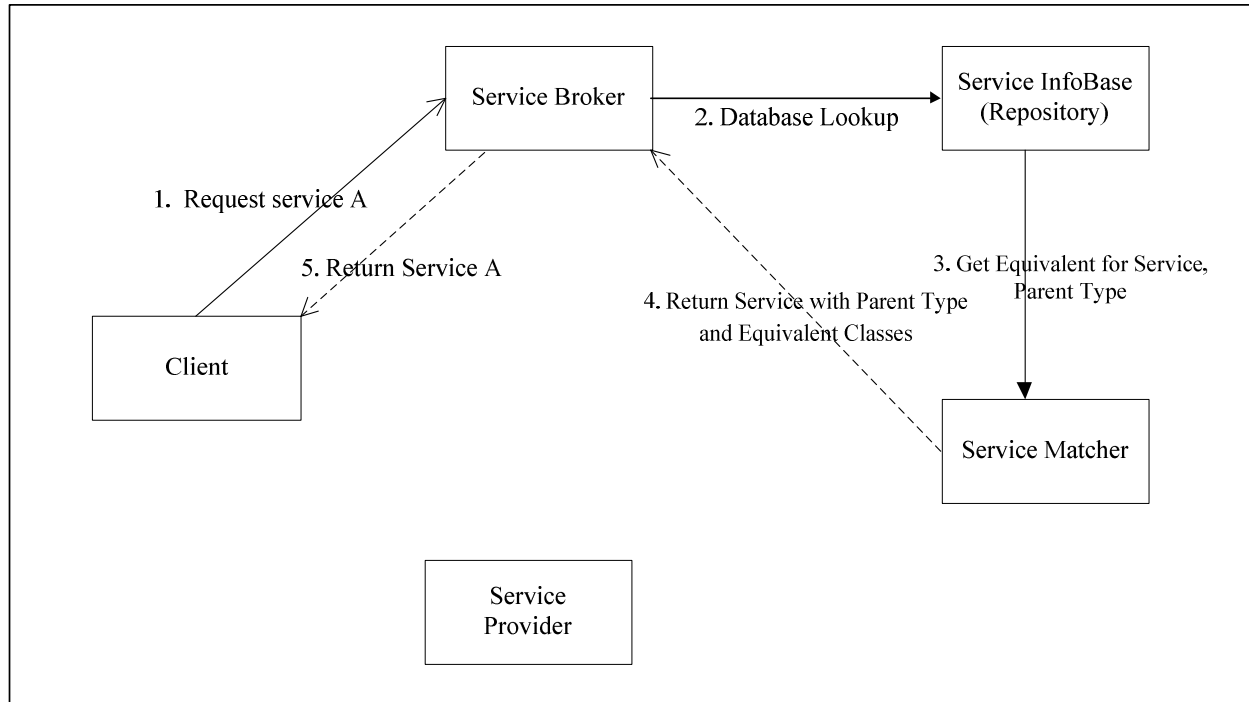


*Figure 4-7 Sequence Diagram for “Service Request”*

The events are executed as follows

- 1) Client request service A specifically with its service type. The service type is sent to avoid confusion as different services can have the same name so just to make sure that right service is retrieved.
- 2) The service broker on receiving the request performs a look up for the desired service in the repository.
- 3) On retrieving the requested service from the repository the service matcher will look for any equivalent class defined for the service as well its parent type
- 4) Required service and its equivalent class if any will then be returned back to the service broker.
- 5) The service broker will then return service A along with their equivalent type if any to the client.

The collaboration diagram for service type is shown below in Figure 4-8



*Figure 4-8 Collaboration Diagram for Service Type*

#### **4.4. Find A Service:**

Client can also find the desired services by searching based on the appropriate keywords. The service provider while registering a service, provides keywords that highlight the main feature of the service. These keywords can be used later on in order to return the appropriate service to the client e.g if a client search for food, dining etc. a hotel service defined with such keywords will be returned. The sequence diagram for finding a service is as follows

- 1) The client sends a request to service broker by providing appropriate keywords in order to find a service.
- 2) The service broker forwards the information to the service info base for service lookup.
- 3) Service InfoBase lookup for the service by searching against the service and the keywords that were defined while registering it by the service provider.
- 4) Once the service is retrieved it might also be of interest for the client to have different service that are equivalent to the service that is retrieved through the keyword search so a lookup will be carried out accordingly for the service type equivalent or its parent type equivalent.

- 5) Services retrieved are then sent to the service broker so that they can send back to the client.
- 6) The client is presented with the service response on bases of his key word search.

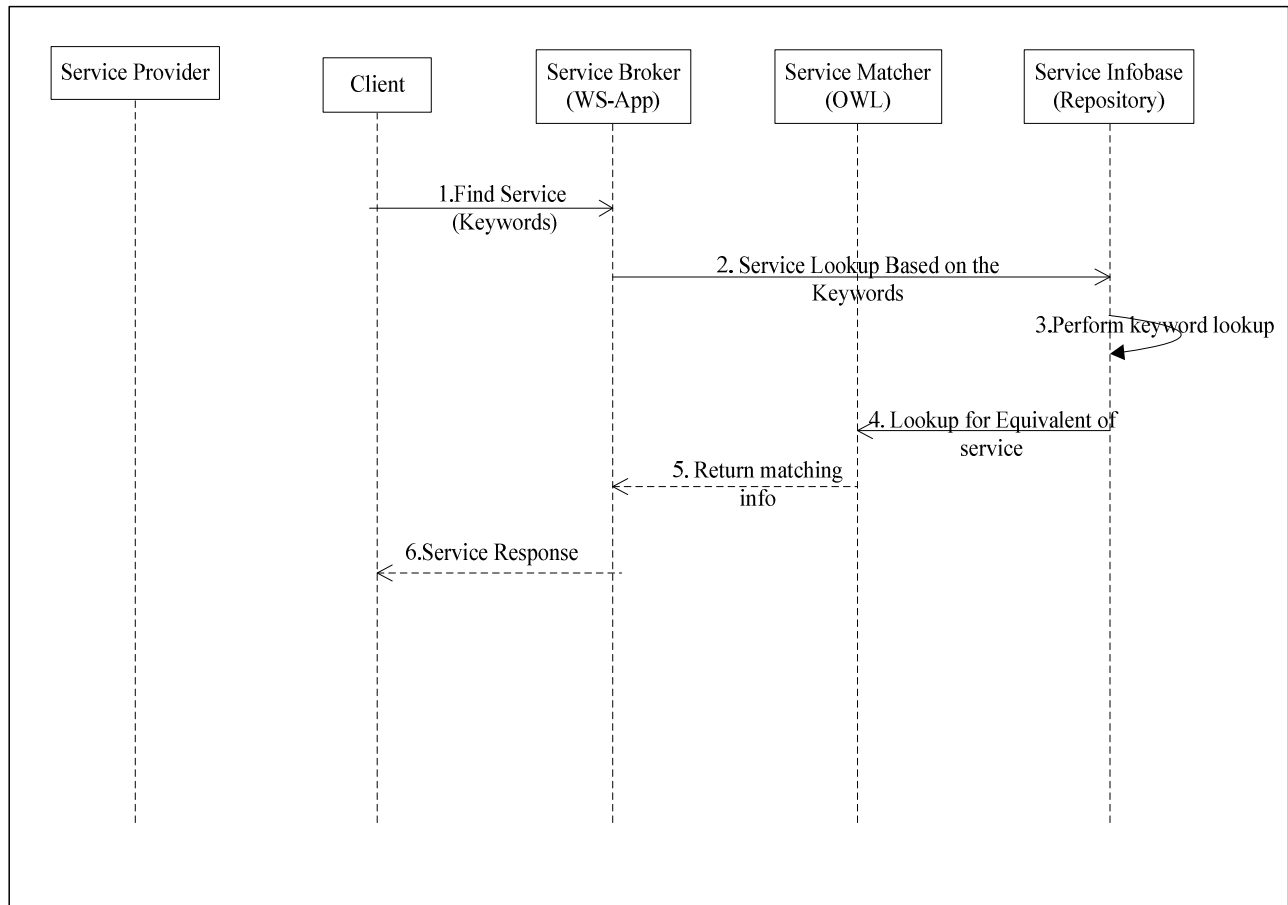


Figure 4-9 Sequence Diagram for “Finding a Service”

The collaboration Diagram for finding a service is presented below in Figure 4-10.

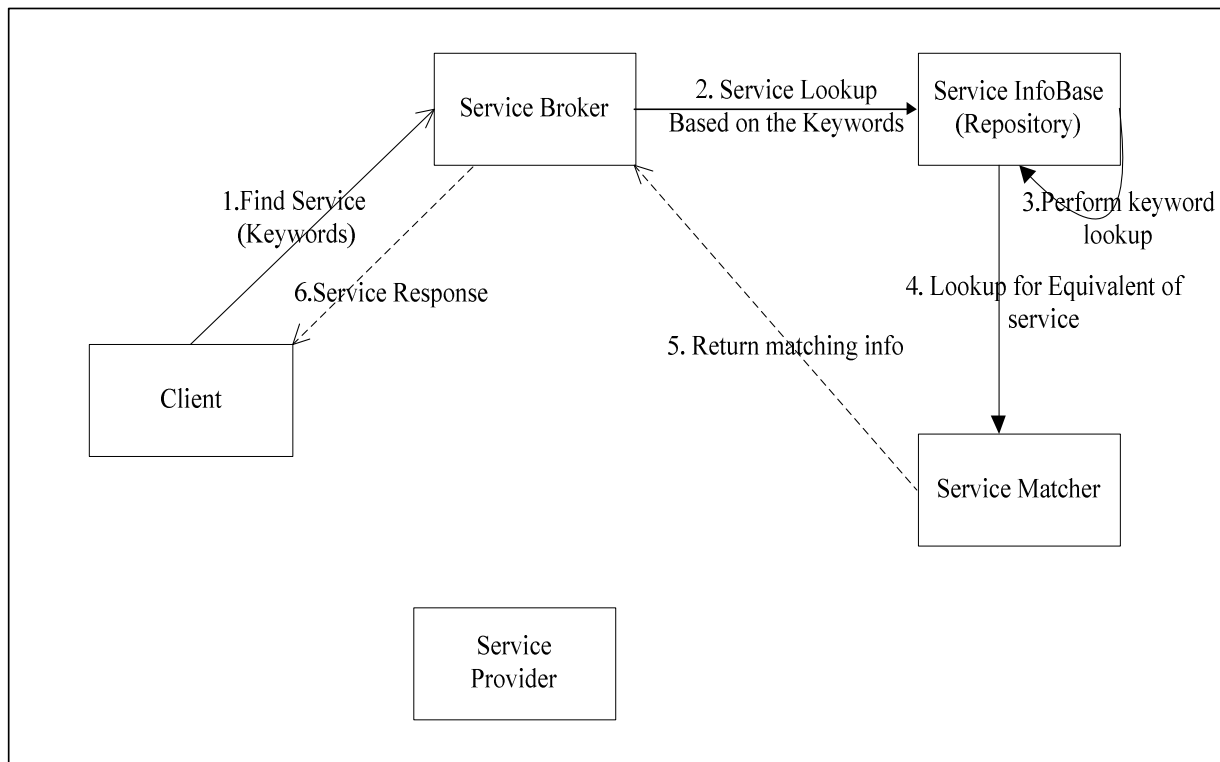


Figure 4-10 Collaboration Diagram for “Finding a Service”

# Chapter 5

## **5 Implementation**

*“The function of good software is to make the complex appear to be simple.”*

**Grady Booch**

This chapter describes implementation details of the service discovery for future mobile environment. Brief overview of complete architectures, classes and database is presented in order to make it simpler to understand that how the system has been implemented. However the four use cases that are specifically presented in design are presented on functional level as it is of interest. For further implementation details refer for class and database overview which is part of this report in Appendix A. Chapter 5 will be divided into Following 4 sections listed below

5.1) System Overview

5.2) Class overview

5.3) Database Overview

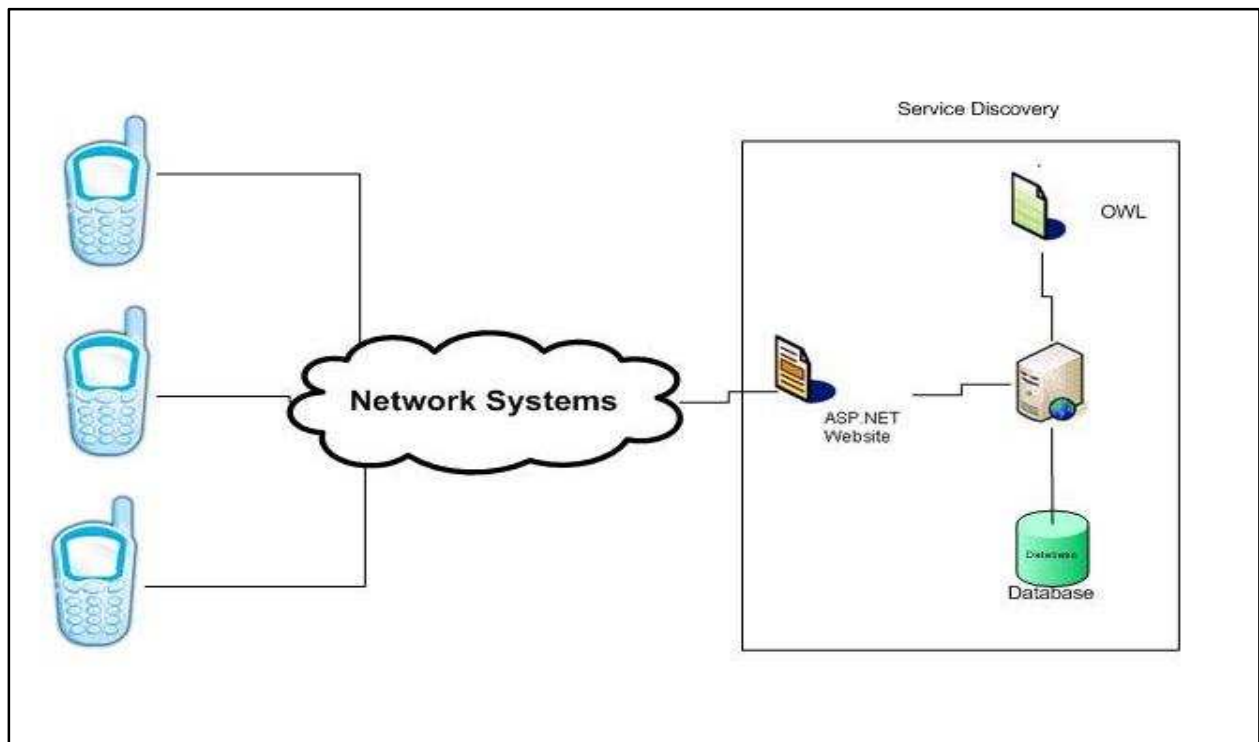
5.4) Methods Descriptions

## 5.1. Overview of the system architecture

The major components of the architecture are:

### 5.1.1. Web Service Client–

This component is an application running on the remote mobile terminal. The Client on behalf of the user can ask/search for a service. It can also on behalf of the Service Provider introduce and register a new service. The protocol used between the Client and the actual XML Web Service is SOAP (Simple Object Access Protocol) (35).



*Figure 5-1 Future service discovery system architecture*

Components of the service discovery system consisting of:

- i) **ASP .Net (36)Website (XML Web Service)** This service acts as an interface between the Client/Service Provider and the System Server. The IIS Internet Information Services 5.01 (37) is customized in order to publish the ASP.Net pages on the Internet.
- ii) **Database** – This component acts as a service repository which stores the services registered by the service provider.



- iii) **OWL (38)file** – is used for semantic matching with the OWL Lite (EquivalenceClass and ParentType).

### **5.1.2. Development Environment**

The hardware as well as the software components used for the system implementation are described below:

#### **i) Hardware**

A computer with installed Windows XP is used as a server. The server can process several request from multiple clients at parallel. The Client can request the server by using any platform or browser to retrieve the desired services. The Server contains an original Intel Processor (2.60 GHZ) along with 2 GB of RAM in order to ensure fast operation of the system.

#### **ii) Software**

##### **5.1.1. Operating System**

The operating system is selected by considering various technical aspects which are discusses below.

##### **Server side:**

Since the application is developed using .Net technology, the .Net framework version 2.0 (39) is installed on the server. Microsoft Windows based operating system is the minimum requirement for this version of .Net framework. Therefore Microsoft Windows XP (40) was installed on the server.

##### **Client:**

The Client is independent of any architecture and hence can access the services by using any industry standard Internet browser from any operating system that has access to the Internet.

##### **5.1.2. Programming languages:**

For the implementation of this project Microsoft Visual Studio 2005 IDE for C# (41) was used. The application was developed in C# whereas the website was developed using ASP.Net technology.

##### **C#:**

C# was mainly used as an implementation language because of its convenient programming capabilities, the object oriented paradigm it supports, type-safety and wide range of libraries available as a part of .Net framework. Another reason to use C# was the OwlDotNetApi that was

mainly used to access OWL file in order to define parent as well as equivalent classes for service types. The detail of this API can be found on (42).

#### ASP.Net:

ASP.Net is used for the XML web service and the web form interface for accessing the web service. There were certain reasons for using ASP.Net during implementation phase

- 1) For building XML web service because it makes exposing and calling web services very simple.
- 2) For Client website because of the flexibility of interface it provides and support for the mobile devices. By building a website in ASP.Net one has to write the code once and the ASP.Net automatically generates pages based on the device they are called.

#### **5.1.3. Support for ontologies:**

As the semantic meaning in this project of service discovery is being achieved by using ontologies. OWL (Ontology) file was continuously monitored manually by using Protégé 4.0.2 (43) in order to check the updates. OWL Lite is chosen in the implementation of this project due to its simplicity and the dynamic nature of the project even though it is known that it has some limitations as to compare with the OWL DL and OWL Full (44). Since the services can be introduced by anybody and there is no control mechanism on introducing the services, therefore it is required to avoid complexity in the system usage and for that only the EquivalenceClass and the ParentType properties is used which can be accomplished by the OWL Lite. More over the owl has another important feature which is the capability to support multiple languages (45) which is very useful in context of our system.

### **5.2. Class Overview**

There are eight classes in total that have been written for the implementation of the system. Each of the classes was used accordingly to fulfill different functionalities. The class diagram is shown in Figure 5-2 where the classes relationship and there functions can be seen. Each method and field functionality for each class is specifically mentioned in Appendix A1 and can be referred to for detail. A brief introduction to each class and its basic functionality is discussed below.

### **5.2.1. Base Class:**

The Base class provides the functionality to connect with the data base tables. In addition to connectivity it provides the database functions such as (Insert, update and retrieval). This class has a number of overloaded methods to extend the functionality. The Base class is inherited by three different classes to reuse the code.

### **5.2.2. DALService**

The Data Access layer service (DALService) class inherits the Base class and reuses its basic functionality to connect with the database, insertion and retrieval. Additionally it provides the functionality to access the service table in the database. This includes adding of equivalent service id, finding of service, retrieval of all the services on matching and insertion of service.

### **5.2.3. DALServiceType**

The Data Access layer service type (DALServiceType) class inherits the Base class. In addition to the base class functionality it also provides access to the servicetype table in the database. This class not only accesses the servicetype table but it has to ensure the consistency between the database table and the ontology.

### **5.2.4. DALKeyword**

The Data Access layer keyword (DALKeyword) class inherits the Base class for reusability of the code such as the basic connectivity with the database, insertion and retrieval. It provides additional functionality to access the keyword in the database. Moreover it ensures not to use the same keyword for two services. At the time of registering a service it first checks whether the same keyword is already stored in database for any other service. If any value is matched then instead of adding the keyword the keywordID is used to avoid the redundancy.

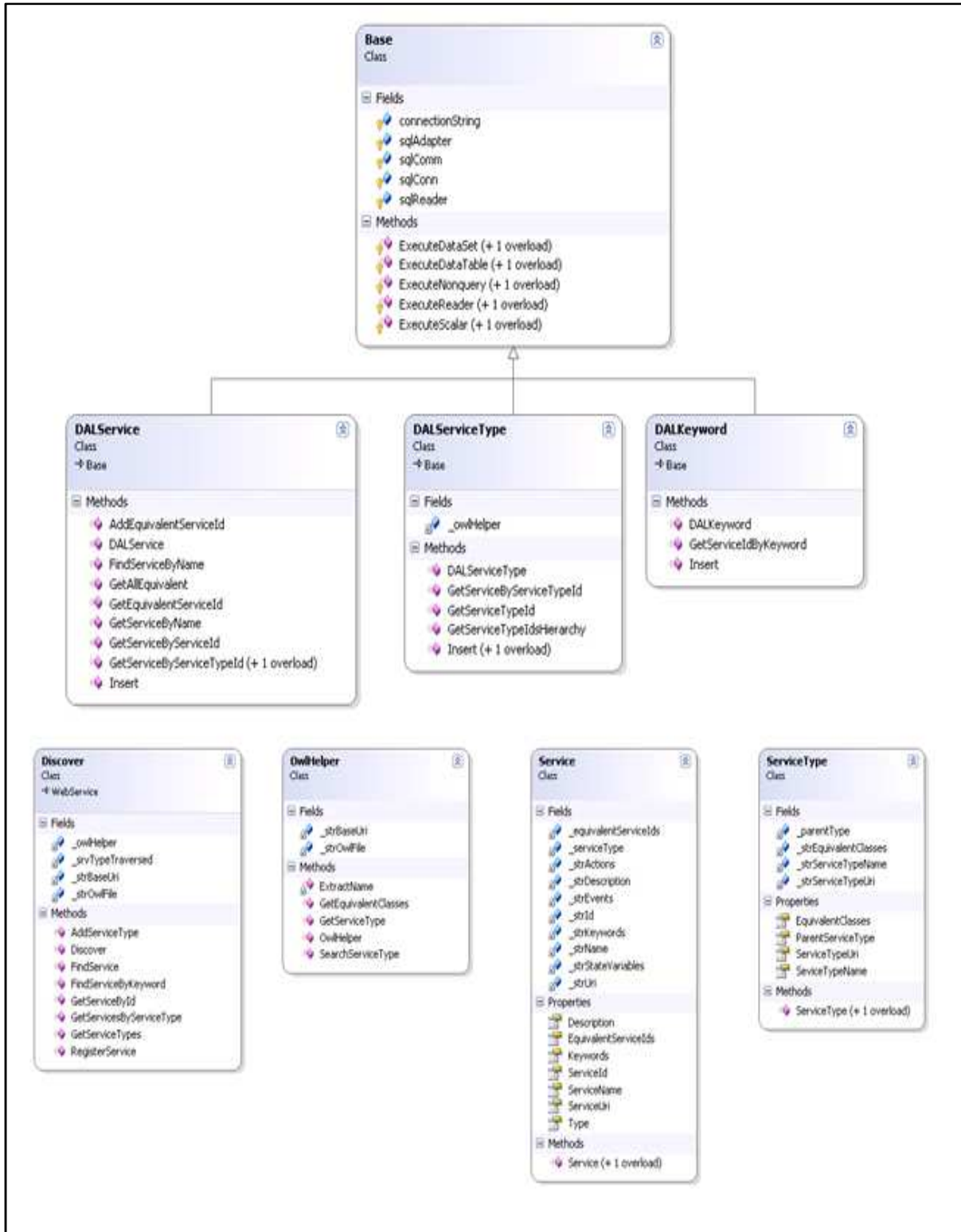


Figure 5-2 Class Diagram

### **5.2.5. Discover**

The Discover class contains the overall logic of web service, the methods of discover class are directly exposed to the client to use them. The ASP.net page also consumes this web-service. Discover class is performs most prominent functionalities by calling other class methods in the same package, the discover class performs different tasks such as adding new service type, finding service by name, keyword, serviceId and by service type. The discover class also performs the process of service registration.

### **5.2.6. OwlHelper**

OwlHelper class uses the OWLNET API to perform different tasks in ontology. These tasks contains searching of class from ontology on the basis of service type name, For every class in ontology contains an URI, this URI uniquely identify the class but this URI is a long string include the name of the class, OwlHelper is used to extract the name of class from the URI. Along this the OwlHelper returns all the equivalent classes from the ontology.

### **5.2.7. Service**

service class is a representation of each individual service which is being used to perform different functions associated with service such as service registration, service retrieval and service matching. Service class includes overloaded constructor, properties (attributes such as description of service, equivalent services, associated keyword, name of the service, address URI) and different fields.

### **5.2.8. ServiceType**

The servicetype class is a representation of each individual servicetype which is being used to perform different functions associated with servicetype such as servicetype registration, servicetype retrieval and servicetype matching. Servicetype class includes overloaded constructor, properties: (attributes such as All equivalent classes in the ontology, Parent (service type) of the servicetype, name of the servicetype in the ontology, associated keyword, unique URI in the ontology and different field.

### 5.3. Database Overview

Microsoft SQL server 2005 enterprise edition (46) was used as repository during implementation of this project. The database was used to store all the available service, service type, service URI, keywords for each specific service as well as equivalent for each service or service type if any exist. A brief overview of each table is discussed below whereas the relationships between these tables can be seen in ERD diagram illustrated Figure 5-3. For more detail on each table sample data is illustrated for each table separately in Appendix A.2 which can be referred to if required.

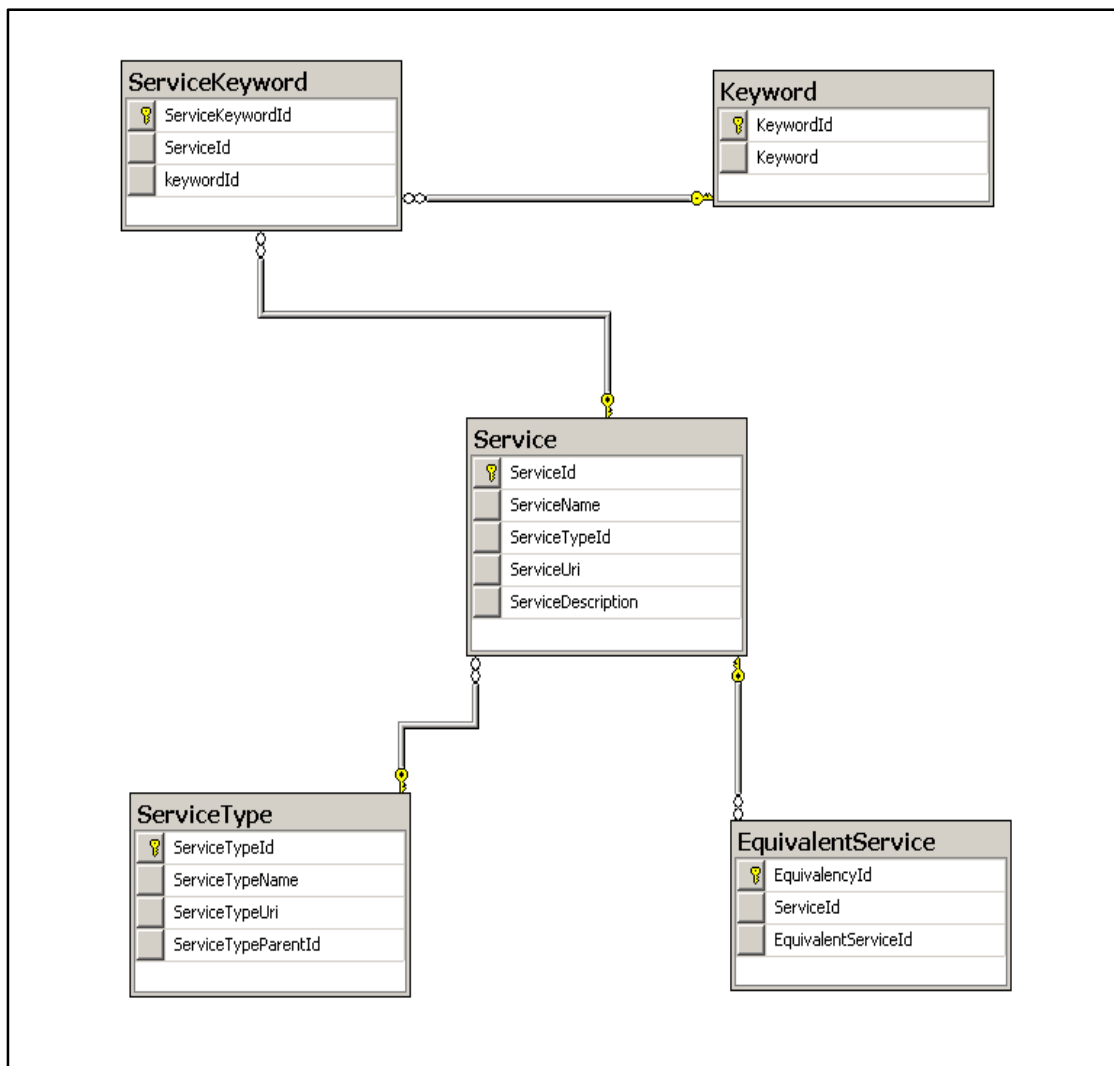


Figure 5-3 Entity Relationship Diagram

### 5.3.1. Service

The table service is used to store the data for every registered service. This table typically contains each service id which is generated by the database at the time of registration of service. The second field is ServiceName which is a string value to store the name in Unicode (the Unicode is used to support characters of different languages. For any service there must be parent service which describes the belonging type or the classification of service, this association is being done by using the servicetypeId as a foreign key from the table of serviceType. Until now there is no field which define the address of the service from where a service can be invoked this is done with ServiceURI to store the URI of a service. To ensure that the user has selected appropriate service there is a field called ServiceDescription to store brief description of the service which could be helpful to select the most appropriate service.

### 5.3.2. ServiceType

This table is used to store every service type which is being inserted in the ontology. There are two reasons for doing this.

- i. First it used to ensure the relationships between the service and their service type along with the equivalent services, this is done to reduce the complexity of the system.
- ii. The second reason is to increase the performance of the system. By decreasing lookups in the ontology which take longer time as compare to database lookup.

The service type table is used to maintain the record of the classes which has been inserted in the ontology at the time of registering service type. At the time of registration of service type the database allocate an auto-generated id which is stored in ServiceTypeId field. The service name could also be retrieved from the ontology but due to the performance issue it is better to store in database field instead of ontology, the serviceTypeName is used to store the name of the service type. The servicetype shows a relation between parent and child including the grandparent relation, for grandparent a servicetype could be child of another service type. To achieve the grandparent relation the ServiceTypeParentId is used, this relation is not only used for the grandparent but it is also used for the semantic application. There must be some link to identify

service type from ontology this is done by storing the unique URI for each class in the ontology in ServiceTypeURI.

### **5.3.3 EquivalentService**

One of the goal of service discovery is to return the services which has partial features. This type of services are stored in the system with the title of equivalent services. To store the equivalent service there must be two services (ServiceId and EquivalentServiceId) to be equivalent of each other and uniquely identify the relation i.e. EquivalencyID.

### **5.3.4. ServiceKeyword**

The keyword table contains the auto generated KeywordId for every keyword. This table is used to store the keyword for the later usage of keyword.

### **5.3.5. Keyword**

The service keyword table has been designed to avoid redundancy of keywords. For this the table stores association between the keywords and services. To accomplish this it contains auto generated ServiceKeywordId for every association between the service (ServiceType) and keyword (KeywordId).

## **5.4. Main methods**

As four use cases were specifically explained in our design, these methods are explained in detail. Each class and its methods are mentioned separately in Appendix A.1, whereas database sample data is presented in Appendix A.2 which can be referred to for detail lookup if required. The four methods are described in detail below.

### **5.4.1. Register Service:**

**Definition of the method:** This method is used to register a service, for registration few things has to be provided such as

- Service name (It could be any name which give an idea about service)
- Service URI (contains the URI/ address of the service to invoke)
- Service description (Short description that can represent a general overview about the service)



- Keyword ( this could be different words which help user to search about service e.g. for taxi service one can insert the keywords such as: public transport, travel )
- Service type (service name alone is not sufficient to define the type of service because two services can have same name they could be differ only from the type they belong the service type makes the difference)
- Available services (it shows all registered services for the selected type. The user optionally selects one of the services to make them equivalent at service level).

### Functionality:

For Registration of service the first method called is `Discover.RegisterService ()` this method calls all the necessary method. The method follows the loop illustrated in Figure 5-4.

Once the user enters the service name and other service parameters, the user also select the service type. For the selected service type the system calls the method `_OWLHelper.SearchServiceType ()` which returns the object of service type and pass it to `DALServiceType.GetServiceTypeId`.

```
DALServiceType dalServiceType = new DALServiceType();

int typeId =
dalServiceType.GetServiceTypeId(_owlHelper.SearchServiceType(srv.Type
e.ServiceTypeName))
```

Once the `ServiceTypeId` is returned then the service is registered from the following code.

```
dalService.Insert(string ServiceName,int ServiceTypeId,string ServiceURI,string ServiceDescription);
```

Optionally a service can be declared as equivalent to any other service if user selects any service that is equivalent to it. This can be accomplished by:

```
dalService.AddEquivalentServiceId(newServiceId,
srv.EquivalentServiceIds[i]);
```

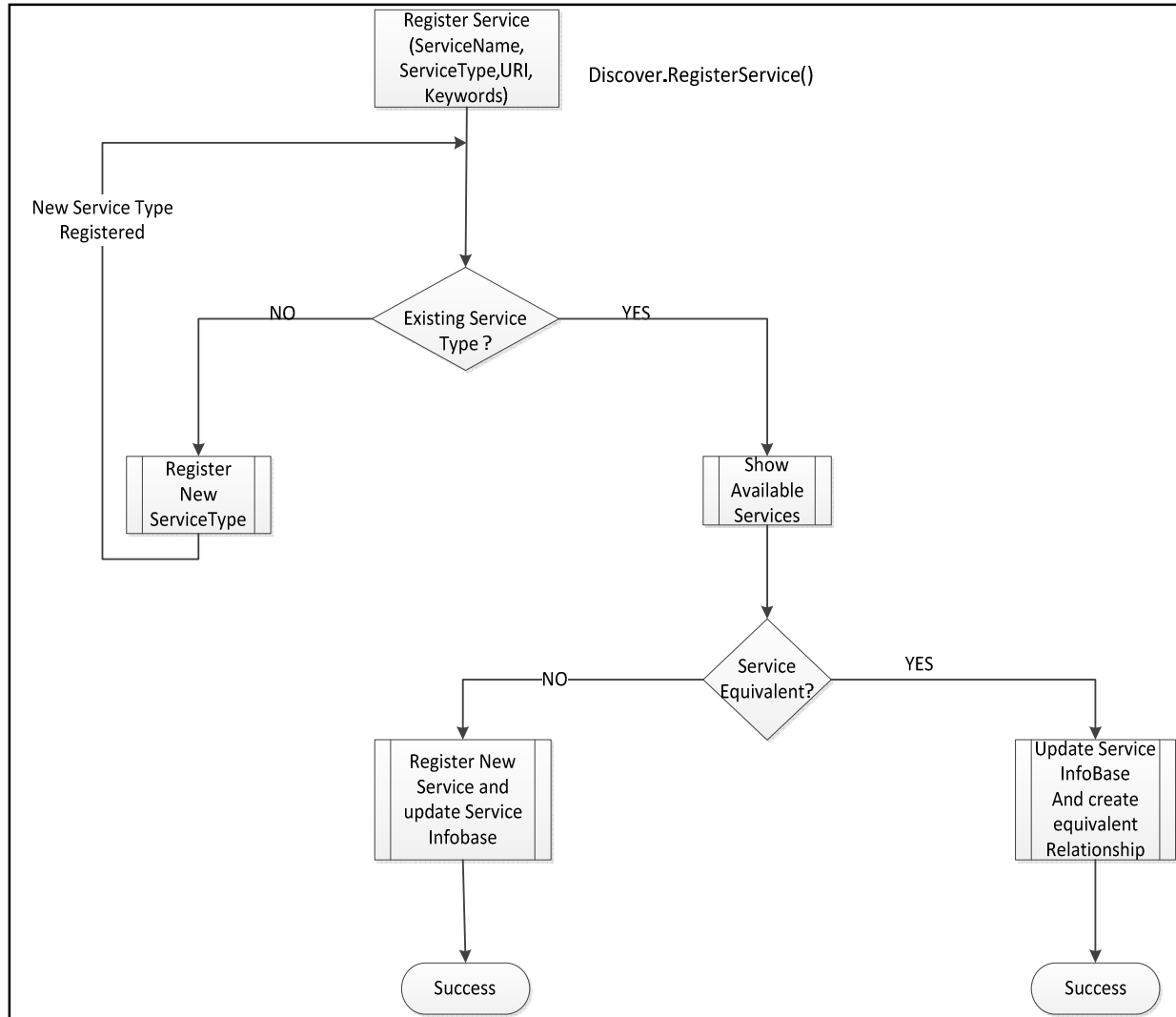
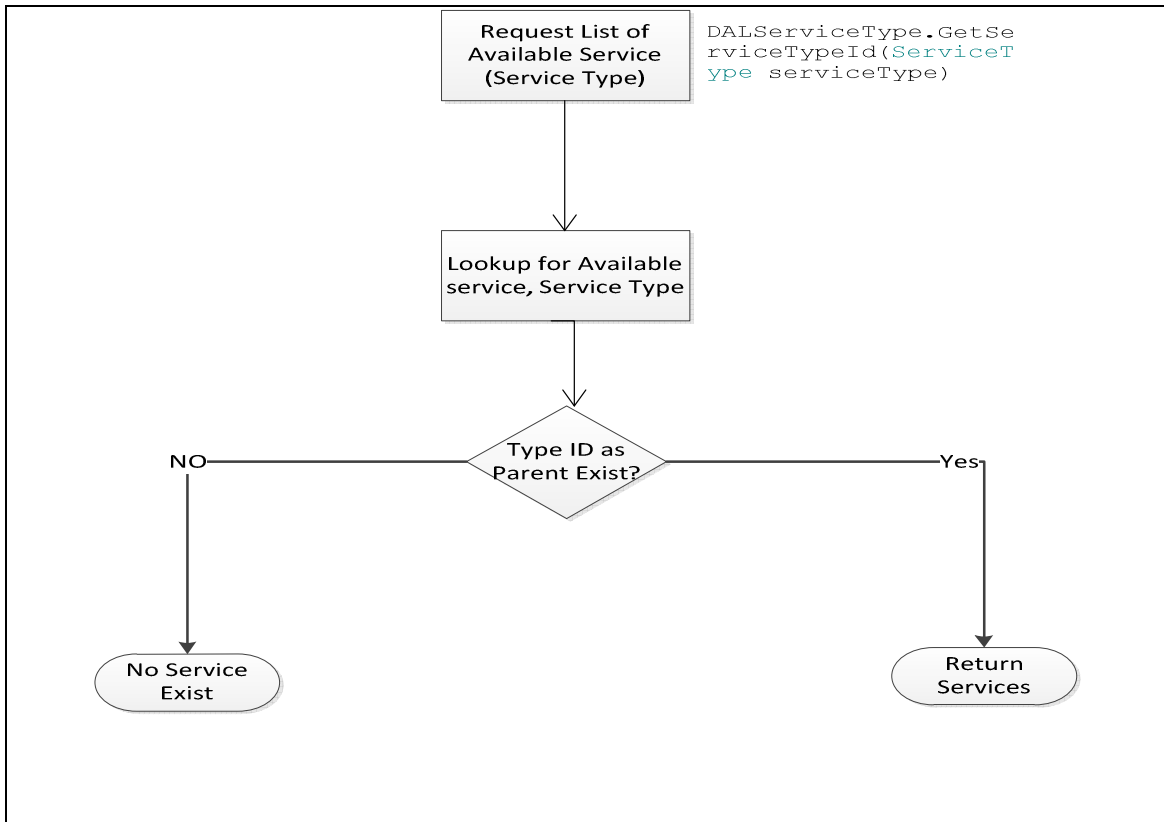


Figure 5-4 Main Loop followed for Registration of Services

#### 5.4.2. Lookup by Service Type:

**Definition of the method:** This method is used to lookup services by service type. The system returns all services of given type. This required a parameter of service type. Service Type Lookup follows loop which is being illustrated in Figure 5-5 below.

- Service type (the user enters the service type)



*Figure 5-5 Main loops for Service Type lookup*

### Functionality

First the system search for given service type id from database by:

```
DALServiceType.GetServiceTypeId(ServiceType serviceType)
```

Once the Service type id is returned, the system will look for all the services whose service type id (Parent id) is the one returned by the GetServiceTypeId().

This task is accomplished by:

```
DALServiceType.GetServiceByServiceTypeId(int ServiceTypeId).
```

After retrieving the values the result is displayed on the ASP Page for the client.

### 5.4.3. Lookup by Keyword:

**Definition of the method:** This method is used to lookup services by Keyword passed in argument. The system returns all the services associated with the keyword. This required a parameter Keyword.

- Keyword(the user enters the keyword)

#### **Functionality**

The system searches for the specified keyword by using the method. Keyword Lookup follow loop illustrated in Figure 5-6

```
DALKeyword dalkeyword = new DALKeyword();
```

```
DataTable dtServices= dalkeyword.GetServiceIdByKeyword(Keyword);
```

The method returns all the associated services and stored the result in a temporary datatable. Each service is returned by using:

```
DALService.GetServiceByServiceID()
```

After getting all the service Ids the system check for all the equivalent services, this accomplished by:

```
DALService.GetEquivalentServiceId()
```

This method returns equivalentIds, however from this equivalent id the services cannot be returned directly, therefore the system uses equivalent id to get service id, which can display all the services.

```
DALService.GetServiceByServiceID()
```

The lookup method avoids circular reference.

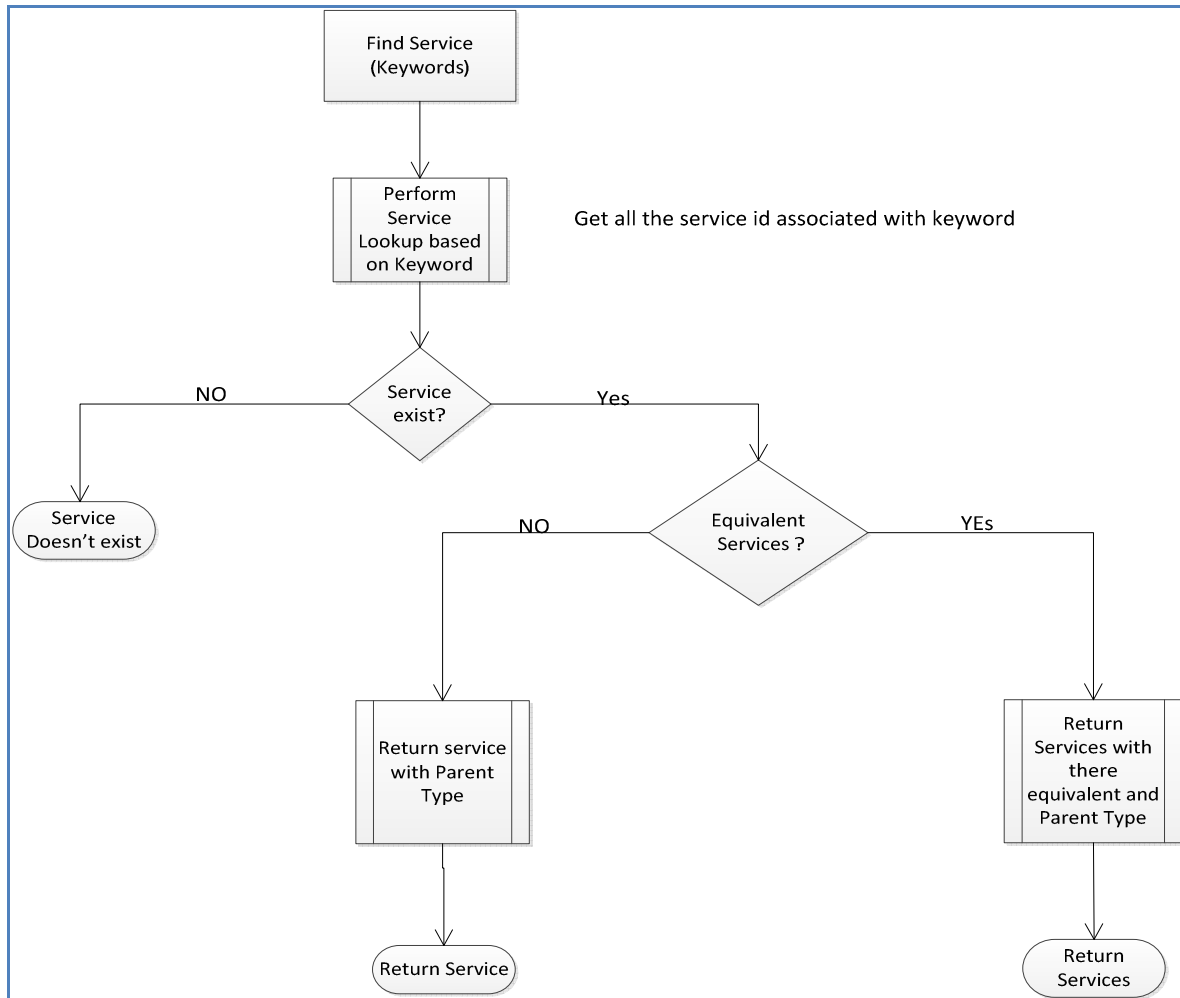


Figure 5-6 Loop followed for Keyword lookup

#### 5.4.4. Get Service by Service name:

**Definition of the method:** This method is used to lookup services by directly providing the service name.

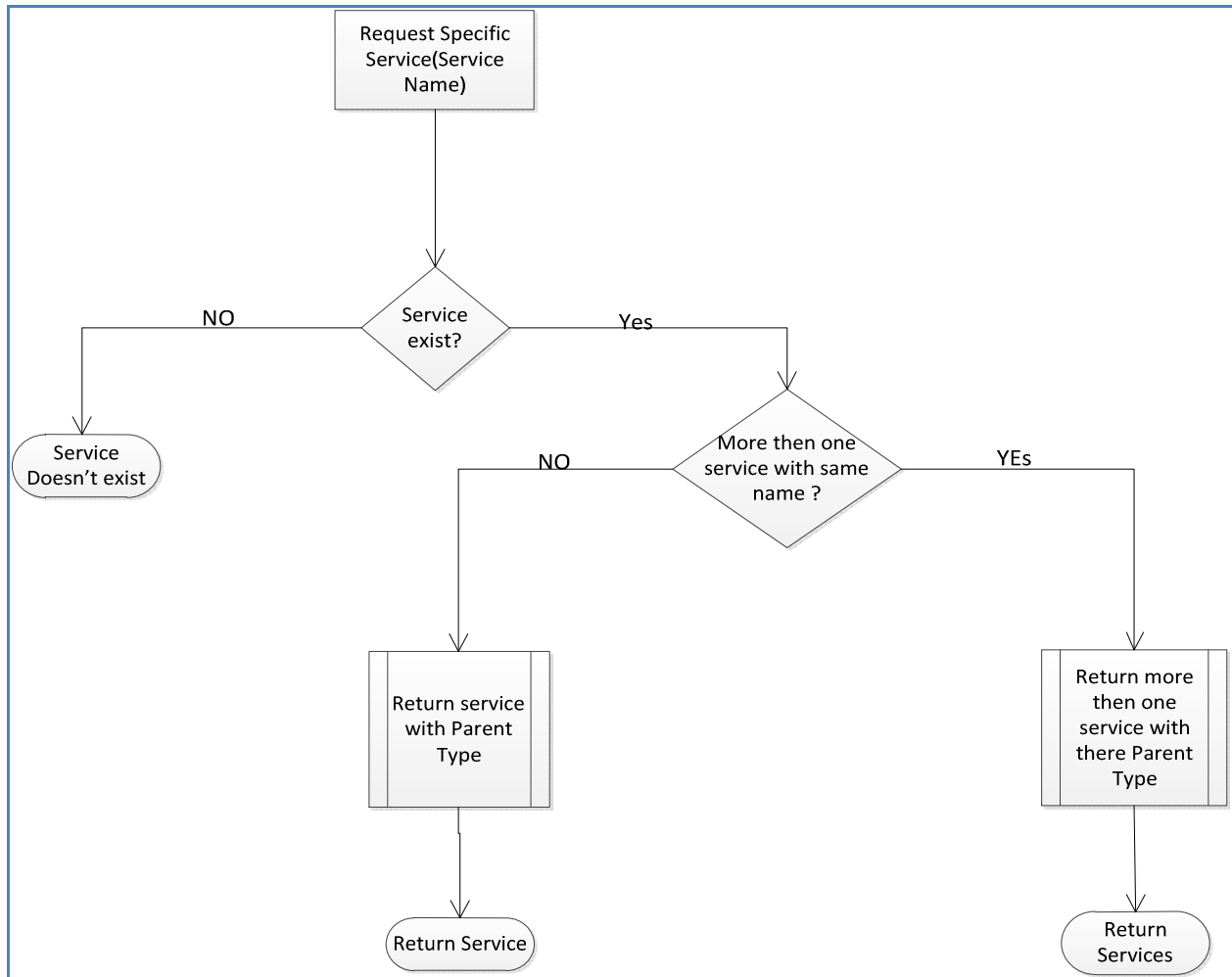
- Service Name (the user enters the service name this could be in English or any other Unicode).

#### **Functionality**

Services are being retrieved on the basis of passed service name as an argument. System retrieves all the services with matching of name. After the syntactic matching, system check for semantic matching which is searching of all equivalent class for a service.

```
DALService.GetServiceByName ( )
```

The above method returns a service (or more than one service with same name). After getting the required service the system checks for the service type (Parent type). At this point the system returns either single or more than one service with their ParentType. This method follows the loop specified in Figure 5-7.



*Figure 5-7 Loop for Service Name Lookup*

# Chapter 6

## 6 Testing

*“Program testing can be used to show the presence of bugs, but never to show their absence!”*

**Edsger Dijkstra**

In order to ensure that the implementation is working properly and is fulfilling the requirements specified in chapter 3 testing was conducted. This chapter is divided into two parts. Part 1 covers five use scenarios that are being set to ensure that the necessary requirements for the future service discovery are being achieved. The five scenarios are as follow

- Anybody can introduce a service at anytime
- Service name with multiple languages
- Same service with different name
- Different services with same name
- Partial matching services

Part 2 deal with the scalability issues and test that how does the system behaves with different number of services. There are five points of check that were laid down in order to note down the system behavior. The check points are as follows

- Service Repository with 50 services
- Service Repository with 100 services
- Service Repository with 250 services
- Service Repository with 500 services
- Service Repository with 1000 services

# Section I

## 6.1 Use Scenarios

*“But what is it good for?”*

**Engineer at the Advanced Computing Systems Division of IBM, commenting on the microchip, 1968**

### 6.1.1. Anybody can Introduce a service at anytime

The system is really flexible for any user to introduce a service anytime without any requirement for registration initially to post a service in the repository. As in traditional service discovery systems the service types are well defined and the user is not able to register a service type until the authority approve and update it at his own .This system allows the user to register any service type at anytime without such delays.

The screenshot shows a web interface titled "Service Discovery". On the left, there is a "Menu:" section with three links: "Register Service", "Register Service Type", and "Find Service". The main content area is titled "Register new service type". It contains a form with the following fields:

- "Service type name:" with a text input field containing the word "Education".
- "Service type:" with a list of options: "food", "Telephony", "book", and "transport", each preceded by a small square icon.
- "Equivalent type:" with a list of options: "food", "Telephony", "book", and "transport", each preceded by a small square icon.
- An "Add Service Type" button at the bottom of the form.

*Figure 6-1 User interface to register a new Service Type*

If a user wants to register a service for his university and no service type of education exist he can register a new service type of education as shown in Figure 6-2 below. If he feels that there should be further subtype of University in education he can add it as a subtype of service type as



shown in Figure 6-2 below. The user also has the option to define service type equivalent to any other service type by using the same interface and without any prior registration.

The screenshot shows a web interface titled "Service Discovery". On the left, there is a "Menu" with three links: "Register Service", "Register Service Type", and "Find Service". The main content area is titled "Register new service type". It contains the following fields and options:

- Service type name:** A text input field containing the text "University".
- Service type:** A list of service types with radio buttons. The options are "food", "Education" (which is selected and highlighted in blue), "Telephony", "book", and "transport".
- Equivalent type:** A list of service types with radio buttons. The options are "food", "Education" (which is selected and highlighted in blue), "Telephony", "book", and "transport".
- Add Service Type:** A yellow button at the bottom right of the form.

Figure 6-2 User interface to register a new Service SubType

Once the service types are registered, the user can register his university service by simply providing service name, service URI, service description, and selecting his service type. The procedure is elaborated in Figure 6.3 below

The screenshot shows a web interface titled "Register new service". On the left, there is a "Menu" with three links: "Register Service", "Register Service Type", and "Find Service". The main content area contains the following fields and options:

- Service name:** A text input field containing "NTNU".
- Service Uri:** A text input field containing "www.ntnu.no".
- Service Description:** A text input field containing "Norwegian University of Science and Technology".
- Keyword:** A text input field containing "Norwegian science university, best education, engineering".
- Service type:** A list of service types with radio buttons. The options are "book", "Education" (which is selected and highlighted in blue), "food", "lovestory", "transport", "ghoragari", and "Telephony".
- Available Services:** An empty list box with a scroll bar.
- Buttons:** Three buttons at the bottom: "Register Service", "Register Service Type", and "Home".

Figure 6-3 User Registration a new service with newly registered Service Type

Once the service types are registered the user can register his university service by simply providing service name, service URI, service description, and selecting his service type. The procedure is elaborated in Figure 6.3 below. The service is successfully registered as illustrated in

figure. Hence one of the basic requirement specifying that anyone should be able to register a service at anytime is successfully fulfilled and running. The service registered and retrieved is shown in Figure 6-4 below.

Service Name	Service Description	Service Type
NTNU	www.ntnu.no Norwegian university of Science and Technology	Education

Figure 6-4 User successfully registering the service received

### 6.1.2. Similar services in different languages.

If a client wants to find a service he can either do that by providing search criteria in English which is the default language or in his native language e.g a taxi service can have different names in multiple languages e.g

Taxi: {Teksi, Riksha, такси, ....}

If the Client want to search for an appropriate service and his search is based on word 'Taxi' or in his native language which is equivalent to 'Taxi' it will not only return taxi service define in English but all other services define in different languages and are declared independent to taxi..The process is illustrated in Figure 6-5 below.

Menu:

- [Register Service](#)
- [Register Service Type](#)
- [Find Service](#)

### Find service

Service Name:

Service Type:

Keyword:

Time taken: 62.5  
Services found: 5

Service Name	Service Description	Service Type
<a href="#">malayTeksi</a> <a href="http://www.malayteksi.com">www.malayteksi.com</a>	malaysia teksi service	teksi Equivalent Type: taxi, transport
<a href="#">London taxi</a> <a href="http://www.londontaxi.com.uk">www.londontaxi.com.uk</a>	Taxi in London	Cab Equivalent Type: taxi, transport
<a href="#">whitecab</a> <a href="http://www.Whitecab.com.pk">www.Whitecab.com.pk</a>	white cab service, available in all major cities of Pakistan	
<a href="#">такси</a> <a href="http://www.такси.com">www.такси.com</a>	taxi service in bulgarian	
<a href="#">ٹیکسی</a> <a href="http://www.whitecab.com.pk">www.whitecab.com.pk</a>	best taxi service in pakistan	taxi Equivalent Type: teksi Equivalent Type: Cab, transport

Figure 6-5 Service in different languages equivalent to Taxi

As shown in Figure 6-5 above if a user try to find a service by entering taxi in its native language which is “urdu” in this case, all the services which are logically equivalent to it and are defined at time of registration are being retrieved and shown to the client. The same process is elaborated for a user who is searching for a taxi service in “bulgarian” by entering такси. The result is illustrated in Figure 6-6 below.

Menu:

- [Register Service](#)
- [Register Service Type](#)
- [Find Service](#)

### Find service

Service Name:

Service Type:

Keyword:

Time taken: 109.375  
Services found: 5

Service Name	Service Description	Service Type
<a href="#">malayTeksi</a> <a href="http://www.malayteksi.com">www.malayteksi.com</a>	malaysia teksi service	teksi Equivalent Type: taxi, transport
<a href="#">London taxi</a> <a href="http://www.londontaxi.com.uk">www.londontaxi.com.uk</a>	Taxi in London	Cab Equivalent Type: taxi, transport
<a href="#">ٹیکسی</a> <a href="http://www.whitecab.com.pk">www.whitecab.com.pk</a>	best taxi service in pakistan	
<a href="#">whitecab</a> <a href="http://www.Whitecab.com.pk">www.Whitecab.com.pk</a>	white cab service, available in all major cities of Pakistan	
<a href="#">такси</a> <a href="http://www.такси.com">www.такси.com</a>	taxi service in bulgarian	taxi Equivalent Type: teksi Equivalent Type: Cab, transport

Figure 6-6 Service in different languages equivalent to Taxch

It can be seen from the figures that more than one service are returned when a user is searching in its native language. It is because the services are logically equivalent and are defined so at the time of registration. It could be really helpful for the user considering the heterogeneous nature of the future mobile services. The services maybe unknown to him because of the different languages but if he is abroad and the services are logically equivalent he can still use them for his purpose.

### 6.1.3. Same services with Different Name

When a Client wants to find a service, for e.g a Restaurant service he will use any appropriate word either in standard English or his native language. The Restaurant service can have other different names for e.g.

Restaurant: {Restoran, Ristorante, Café, Bistro, Warung, ... }

If the Client enters the word Café, it will return not only the Café service but also other similar service which are defined with Café in other names which are equivalent to it and are specified at time of its definitions. This is illustrated in Figure 6-7 below.

- [Register Service](#)
- [Register Service Type](#)
- [Find Service](#)

#### Find service

Service Name:

Service Type:

Keyword:

Time taken: 62.5  
Services found: 3

Service Name	Service Description	Service Type
<a href="http://www.jumeirah.com/en/hotels-and-resorts/destinations/dubai/burj-al-arab/">http://www.jumeirah.com/en/hotels-and-resorts/destinations/dubai/burj-al-arab/</a> برج العرب	burj Al rab on of the best restaurant	restaurant Equivalent Type: ristorante Type: restoran Equivalent Type: bistro, food
<a href="http://www.paktive.com/Sabri-Nihari_144WB01.html">http://www.paktive.com/Sabri-Nihari_144WB01.html</a> میلاری نیری	famous dish of pakistan Nihari by sabri brothers	restaurant Equivalent Type: ristorante Type: restoran Equivalent Type: bistro, food
<a href="http://www.italycafe.com">www.italycafe.com</a> italy cafe	best italy cafe in the center of italy city	ristorante Equivalent Type: restaurant, food

Figure 6-7 same services with different name

### 6.1.4. Different Services with Same Name

Since in the future ubiquitous communication systems the service can be anything and introduced by anybody, there can be a possibility of having different terms and perspectives of knowledge of the service from the user and service provider. The future service discovery supports the function of having different services with the same name as shown in Figure 6-8.

**Service Discovery**

---

Menu:

- [Register Service](#)
- [Register Service Type](#)
- [Find Service](#)

**Find service**

Service Name:

Service Type:

Keyword:

Time taken: 46.875  
**Services found: 2**

Service Name	Service Description	Service Type
<a href="#">Book</a> <a href="http://www.amazon.com/harrypotter">www.amazon.com/harrypotter</a>	Harry Potter Novel story	Novel, book
<a href="#">book</a> <a href="http://www.eazyjet.com/book">www.eazyjet.com/book</a>	online Booking service EazyJet airline	EazyJet, Plane

*Figure 6-8 Different service with same name*

As in the above example, the word Book can refer to a type of Book service (for e.g. buying online book or information about a book). There can also be other meanings of Book service which may refer to a Reservation service. Even though the future service discovery allows having different services with the same name, the ambiguity and confusion can still be avoided via the details of the Service Description and Service Type returned during the service search.

### 6.1.5. Partially Equivalent Service

The future service discovery is using a semantic matching instead of just syntactical match as the one used in the existing service discovery. This is very important especially in the situation where no equivalent service is available but there exist service, which have more additional functions than the requested one – partially match service.

The future service discovery introduces the use of service sub-typing by having ParentType and attributes ParentType in the Service Type description template.

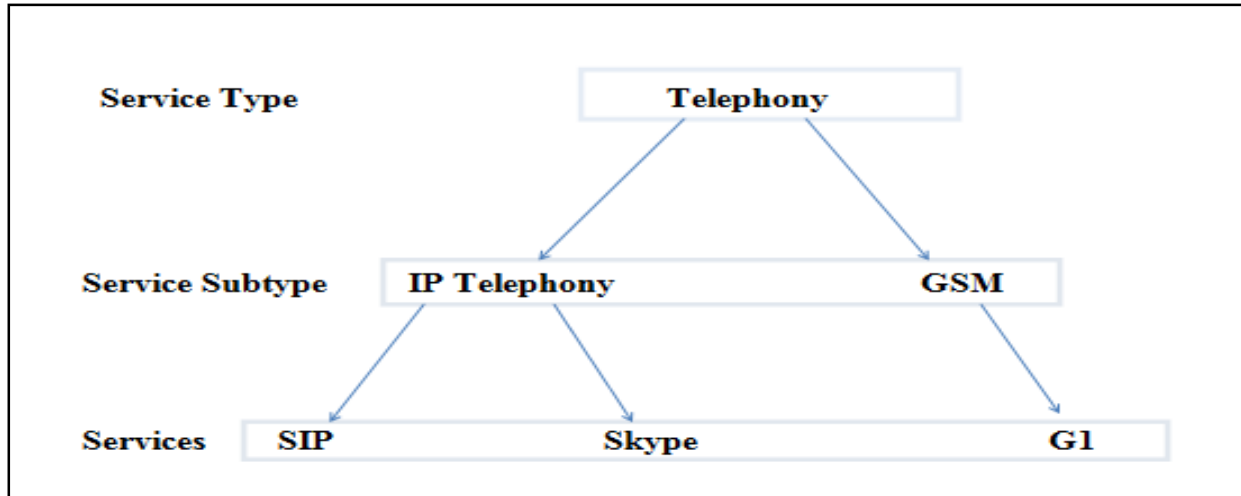


Figure 6-9 Parent type and Subtype relation for telephony

As illustrated in Figure 6-9, when the Client is asking for Telephony service; Skype, SIP and G1 are returned because they are grand children of Telephony (Skype and SIP are subtype of IP Telephony while G1 is subtype of GSM) and have inherited all the characteristics of Telephony. In this case the Skype, SIP and G1 are having similar functions (voice call – which is the generic Telephony features) but some of them have more or different additional functions (for e.g. Skype has video call feature) and they are also different in service components and service implementations.

### Service Discovery

Menu:

- [Register Service](#)
- [Register Service Type](#)
- [Find Service](#)

#### Find service

Service Name:

Service Type:

Keyword:

Time taken: 62.5  
Services found: 3

Service Name	Service Description	Service Type
Skype	<a href="http://skype.com/">http://skype.com/</a> make free voice and video call on internet	IP%20Telephony, Telephony
SIP	<a href="http://www.sipforum.org/">http://www.sipforum.org/</a> To make free calls	IP%20Telephony, Telephony
G1	<a href="http://www.gsmworld.com/">http://www.gsmworld.com/</a> To make payed voice calls from your mobile phone	GSM, Telephony

Figure 6-10 Retrieval of services in telephony

However, if the Client is asking for a specific Skype service, only the Skype is returned and nothing else. This is as illustrated in Figure 6-11.

## Service Discovery

Menu:

- [Register Service](#)
- [Register Service Type](#)
- [Find Service](#)

### Find service

Service Name:

Service Type:

Keyword:

Time taken: 15.625  
Services found: 1

Service Name	Service Description	Service Type
Skype - <a href="http://skype.com">http://skype.com</a>	make free voice and video call on internet	IP%20Telephony, Telephony

Figure 6-11 Retrieval of service by service name

## Section II

### 6.2. Scalability

*“When you are stuck in a traffic jam with a Porsche, all you do is burn more gas in idle. Scalability is about building wider roads, not about building faster cars.”*

– Steve Swartz

As per the requirements specified previously a service should be discovered really fast. Ideally the system should therefore discover the services quickly. The retrieval time for the tests conducted is when a client requests a service from the service broker till the time it is being returned back to the client. The system behavior is expected to be stable with service name lookup being the fastest way to retrieve the service and keyword lookup being the slowest. The minimum time is always expected to be with service name search as the search will be conducted in database and no lookup in the ontologies is required in this case. In case of service type and keyword search it will consume more time to return the result back to client. Reason for this delay is Owl lookup which is required to determine ParentType and EquivalentType etc after database lookup is performed. Considering that tests were conducted to measure how does the system scale with the different number of services registered in repository. The system behavior was tested with system in following states

- 1) Service repository with 50 services
- 2) Service repository with 100 services
- 3) Service repository with 250 services
- 4) Service repository with 500 services
- 5) Service repository with 1000 services

Considering the internet architecture, fluctuation in response is expected for every process once repeated. Moreover it will also depend how much resources are being consumed at the server once a request is being received, therefore to ensure the authenticity of these tests each process was repeated for 5 times and the mean value was considered as the final concrete response time.



### 6.2.1. Service Repository with 50 services

The first test was conducted when the total numbers of services registered were fifty whereas twenty service types existed in the OWL file. The scenario was considered in order to check the system behavior with different ranges of services stored. Initial point was decided with small number of services to see how does the system behave and to have a clear picture for future behavior of system when more number of services are being added in the system and more tests are being carried out.

#### a) Retrieving All Services

When all services were retrieved the system behaved stable and no major fluctuation in retrieval time was being observed that can be clearly seen in chart 6-1 displayed below. The mean time was found to be 178.125ms.

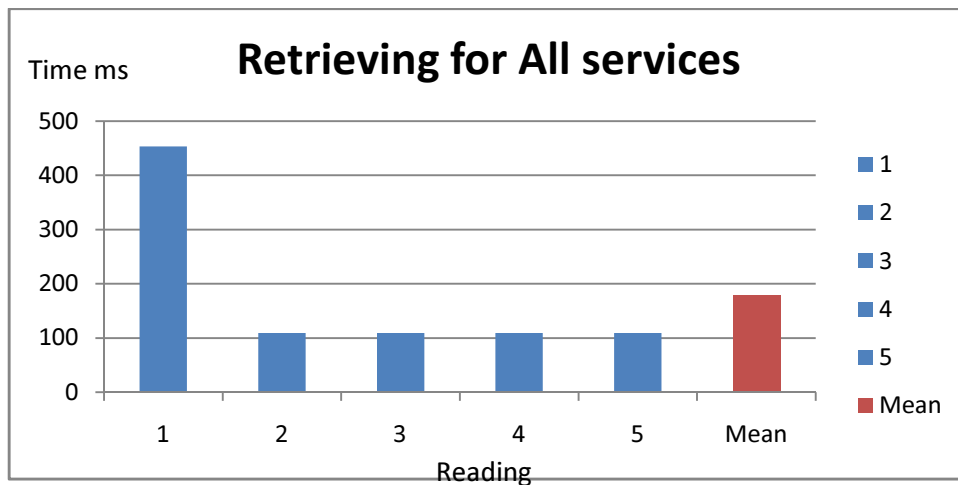


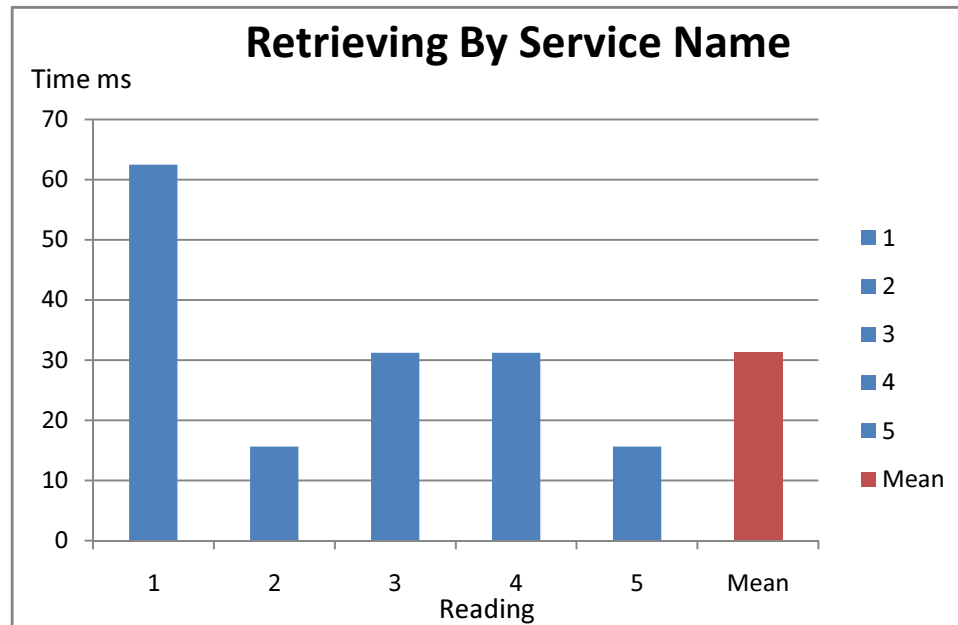
Chart 6-1 Retrieving all services when number of services registered are 50

The readings time for discovering all services are as follows

Reading 1	453.125 ms
Reading 2	109.375 ms
Reading 3	109.375 ms
Reading 4	109.375 ms
Reading 5	109.375 ms
<hr/>	
Mean Reading Value	178.125 ms

**b) Retrieving specific service by service name**

When a specific service which was “Skype” in this case was requested by its service name the system behaved efficiently as expected and consume very little time as only the lookup was being performed in the database. The time reading that were observed are displayed in chart 6-2 below. The mean time was found to be 31.255 ms.



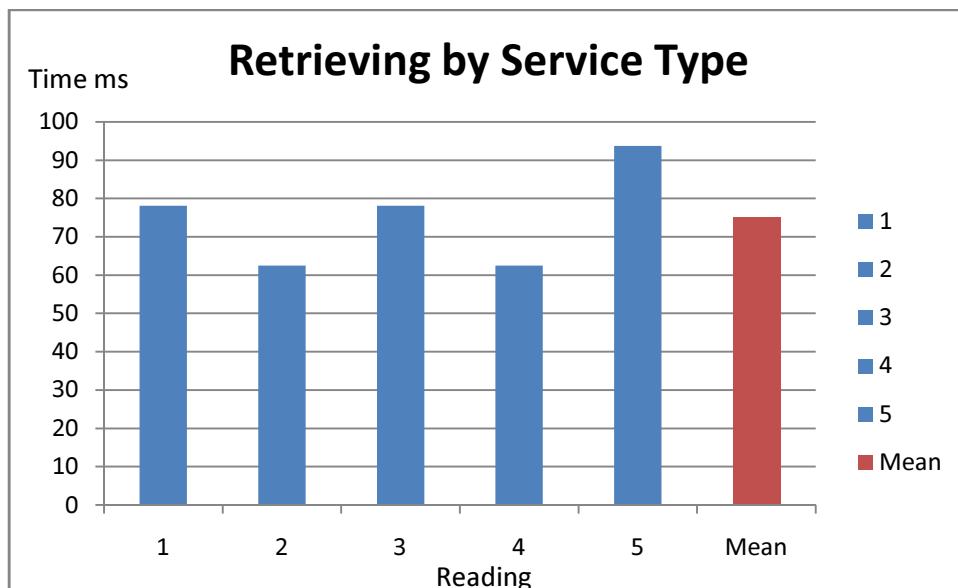
*Chart 6-2 Retrieving services by service name when number of services registered are 50*

The readings time for retrieving a service by its service name is as follows

Reading 1	62.500 ms
Reading 2	15.625 ms
Reading 3	31.255 ms
Reading 4	31.250 ms
Reading 5	15.650 ms
<hr/>	
Mean Reading Value	31.255 ms

c) **Retrieving services by service type**

When the services were tried to be retrieved by the service type lookup the system behaved slower as expected as compared to the results of the service name lookup. The reason for this delay is that the initial look up is being carried out in database and then the equivalent classes are being looked in ontology file which consumed considerably more time. However the system still discovers the services really fast and scale well. The mean time for service type lookup was found to be 75.000 ms. Time consume for retrieving the services can be seen in chart 6-3 below.



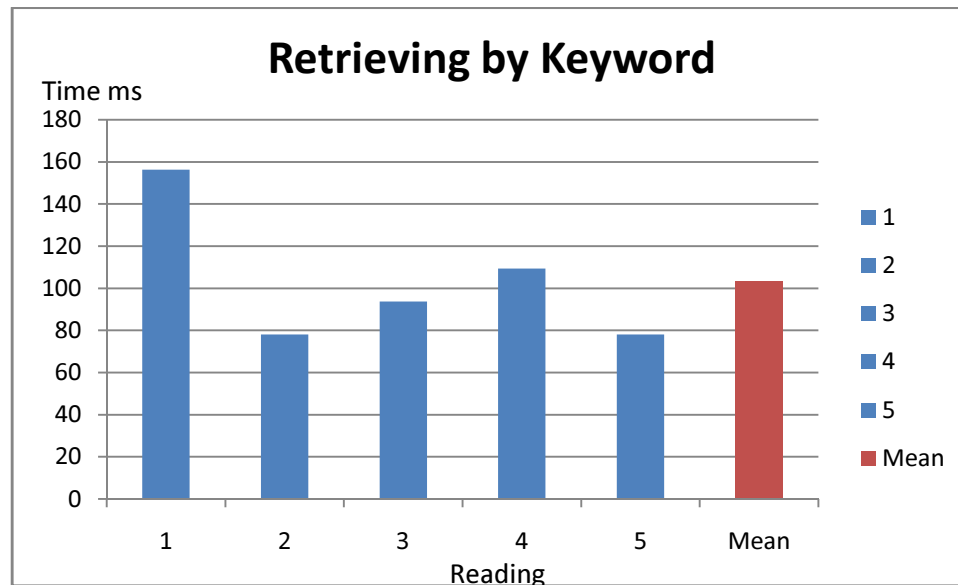
*Chart 6-3 Retrieving services with service type lookup where services registered are 50*

The readings time for retrieving a service by its service type is as follows

Reading 1	78.125 ms
Reading 2	62.550 ms
Reading 3	78.125 ms
Reading 4	62.500 ms
Reading 5	93.750 ms
<hr/>	
Mean Reading Value	75.000 ms

#### d) Retrieving services by Keyword

When the services were tried to be retrieved by keyword lookup the system behaved slower than the preceding both cases as expected. The reason for this delay is that the initial look up is being carried out in database and syntax based matching is being done in the keyword table. Once the services are retrieved a lookup against these values is performed in ontology file. It also depend that how many services were returned as a result of key word search as against each value a lookup in ontology will be carried out. Time consume for retrieving the services when performing keyword search can be seen in chart 6-4 below. The mean time was found to be 103.125.



*Chart 6-4 Retrieving services by keywords when number of services registered are 50*

The readings time for retrieving a service by its service name is as follows

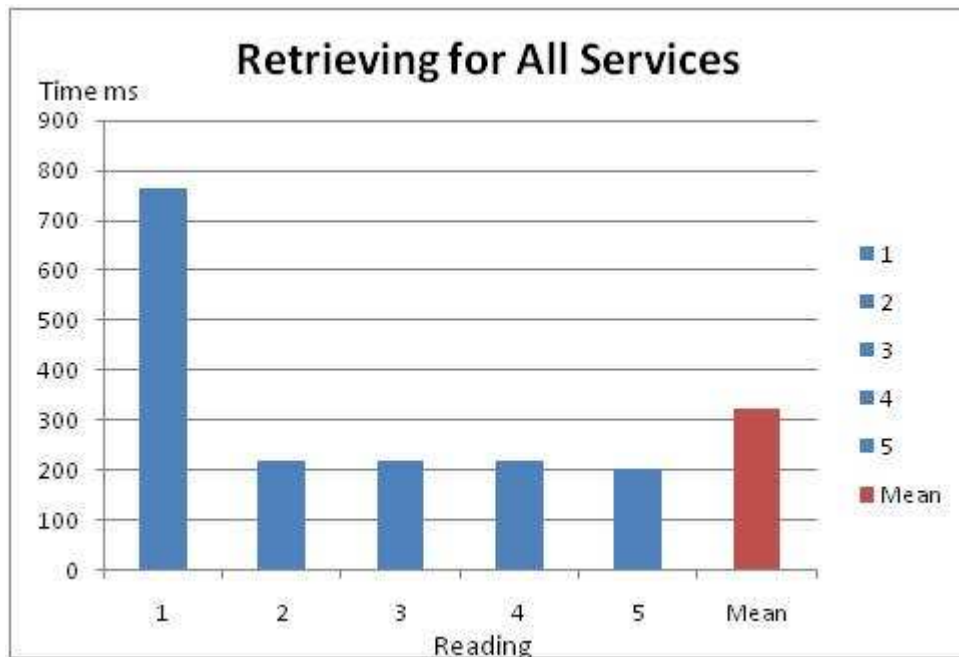
Reading 1	156.25 ms
Reading 2	78.125 ms
Reading 3	93.750 ms
Reading 4	109.37 ms
Reading 5	78.125 ms
<hr/>	
Mean Reading Value	103.125 ms

### 6.2.2. Service Repository with 100 services

The second test was conducted when the total numbers of services registered were hundred and service types stored in OWL file were thirty in number. This scenario was considered in order to check the system behavior by increasing the value of services twice then the first test.

#### a) Retrieving All Services

When all services were retrieved the system behaved slower. As all information specific to each service is stored in different tables at the time of registration and when retrieval is requested for each service a lookup against each service is conducted in all the database tables and it results in a slower time response then all three other scenarios which are discussed later on in this section. A look at Figure 5-3 can explain the database relation i-e ERD diagram moreover the database tables and the sample data can be seen in Appendix A.2. However no major fluctuation in retrieval time was being observed that can be seen in chart 6-5 displayed below. The mean time was found to be 325.00 ms.



*Chart 6-5 Retrieving all services when number of services registered are 100*

The readings time for discovering all services are as follows

Reading 1	765.625 ms
Reading 2	218.750 ms

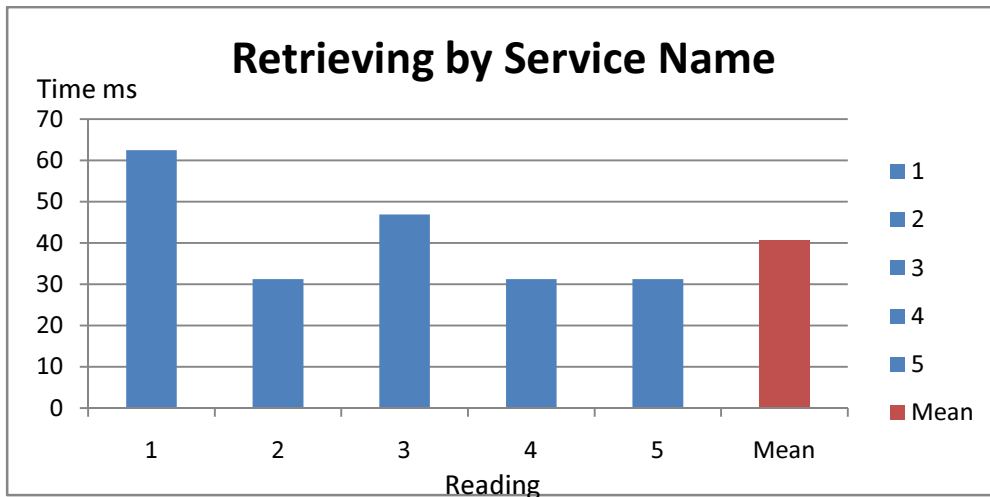
Reading 3	218.750 ms
Reading 4	218.750 ms
Reading 5	203.125 ms

---

Mean Reading Value	325.000 ms
--------------------	------------

**b) Retrieving specific service by service name**

“Skype” service was searched once again to test the system behaviour. The system response time was extremely efficient and a faster response was experienced. The mean time was found to be 40.625 ms and the response time experienced for each reading can be seen in chart 6-6 displayed below



*Chart 6-6 Retrieving services by service name when number of services registered are 100*

The readings time for retrieving a service by its service name is as follows

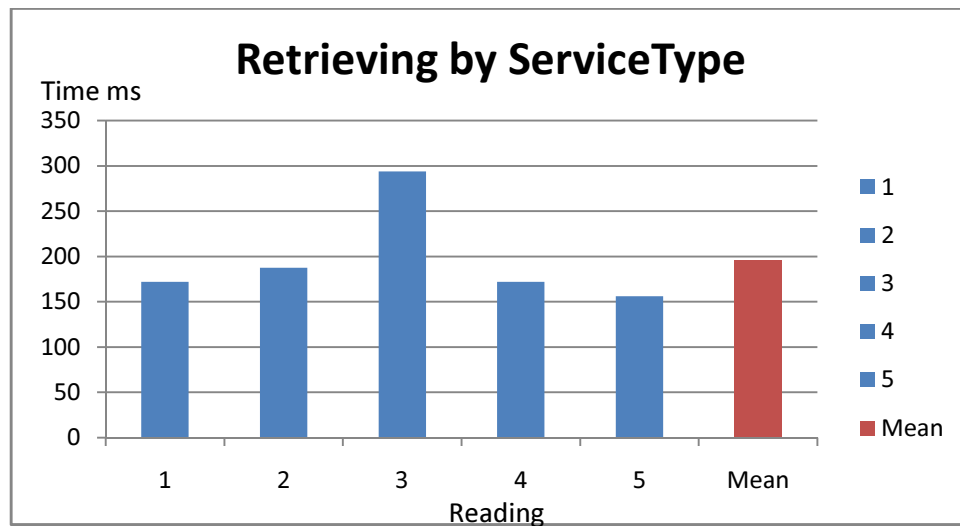
Reading 1	62.50 ms
Reading 2	31.25 ms
Reading 3	46.875 ms
Reading 4	31.25 ms
Reading 5	31.25 ms

---

Mean Reading Value	40.625 ms
--------------------	-----------

c) **Retrieving services by service type**

The service type instances were also increased in our OWL file to test how the system scales to the increased number of service type along with the services. The system was still observed to be efficient and the service search by its service type was performed in a competent way. No major fluctuations were observed in the system behaviour. The mean time for service type lookup was found to be 196.250 ms. Time consumed for retrieving the services can be seen in chart 6-7 below.



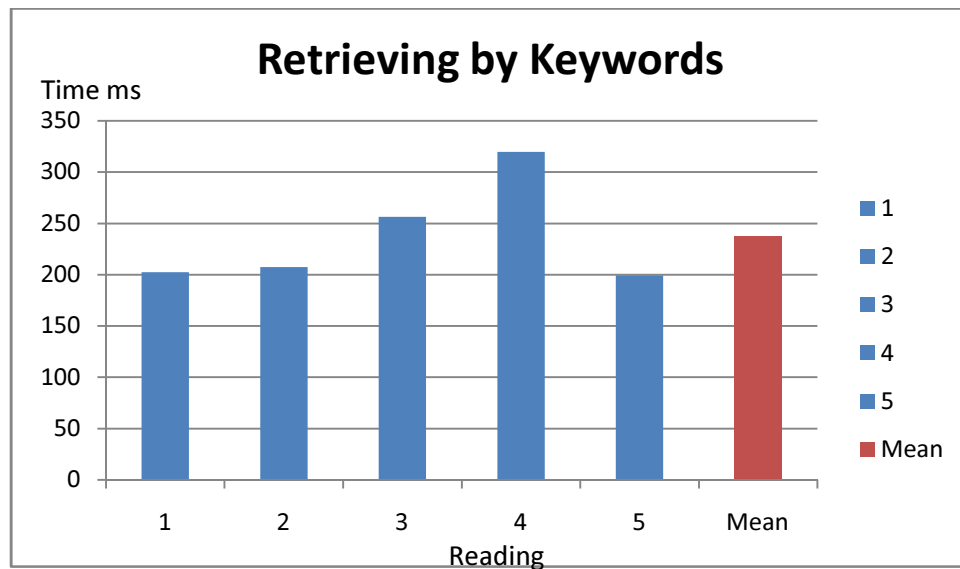
*Chart 6-7 Retrieving services by service Type when number of services registered are 100*

The readings time for retrieving a service by its service type is as follows

Reading 1	171.875 ms
Reading 2	187.500 ms
Reading 3	293.750 ms
Reading 4	171.875 ms
Reading 5	156.250 ms
<hr/>	
Mean Reading Value	196.250 ms

#### d) Retrieving services by Keyword

When the services were tried to be retrieved by keyword lookup the system behaved slower than the preceding both cases as expected. ‘Food’ was used as the keyword to search all the available services relevant to it and 17 services were returned as a result of search criteria. As explained earlier these services were not only the result of the services which were stored in database with following keywords but also which share the equivalent relationship with these values. The system did scale well in this case as well and performed efficiently. Time consume for retrieving the services when performing keyword search can be seen in chart 6-8 below. The mean time was found to be 237.034.



*Chart 6-8 Retrieving services by Keyword lookup when number of services registered are 100*

The readings time for retrieving a service by keywords is as follows

Reading 1	202.35 ms
Reading 2	207.46 ms
Reading 3	256.45 ms
Reading 4	319.65 ms
Reading 5	199.52 ms
<hr/>	
Mean Reading Value	237.034 ms

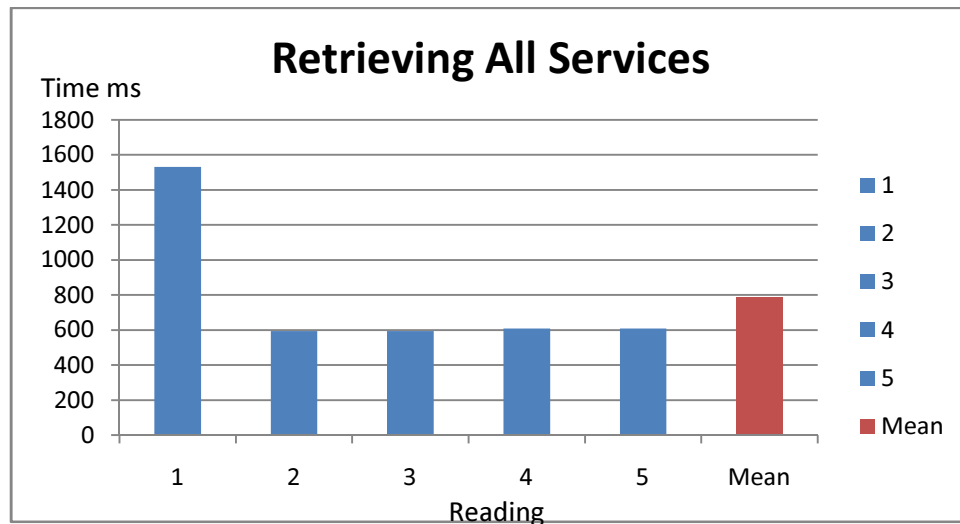


### 6.2.3. Service Repository with 250 services

Total number of services was increased to 250 whereas service type was increased to 50. This scenario was considered in order to develop a concrete view of the system behaviour at regular interval of times.

#### a) Retrieving All Services

As experienced previously the system response was slower while retrieving all services once the number is been increased to 250. The mean time was found to be 787.5.500 ms for the five reading been carried out to retrieve all the services. The reading values are been display below in chart 6-9 below



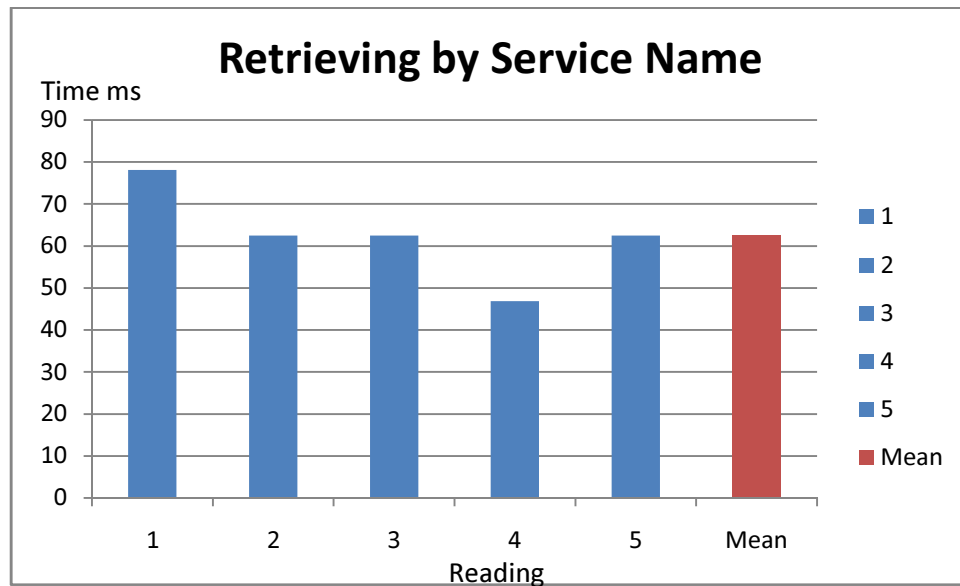
*Chart 6-9 Retrieving all services when number of services registered are 250*

The readings time for discovering all services are as follows

Reading 1	1531.25 ms
Reading 2	593.750 ms
Reading 3	593.750 ms
Reading 4	609.375 ms
Reading 5	609.375 ms
<hr/>	
Mean Reading Value	787.500 ms

**b) Retrieving specific service by service name**

“Skype” service was searched once again to test the system behaviour when the total number of services were increased to 250. The system response time was extremely efficient just like the prior cases. The mean time was found to be 62.50 ms and the response time for each reading can be seen in chart 6-10 displayed below



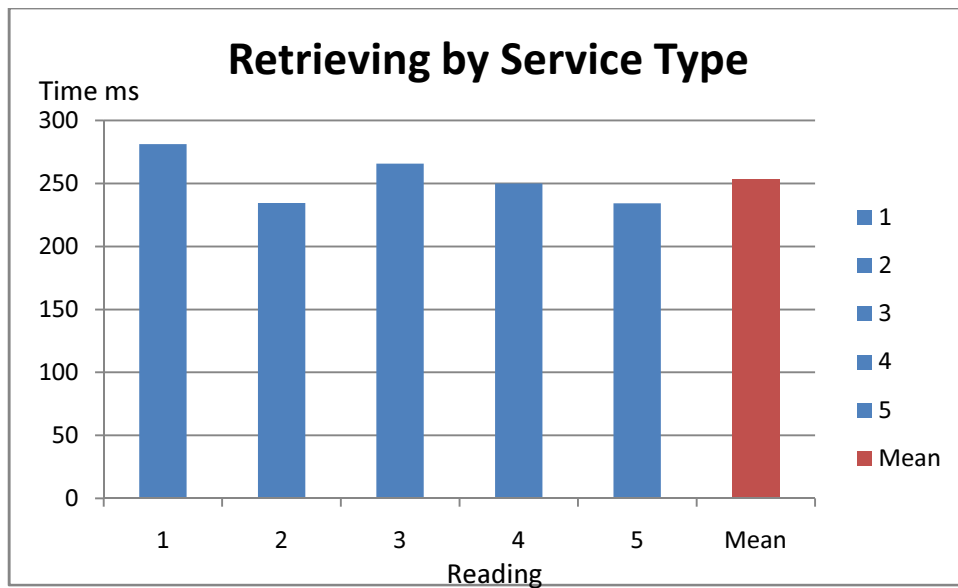
*Chart 6-10 Retrieving services with service name when number of services registered are 250*

The readings time for retrieving a service by its service name is as follows

Reading 1	78.125ms
Reading 2	62.500ms
Reading 3	62.500ms
Reading 4	46.875ms
Reading 5	62.500ms
<hr/>	
Mean Reading Value	62.500ms

### c) Retrieving services by service type

The service types were also increased to fifty along with the value of services to take a closer look at system behaviour. Service type 'Telephony' was searched and 8 services were returned. The services returned was a result of database lookup as well as OWL look up in order to determine the relationship that exist through OWL file i-e parent, child etc. The system is observed to be behaving stable with service lookup search as efficient discovery is experienced. The readings are illustrated below in chart 6-11.



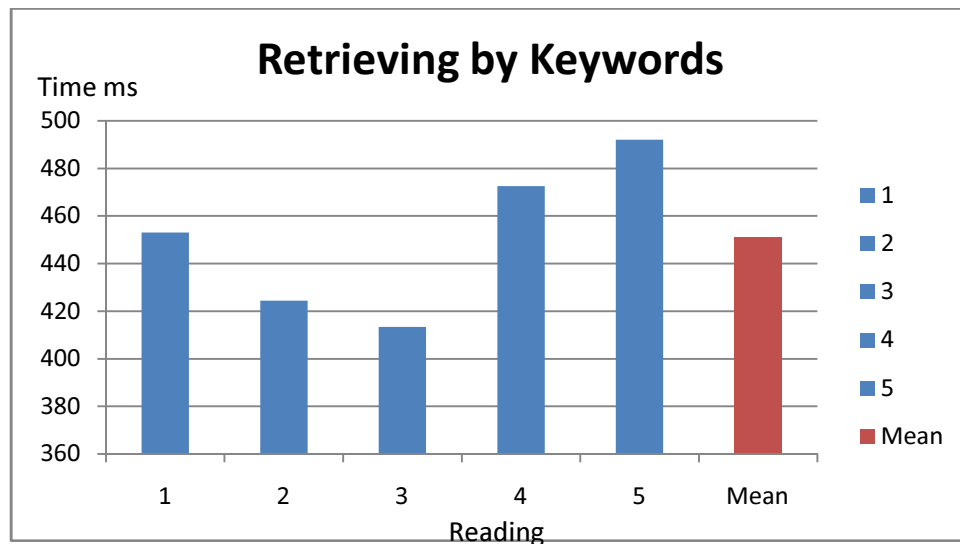
*Chart 6-11 Retrieving services by Service Type with no. of services 250*

The readings time for retrieving a service by its service type is as follows

Reading 1	281.250 ms
Reading 2	234.375 ms
Reading 3	265.625 ms
Reading 4	250.000 ms
Reading 5	234.268 ms
<hr/>	
Mean Reading Value	253.103 ms

#### d) Retrieving services by Keyword

When the services were tried to be retrieved by keyword lookup the system behaved a bit slower as expected. Once again the search was conducted by providing the keyword ‘Food’ and 22 services were retrieved as a result of it. The response was a bit slower compared to service name or service type search but still it was quiet efficient considering the syntax matching that is being conducted in the database and then the lookup that is performed in ontology files. The mean time for key word lookup with 250 services stored in repository was found to be 450.094 ms. all five reading that were carried out are displayed in chart 6-12 below and reading value can also be seen beneath that.



*Chart 6-12 Retrieving services by keyword when number of services registered are 250*

The readings time for retrieving a service by its service name is as follows

Reading 1	453.125 ms
Reading 2	424.376 ms
Reading 3	413.436 ms
Reading 4	472.505 ms
Reading 5	492.015 ms
<hr/>	
Mean Reading Value	451.091 ms

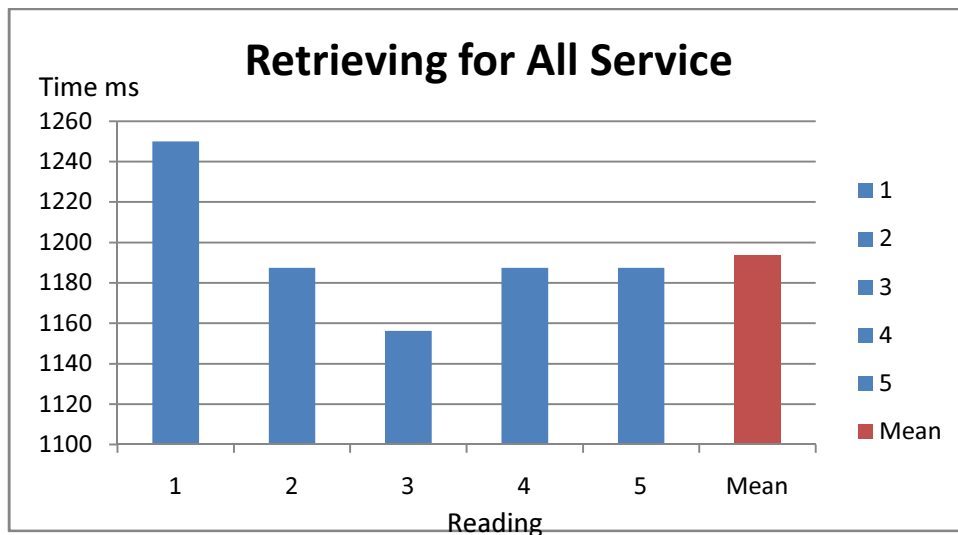
### 6.2.4. Service Repository with 500 services

Fourth interval for testing was decided at 500 services with 75 service types. This was one of an important real time test by adding a lot of services in the system to monitor the system behavior. It will make it easier to understand how the system will behave in real time.

#### a) Retrieving All Services

When all services were retrieved the system behaved slower as estimated, however no major fluctuation in retrieval time was being observed that can be seen in chart 6-13 displayed below.

The mean time was found to be 1193.75 ms.



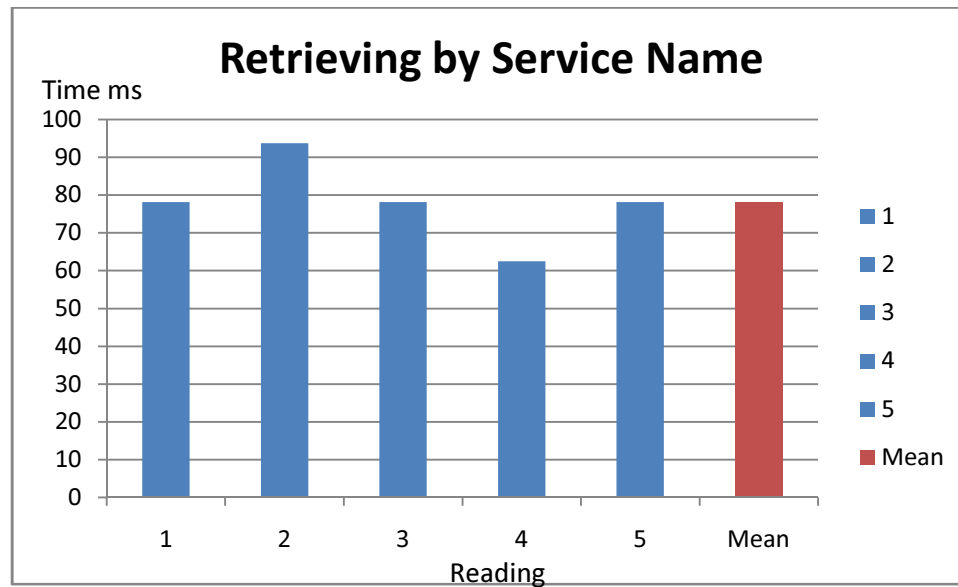
*Chart 6-13 Retrieving all services when number of services registered are 500*

The readings time for discovering all services are as follows

Reading 1	1250.00ms
Reading 2	1187.50ms
Reading 3	1156.25 ms
Reading 4	1187.50ms
Reading 5	1187.50ms
<hr/>	
Mean Reading Value	1193.75 ms

**b) Retrieving specific service by service name**

“Skype” service was searched once again to test the system behaviour when the total number of services were increased to 500. The system response time was extremely efficient as anticipated and been observed in previous cases. The mean time was found to be 78.125 ms. The response time for each reading can be seen in chart 6-14 displayed below



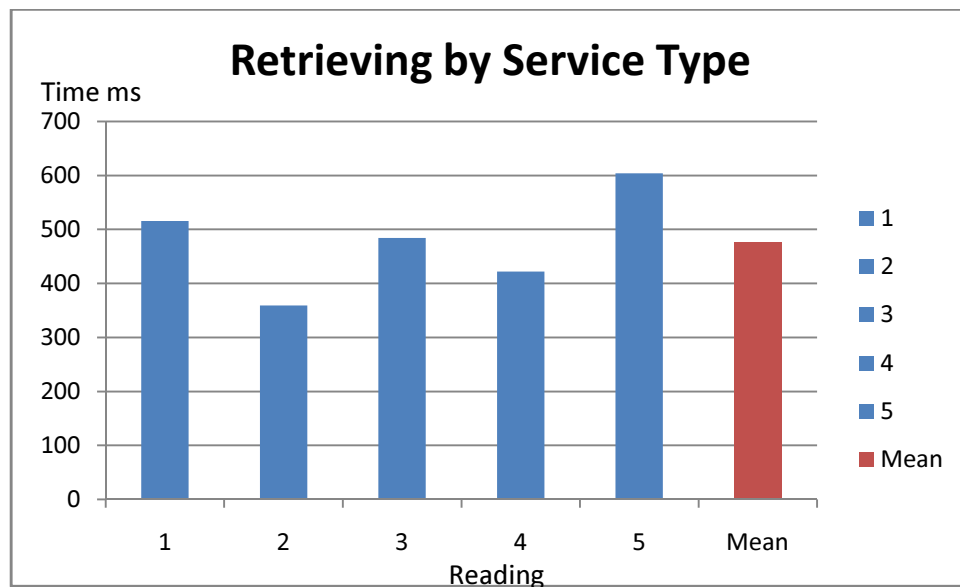
*Chart 6-14 Retrieving services by name when number of services registered are 500*

The readings time for retrieving a service by its service name is as follows

Reading 1	78.125ms
Reading 2	93.750ms
Reading 3	78.125ms
Reading 4	62.500ms
Reading 5	78.125ms
<hr/>	
Mean Reading Value	78.125ms

c) **Retrieving services by service type**

The service types were further increased to seventy five along with the value of services to make the system more real time and tests should be more efficient accordingly. Service type ‘Telephony’ was searched once again and 8 services were returned as no new services were registered in this service type category. The system is observed to be behaving stable with service lookup search as efficient discovery is experienced and no major fluctuation in real time was noted. The readings are illustrated below in chart 6-15.



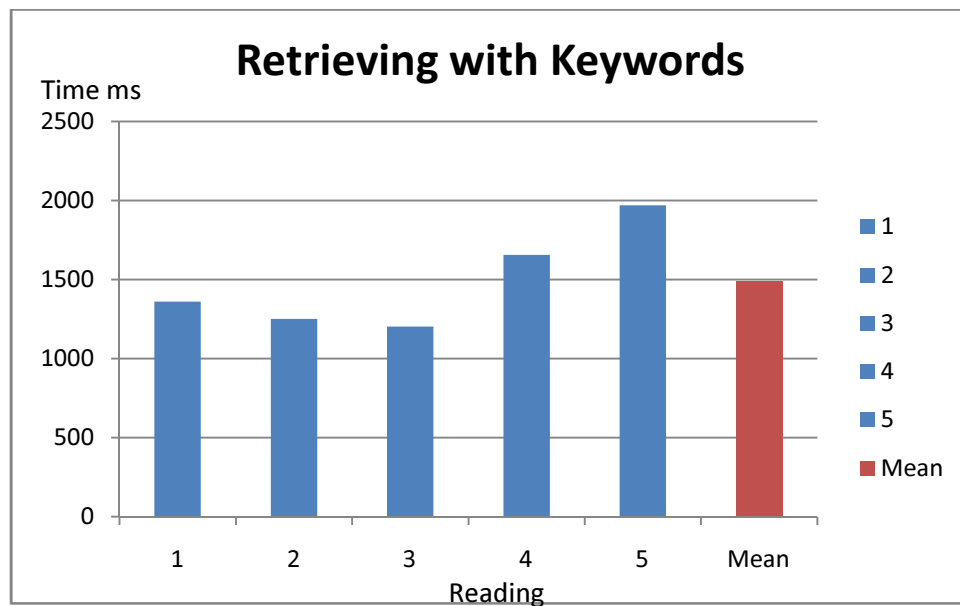
*Chart 6-15 Retrieving services with service type having 500 services*

The readings time for retrieving a service by its service type is as follows

Reading 1	515.625 ms
Reading 2	359.375 ms
Reading 3	484.375 ms
Reading 4	421.875 ms
Reading 5	604.125 ms
<hr/>	
Mean Reading Value	477.075 ms

d) **Retrieving services by Keyword**

The last test for this case was key word lookup in order to retrieve the desired services. Once again 'Food' was searched and the mean time taken for returning the list of matching services was found to be 1487.50 ms. It was slower for the reason explained previously, however it was still efficient considering that 27 services were look up from 500 services based on syntax matching and then performing ONTOLOGY file matching accordingly. No major fluctuation of timing was experienced while conducting this testing. The readings are illustrated in chart 6-16 below



*Chart 6-16 Retrieving services by keyword when number of services registered are 500*

The readings time for retrieving a service by its service name is as follows

Reading 1	1359.375 ms
Reading 2	1250.000 ms
Reading 3	1203.000 ms
Reading 4	1656.255 ms
Reading 5	1968.715 ms
<hr/>	
Mean Reading Value	1487.500 ms

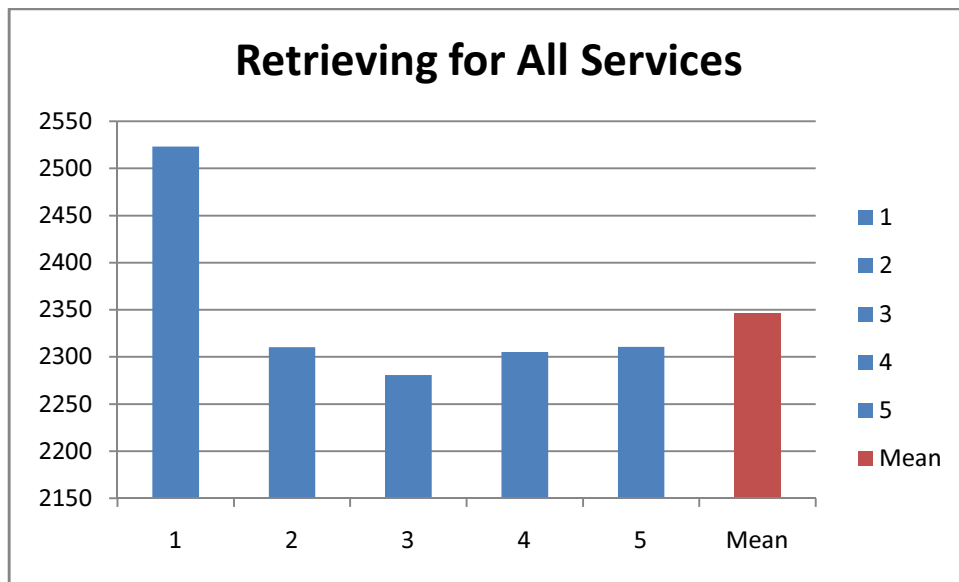


### 6.2.5. Service Repository with 1000 services

Fourth interval for testing was decided at 500 services with 75 service types. This was one of an important real time test by adding a lot of services in the system to monitor the system behaviour. It will make it easier to understand how will the system will behave in real time.

#### a) Retrieving All Services

When all services were retrieved the system no major difference were observed except a slight slower as estimated, however no major fluctuation in retrieval time was being observed that can be seen in chart 6-17 displayed below. The mean time was found to be 2345.6 ms.



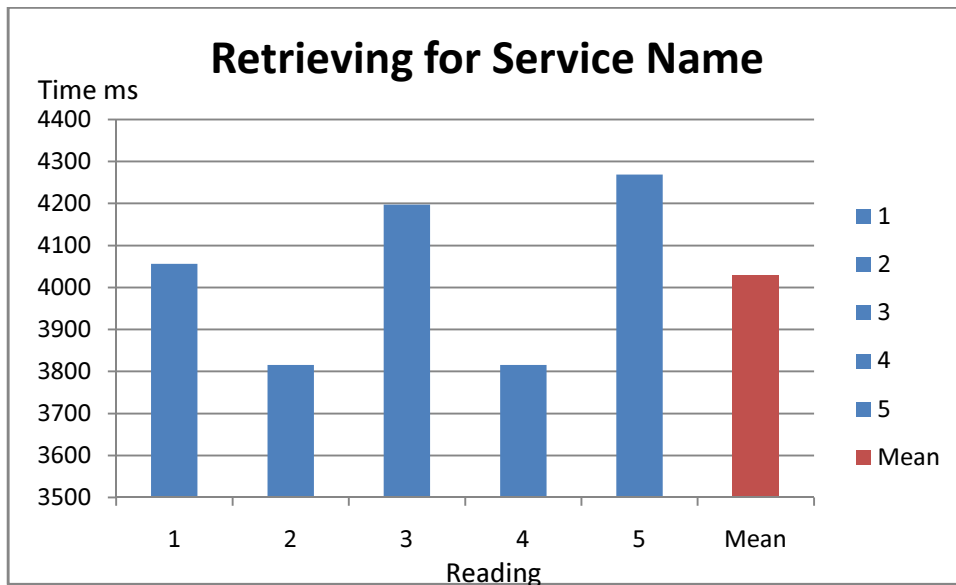
*Chart 6-17 Retrieving all services when number of services registered are 1000*

The readings time for discovering all services are as follows

Reading 1	2523.175 ms
Reading 2	2310.425 ms
Reading 3	2280.605 ms
Reading 4	2305.225 ms
Reading 5	2310.475 ms
<hr/>	
Mean Reading Value	2345.600 ms

**b) Retrieving specific service by service name**

“Skype” service was searched once again to test the system behaviour when the total number of services stored in database are 1000. The system response time was extremely efficient as been observed previously. The mean time was found to be 110.653 ms. The response time for each reading can be seen in chart 6-18 displayed below



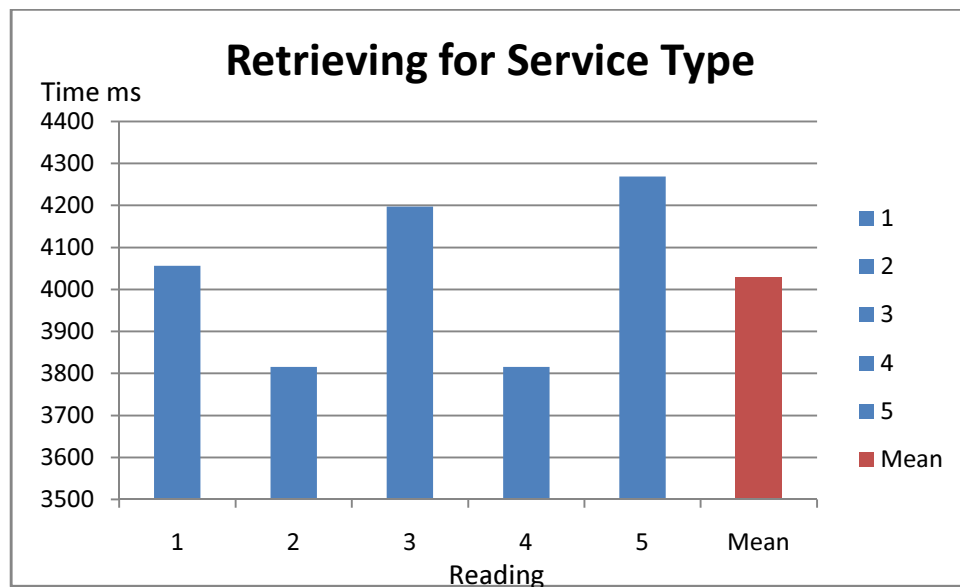
*Chart 6-18 Retrieving services by name when number of services registered are 1000*

The readings time for retrieving a service by its service name is as follows

Reading 1	112.250ms
Reading 2	108.550ms
Reading 3	106.125ms
Reading 4	119.215ms
Reading 5	107.125ms
<hr/>	
Mean Reading Value	110.653ms

c) **Retrieving services by service type**

The service types were further increased to hundred for this final test case along with the value of services to make the system more vulnerable to real time. Service type ‘Telephony’ was searched once again and 17 services were returned as response by the application. New services were registered in this service type category which results in retrieval of more services. The system is observed to be behaving a bit slow because of database and then OWL lookup accordingly. The readings are illustrated below in chart 6-19. The mean time has been found to be 1340.635 ms in this case.



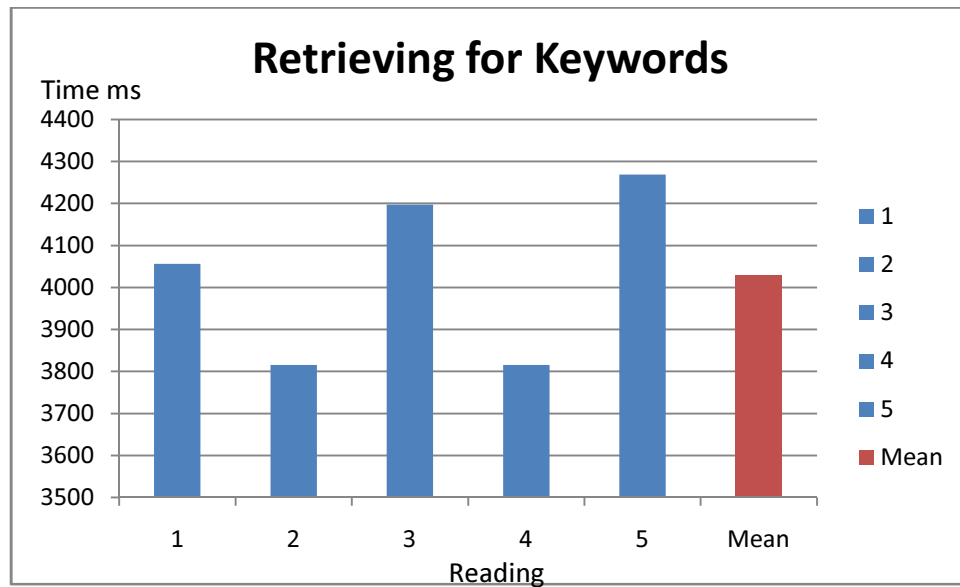
*Chart 6-19 Retrieving services with service type having 1000 services*

The readings time for retrieving a service by its service type is as follows

Reading 1	1378.725 ms
Reading 2	1269.425 ms
Reading 3	1379.450 ms
Reading 4	1321.125 ms
Reading 5	1354.450 ms
<hr/>	
Mean Reading Value	1340.635 ms

d) **Retrieving services by Keyword**

When the services were tried to be retrieved by keyword lookup the system behaved a bit slower as expected. Once again the search was conducted by providing the keyword ‘‘Food’’ and 31 services were retrieved as a result of it. The mean time for key word lookup with 1000 services stored in repository was found to be 4030.492 ms. all five reading that were carried out are displayed in chart 6-20 below and reading value can also be seen beneath that.



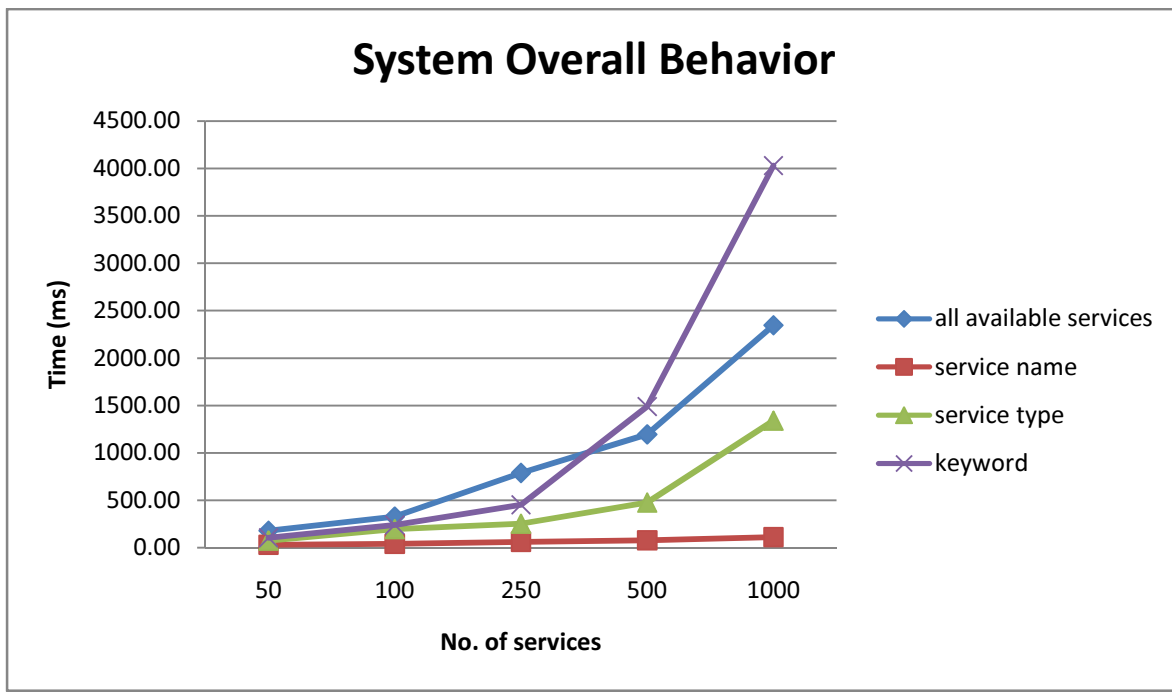
*Chart 6-20 Retrieving services by keywords when services registered are 1000*

The readings time for retrieving a service by its service name is as follows

Reading 1	4055.975ms
Reading 2	3815.325ms
Reading 3	4197.125ms
Reading 4	3815.325 ms
Reading 5	4268.715ms
<hr/>	
Mean Reading Value	4030.492ms

### 6.3 System Overall Behavior

As per the requirements specified previously a service should be discovered really fast. The system is expected to take minimum time in retrieval of service by its service name as only database search will be performed, whereas more time is consumed for ServiceType and Keyword lookup as ontologies are also looked up after database lookup. Chart 6-21 illustrates the overall system behavior for retrieving of services as per user request i-e Service name, service type etc with different number of services registered in system repository.



*Figure 6.21 System Overall Behavior*

The x-axis shows the number of services used for testing while y-axis is the time taken (in milliseconds) to return the results based on different types of parameters used in discovery of services which are:

- 1) retrieving all available services
- 2) retrieving by service name
- 3) retrieving by service type
- 4) retrieving by keywords

The mean reading value for each number of service tested is used to plot the graph hence this result is retrieved. The system behaved exactly as expected and been discussed at the beginning of this section. Service name lookup was really fast and scales well to real time tests that were conducted at different intervals by adding more services in the system repository. However the service lookup and key word look were considerably slower then preceding service name lookup as after retrieving the information from the database, lookup was carried out in ontologies to provide semantic meaning to the search and service discovery, hence making it gradually slower. However the system overall real time performance was found to be satisfactory and it scales well to the increased number of services.

The System performance can be further enhanced by introduction of a powerful server to host this application. Search tends to be slower once more resources are being consumed at the machine so a powerful server will ensure more efficiency. OWLNETAPI can also be edited for scaling well to large number of services and will definitely help a great deal to increase the architecture efficiency. Due to the shortage of time this cannot be conducted in this master thesis but can definitely be an area of interest for future work.

# Chapter 7

## 7 Conclusion

*“There’s two possible outcomes: if the result confirms the hypothesis, then you’ve made a discovery. If the result is contrary to the hypothesis, then you’ve made a discovery.”*

**Enrico Fermi**

This chapter summarizes the works done in this master thesis. Major contributions through this research oriented project along with the result obtained are discussed in detail. We will also discuss the test results that have been conducted in order to measure the system efficiency in real time environment along with the future work that should be focused on. This chapter is organized as follows

7.1) Major Contribution of this thesis

7.2) Summary of thesis

7.3) Future Work

## 7.1 Major Contribution of this thesis

The aim of this thesis has been to identify requirements for future mobile services, design an architecture that fulfill future mobile services requirements and implement the design accordingly. The major contributions in this project are therefore summarized as follows:

- **Requirements** for future mobile services are proposed by foreseeing that how future mobile environment will be shaping in years to come. Based on these requirements current well known service discovery architectures are evaluated and their shortcomings were specifically identified.
- **Design of system for future mobile services** has been proposed. In traditional service discovery architectures no method has been specified to define services as equivalent or partially equivalent to one another, however considering ubiquitous nature of the future mobile environment it will be really important. The system has been designed to allow the services to be introduced by anybody at any time, to be defined as equivalent or partially equivalent to any other service of its type along with multi language support. No formal approval is required for the introduction of a service which makes this design really flexible.
- **Implementation** is successfully carried out for the design proposed. The development was carried out in Windows environment and makes the system usable for any user in mobile or normal environment. This implementation is for real-life usage, and makes it possible to use service discovery in any distributed application by using a simple graphical user interface. The class view, Database tables sample data and source code is presented in Appendix A. System design and its implementation has been submitted for approval to WIMOB conference 2010 in Canada. The paper draft can be found in Appendix B.
- **Testing** was conducted considering use scenarios which are still lacked in current service discovery architectures. The system was evaluated specifically for scalability in the real time environment and was found to be quiet efficient and stable in most of the scenarios.



## **7.2 Summary of thesis**

We successfully created a system that fulfills the requirement of our system allowing anyone to act as a service provider and introduce a service at any time. The services can be introduced in any language by the service provider as well as can define it equivalent to any existing service. The service can be searched by its service type or service name. Moreover the user can also retrieve similar services by requesting any specific service type or all available services registered. The system provides Semantic meaning to the service discovery by using OWL to define the relationships among the services.

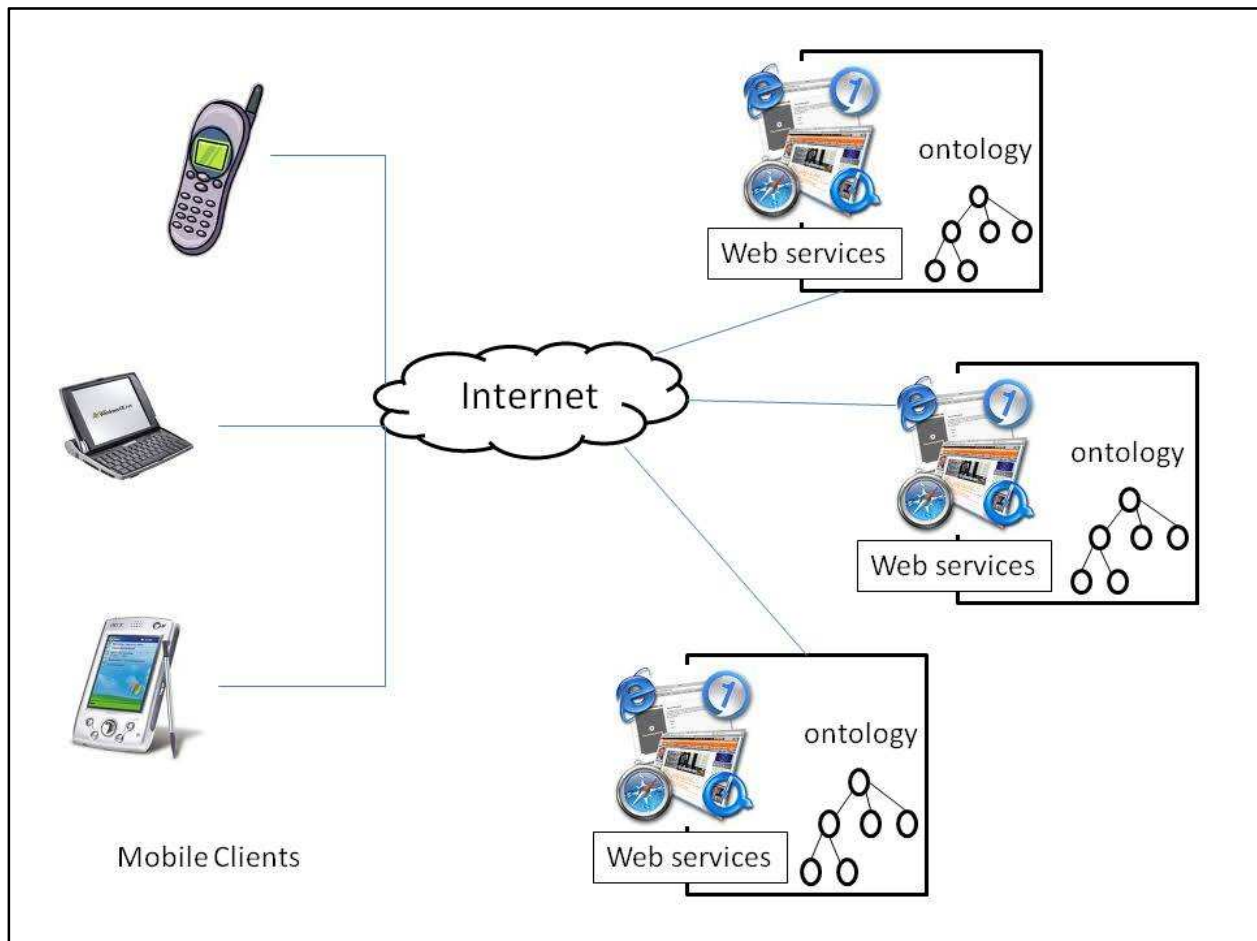
We consider five real time use scenarios in order to check the system functionality that is missing in current architectures. Some services with different names, different services with same names, services in multiple languages, and partial equivalent services were specifically identified and were found to be successfully providing semantics to the service discovery. Anyone can introduce a service at any time and the flexibility proposed was ensured. The system was tested for scalability by adding a lot of services and at different check points. The system behaved quite efficient in service name search and all the desired information was retrieved really fast. The service type lookup and key word look up was found to be a bit slow because the system once searching in database has to look accordingly in OWL to find the equivalent and ensure semantic needed by the service discovery. The system resources were found to be one of an important feature at time of searching so a powerful server is proposed in order to implement it in real time environment.

## **7.3 Future Work**

Due to the shortage of time there are several tasks that could not be completed during the work of this thesis. Further works include:

- i. Carry out larger experiments and test on the future service discovery system with:
  - a. various number of clients
  - b. different bandwidth
  - c. by increasing number of Parents (ParentType)
  - d. complicated service ontology

- ii. To move the future service discovery system to a real mobile and wireless environment (the Client is implemented on real mobile phones). The connection of the mobile could be either via the mobile network or the local WLAN etc. It might also be interesting to carry out experiment with a larger number of mobile phones to challenge the capacity and robustness of the service discovery.
- iii. To have a distributed ontology framework on the future service discovery system as illustrated in *Figure 7-1*.



*Figure 7-1 Distributed ontology framework for future service discovery system*

## Appendix A











### A.1 Class Description

#### A1.1. Base Class:






The Base class provides the main connectivity with the data base. In addition to connectivity it provides the database functions such as (Insert, update and retrieval). This class has a number of overloaded methods to extend the functionality. The Base class is inherited by three different classes to reuse the code.

The Base class exposes the following method and fields.

#### Method

	Name	Description
	ExecuteDataSet(string sqlString)	The method argument takes the sql string and returns dataset.
	ExecuteDataSet(string storeProcedure, params object[] KeyValueParameters)	The method argument takes the stored procedure & key value parameter and returns dataset.
	ExecuteDataTable(string sqlString)	It returns the data table for both methods; the only difference is the argument.
	DataTable ExecuteDataTable(string storeProcedure, params object[] KeyValueParameters)	
	ExecuteNonQuery(string sqlString)	The method is used for (update, delete, insert) & return number of rows affected.
	ExecuteNonQuery(string storedProcedure, params object[] KeyValueParameters)	
	ExecuteReader(string sqlString)	It returns the reader object for both methods; the only difference is the argument.
	ExecuteReader(string storedProcedure, params object[] KeyValueParameters)	
	ExecuteScalar(string sqlString)	It returns a single value.
	ExecuteScalar(string storedProcedure, params object[] KeyValueParameters)	

**Fields**


	Name	Description
	connectionString	A protected field type of SqlConnection.
	sqlAdapter	A protected field type of SqlDataAdapter.
	sqlComm	A protected field type of SqlCommand.
	sqlConn	A protected field type of SqlConnection.
	sqlReader	A protected field type of SqlDataReader.

**A1.2. DALService**









The Data Access layer service (DALService) class inherits the Base class and reuses its basic functionality as connectivity with the database, insertion and retrieval. Additionally it provides the functionality to access the service table in the database. This includes adding of equivalent service id, finding of service, retrieval of all the services on matching and insertion of service.


The DALService exposes the following members.

**Constructor**

	Name	Description
	DALService()	Initializes a new instance of the DALService class.

**Methods**

Access modifier	Name	Description
	AddEquivalentServiceId(int ServiceId, int EquivalentServiceId)	It creates the equivalent relation in the data base.
	FindServiceByName(string ServiceName)	It returns the data table for the service name specified in argument. It's a wild card matching.
	GetAllEquivalent(int ServiceId, ArrayList serviceIds)	It adds all the service ids which are equivalent to specified service id in array.
	GetEquivalentServiceId(int ServiceId)	It return array id for those services which are equivalent with the specified argument.
	GetServiceByName(string ServiceName)	The method returns the data table for exact service name.
	GetServiceByServiceId(int ServiceId)	Returns the complete information in data table for a service id.
	GetServiceByServiceTypeId(int[] ServiceTypeId)	It returns the data table for more than one service type id.
	GetServiceByServiceTypeId(int ServiceTypeId)	It returns the data table for a service type id.


	ServiceTypeId)	
	Insert(string ServiceName, int ServiceTypeId, string ServiceURI, string ServiceDescription)	It returns the service ID generated by the data base.

### A1.3. DALServiceType


The Data Access layer service type (DALServiceType) class inherits the Base class. In addition to the base class functionality it also provides access to the servicetype table in the database. This class not only accesses the servicetype table but it has to ensure the consistency between the database table and the ontology.

DALServiceType exposes following methods and fields.






#### Fields

	Name	Description
	_owlHelper	An object of OWL helper class to work with ontology.

#### Constructor

	Name	Description
	DALServiceType()	Initializes a new instance of the DALServiceTyp class.


#### Methods

	Name	Description
	GetServiceByServiceTypeId(int ServiceTypeId)	It return the service type and the parent type based of service id passed as an argument
	GetServiceTypeId(ServiceType serviceType)	It return the database id of a service type based on the object of servicetype as an argument
	GetServiceTypeIdsHierarchy(int ServiceTypeId)	It returns the parent servicetype id i-e an array of servicetype id for passed ServiceTypeId.
	Insert(string ServiceTypeName, string ServiceTypeURI, int ParentTypeId)	It inserts a servicetype in the database.
	Insert(ServiceType newType)	It insert the servicetype but first it checks for the parent of passed servicetype id. If the parent does not exist then it first add the servicetype id for the parent and then perform the insertion of servicetype.



**A1.4. DALKeyword**

The Data Access layer keyword (DALKeyword) class inherits the Base class for reusability of the code such as the basic connectivity with the database, insertion and retrieval. It provides the additional functionality to access the keyword in the database. Moreover it ensures not to use the same keyword for two services. At time of registering a service it first checks whether the same keyword is already stored in database for any other service. If any value is matched then instead of adding the keyword the keywordID is used to avoid the redundancy. The DALKeyword exposes the following methods.

**Constructor**

	Name	Description
	DALKeyword ()	Initializes a new instance of the DALKeyword class.


**Methods**

	Name	Description
	GetServiceIdByKeyword(string Keyword)	It returns the array of service id associated with the passed argument.
	Insert(int ServiceId, string Keyword)	It inserts the keyword with its associate service id.


**A1.5. Discover Members**







The Discover class contains the overall logic of web service, the methods of discover class are directly exposed to the client to use them. The ASP.net page also consumed this web-service. The Discover class is performing the most prominent functionalities by calling other class methods in the same package, the discover class performs different tasks such as adding new service type, finding service by name, keyword, serviceId and by service type. The discover class also performs the process of service registration. The discover class exposes the following members.

**Constructor**




	Name	Description
	Discover()	Initializes a new instance of the Discover class.

**Method**

	Name	Description
	AddServiceType(ServiceType serviceType)	This method adds a service type in the database and in ontology it creates a class.

	FindService(string ServiceName)	It find the service based on name
	FindServiceByKeyword(string Keyword)	It returns all the service which associate with the keyword.
	GetServiceById(string ServiceId)	It returns the service by service id.
	GetServicesByServiceType(ServiceType srvType)	It returns all services by passing the service type.
	GetServiceTypes()	It returns all available service types.
	RegisterService(Service srv)	It registers the service

**Fields**


	Name	Description
	_owlHelper	An object of OWL helper class to work with ontology.
	_strBaseURI	This field stores the main URI to indicate each class in the ontology.
	_strOwlFile	File location of Ontology.

**A1.6. OwlHelper Members**




The OwlHelper class uses the OWL DOT NET API to perform different tasks in ontology. These tasks contains searching of class from ontology on the basis of service type name, Each of the class in the ontology contains an URI, this URI uniquely identify the class but this URI is a long string include the name of the class, OwlHelper is used to extract the name of class from the URI. Along this the OwlHelper returns all the equivalent classes from the ontology.


The OwlHelper exposes the following members.

**Constructor**



	Name	Description
	OwlHelper()	Initializes a new instance of the OwlHelper class.

**Method**

	Name	Description
	ExtractName(String node)	It extracts the class name from the URI in the ontology.
	GetEquivalentClasses(string ServiceTypeURI)	It returns all the equivalent classes for passed serviceTypeURI from the ontology.
	GetServiceType()	It returns all the classes from the ontology.

	SearchServiceType(string ServiceTypeName)	It returns the matched class for the specified argument.
---	---	--



**Fields**

	Name	Description
	_strBaseURI	This field stores the main URI to indicate each class in the ontology.
	_strOwlFile	File location of Ontology.








**A1.7. Service Members**

The service class is a representation of each individual service which is being used to perform different functions associated with service such as service registration, service retrieval and service matching. Service class includes overloaded constructor, properties: (attributes such as description of service, equivalent services, associated keyword, name of the service, address URI) and different fields.

**Constructor**

	Name	Description
	Service()	Initializes a new instance of the Discover class.
	Service(string Name, string URI, string Description, int[] EquivalentServiceIds, string[] Keywords, ServiceType Type, string StateVariables, string Actions, string Events)	Overloaded constructor

**Properties**

	Name	Description
	Description	String field to store short description about the service
	EquivalentServiceIds	Array of all equivalent service ids
	Keywords	Associated keywords
	ServiceId	Integer ID for Service
	ServiceName	To store the name of service.
	ServiceURI	URI of service <a href="http://WWW.GOOGLE.COM">WWW.GOOGLE.COM</a>
	Type	OBJECT OF SERVICE TYPE





**A1.8. ServiceType Members**





The servicetype class is a representation of each individual servicetype which is being used to perform different functions associated with servicetype such as servicetype registration, servicetype retrieval and servicetype matching. Servicetype class includes overloaded constructor, properties: (attributes such as All equivalent classes in the ontology, Parent (service type) of the servicetype, name of the servicetype in the ontology, associated keyword, unique URI in the ontology and different field.

The Base type exposes the following members.

**Constructor**

	Name	Description
	ServiceType()	Initializes a new instance of the ServiceType class.
	ServiceType(string ServiceTypeName, string ServiceTypeURI, string[] EquivalentClasses, ServiceType ParentServiceType)	Overloaded constructor

**Fields**

	Name	Description
	_parentType	Value of Parent type e.g (Transport is parent type of TAXI).
	_strEquivalentClasses	Value of equivalent e.g(Cab is equivalent of Taxi)
	_strServiceTypeName	Value of the servicetype itself e.g(Taxi)
	_strServiceTypeURI	This field stores the main URI to indicate each class in the ontology.

### A.2 Database table overview

Microsoft SQL server 2005 enterprise edition (46) was used as repository during implementation of this project. The database was used to store all the available service, service type, service URI, keywords for each specific service as well as equivalent for each service or service type if any exist. A brief overview of each table is discussed below whereas the relationships between these tables can be seen in ERD diagram illustrated Figure 5-3. For more detail on each table sample data is illustrated for each table separately in Appendix A.2 which can be referred to if required.

#### A2.1. Service Table

The table service is used to store the data for every registered service. This table typically contains each service id which is generated by the database at the time of registration of service. The second field is ServiceName which is a string value to store the name in Unicode (the Unicode is used to support characters of different languages. For any service there must be parent service which describes the belonging type or the classification of service, this association can be possible by using the servicetypeId as a foreign key from the table of serviceType. Until now there is no field which define the address of the service from where a service can be invoked this is done with ServiceURI to store the URI of a service. To ensure that the user has selected appropriate service there is a field called ServiceDescription to store brief description of the service which could be helpful to select the most appropriate service.

The Sample data for service table

ServiceId	ServiceName	ServiceTypeId	ServiceURI	ServiceDescription
78	برعلا جرب	108	<a href="http://www.jumeirah.com/en/hotels-and-resorts/destinations/dubai/burj-al-arab/">http://www.jumeirah.com/en/hotels-and-resorts/destinations/dubai/burj-al-arab/</a>	burj Al rab on of the best restaurant
79	Cassis Restaurant	155	<a href="http://www.cassis-gourmand.com/">http://www.cassis-gourmand.com/</a>	restaurant in Jakarta
86	Book	161	<a href="http://www.amazon.com/harrypotter">www.amazon.com/harrypotter</a>	Harry Potter Novel story
87	Book	163	<a href="http://www.eazyjet.com/book">www.eazyjet.com/book</a>	online Booking service EazyJet airline

### A2.2. Service Type Table

This table is used to store every service type which is being inserted in the ontology. There are two reasons for doing this.

- i. First it used to ensure the relationships between the service and their service type along with the equivalent services, this is done to reduce the complexity of the system.
- ii. The second reason is to increase the performance of the system. By decreasing lookups in the ontology which take longer time as compare to database lookup.

The service type table is used to maintain the record of the classes which has been inserted in the ontology at the time of registering service type. At the time of registration of service type the database allocate an auto-generated id which is stored in ServiceTypeId field. The service name could also be retrieve from the ontology but due to the performance issue it is better to store in database field instead of ontology, the serviceTypeName is used to store the name of the service type. The servicetype shows a relation between parent and child including the grandparent relation, for grandparent a servicetype could be child of another service type. To achieve the grandparent relation the ServiceTypeParentId is used, this relation is not only used for the grandparent but it also used for the semantic application. There must be some link to identify the service type from ontology this is done by storing the unique URI for each class in the ontology in ServiceTypeURI.

Sample data for service type table

ServiceTypeId	ServiceTypeName	ServiceTypeURI	ServiceTypeParentId
108	restaurant	<a href="http://www.semanticweb.org/ontologies/de-mo1.owl#restaurant">http://www.semanticweb.org/ontologies/de-mo1.owl#restaurant</a>	107
155	restoran	<a href="http://www.semanticweb.org/ontologies/de-mo1.owl#restoran">http://www.semanticweb.org/ontologies/de-mo1.owl#restoran</a>	-1
161	Novel	<a href="http://www.semanticweb.org/ontologies/de-mo1.owl#Novel">http://www.semanticweb.org/ontologies/de-mo1.owl#Novel</a>	160
163	EazyJet	<a href="http://www.semanticweb.org/ontologies/de-mo1.owl#EazyJet">http://www.semanticweb.org/ontologies/de-mo1.owl#EazyJet</a>	162

**A2.3. EquivalentService**

One of the goal of service discovery is to return the services which has partial features. This type of services are stored in the system with the title of equivalent services. To store the equivalent service there must be two services (ServiceId and EquivalentServiceId) to be equivalent of each other and uniquely identify the relation ie EquivalencyID.

Sample data for EquivalentService table

EquivalencyId	ServiceId	EquivalentServiceId
16	91	83

**A2.4. ServiceKeyword**

The keyword table contains the auto generated KeywordId for every keyword. This table is used to store the keyword for the later usage of keyword.

Sample data for service keyword table

ServiceKeywordId	ServiceId	KeywordId
79	78	60
80	78	61
81	78	62
82	79	60
83	79	63

**A2.5. Keyword**

The service keyword table has been designed to avoid redundancy of keywords. For this the table stores the association between the keywords and services. To accomplish this it contains auto generated ServiceKeywordId for every association between the service (ServiceType) and keyword (KeywordId).

Sample data for serviceKeyword table

KeywordId	Keyword
60	Restaurant
61	Dubai
62	Italy
63	Karachi

## **Appendix B. Publication**

The following paper draft has been submitted to WIMOB conference 2010 under category “**Ubiquitous Computing, Services and Applications**”. The conference is scheduled to be held in October whereas the decision on the approval of this paper draft will be announced by end of July. The draft of the paper is attached from next page.

# Service Discovery for mobile multi-domain multi-language environments

Nor Shahniza Kamal Bashah<sup>1</sup>, Atif Bhatti,<sup>2</sup> Imran Aslam Choudhary<sup>3</sup>, Ivar Jørstad<sup>4</sup> & Do van Thanh<sup>5</sup>

<sup>1</sup>Department of Telematics, NTNU, O.S. Bragstads Plass 2E, NO-7491 Trondheim, Norway- nor@item.ntnu.no

<sup>2</sup>Department of Telematics, NTNU, O.S. Bragstads Plass 2E, NO-7491 Trondheim, Norway- bhattiatif@gmail.com

<sup>3</sup>Department of Telematics, NTNU, O.S. Bragstads Plass 2E, NO-7491 Trondheim, Norway- iachouhdary@gmail.com

<sup>4</sup>Ubisafe, Gamleveien 252, 2624 Lillehammer – Norway

<sup>5</sup>Telenor & NTNU, Snaroyveien 30, 1331 Fomebu, Norway

*Abstract*—In mobile multi-domain multi-language environments, a service can be anything and introduced by anybody. Consequently, same or equivalent services may have different names and services with same name or type may be completely different. Existing service discovery systems are incapable of handling these situations. We propose a service discovery, which is able to discover all these new service types. In addition, it is capable to find services that are not exact matches of the requested ones. More semantics are introduced through attributes like `EquivalenceClass`, `ParentType` and `Keywords`.

*Keywords*—service discovery; service lookup; service advertisement; service request; service matching; semantic matching

## I. INTRODUCTION

The popularity of the mobile phones reflects the user's appreciation of the freedom, i.e. having access to any service anytime anywhere. However, so far the mobile services are mostly limited to voice communication and Short Message Service (SMS). Advances in wireless technologies have allowed mobile phones to be connected simultaneously to several network systems, e.g. GSM, GPRS, UMTS, WLAN, WiMAX, LTE, etc. and paved the way for innovative mobile services. But the success criterion is this existence of a sound, flexible and efficient service discovery which enables the clients to find and use the services. The future service discovery must be capable of finding relevant services offered by heterogeneous network systems. In such a ubiquitous communication environment, similar services can have different names in different languages. Furthermore, services with same name may not offer the same functions and capabilities. A future service discovery must be capable of dealing with the described challenges without confusion and returning correct answers in acceptable amount of time. In addition interoperability with existing service discovery systems must be ensured.

The goal of this paper is to present a service discovery system for mobile multi-domain multi-language environments. The paper starts with a brief review of the state-of-the art service discovery systems and their limitations. Next, is the

specification of the requirements of the future service discovery system. The main part of the paper provides a comprehensive description of the system, which includes a service naming and description convention and an overall architecture. The sequence diagrams and use scenarios are given to illustrate the effectiveness of the proposed service discovery. Future works are summarized in the conclusion.

## II. STATE OF THE ART SERVICE DISCOVERY

In this section the state of the art service discovery systems are discussed.

### A. Jini

In Jini, the service is classified by the types of the service object also known as service item. It contains three fields which are the `ServiceID` (globally unique 128-bit value generated by the Lookup service), `Service` (a reference to the object implementing the service) and `AttributeSets` (a set of tuples describing the service) [1]. The Client can request a service by using the `ServiceID` (if the Client knows exactly which service it wants). However, the `ServiceID` might sometimes be long and difficult to be remembered. Therefore a Lookup service is necessary. A Client locates an appropriate service by its type – that is, by its interface written in the Java programming language – along with descriptive attributes that are used in a user interface for the Lookup service. It is worth noting that two different service instances or two service types are not allowed to have same name. If the first level checking (first filter) does not give any result or it does give multiple results, the Lookup service will then check on the `AttributeSets` for the second level matching and to reduce the number of matching services. There can be multiple instances of the same class with different attribute values, as well as multiple instances of different classes in the service item contained in the service Lookup directory.

The Jini system uses the Java classes for a syntactic ontology (e.g. using printer class for a printer) and includes a Lookup Attribute system to introduce more semantics. A wide variety of hierarchical views is obtained by aggregating items

according to service type and attributes. The main limitation that makes Jini inappropriate for future mobile environment is the strict naming convention, which requires the uniqueness of the service name and type.

B. UPnP

The classification for UPnP is based on the device type which consists of a UUID (Universal Unique Identifier) and a URL (Uniform Resource Locator) of the description of the device [2]. This information is retrieved by the Control Point during the discovery. There are very few details about the device and a more detailed description of the device is then retrieved. The device description is divided into two logical parts: a device description describing the physical and logical containers and service description describing the capabilities exposed by the device. A single physical device may contain multiple logical devices which can be a single root device with embedded devices (and services) or multiple root devices (with no embedded devices). The UPnP Device Architecture defines a schema or template for creating device and service descriptions for any device or service type. Individual working committees subsequently standardize on various device and service types and create a template for each individual device or service type. Finally, a vendor fills in this template with information specific to the device or service, such as the device name, model number, manufacturer name and URL of the service description.

The service description in UPnP is at syntactic level hence the service matching is limited to syntactic comparison based on attributes or interfaces. It will not be able to detect a partially match such as the case where the service descriptions involve different representations of conceptually equivalent content. Furthermore, the device type must be predefined resulting to an inflexibility of naming and classifying of devices and services.

III. FUTURE SERVICE DISCOVERY REQUIREMENTS

The requirements for future service discovery have been derived and highlighted in [3]. This section will specifically discuss thoroughly the requirements of future service discovery which are not fulfilled by the existing service discovery systems like Jini and UPnP.

The emergence of network technologies has allowed mobile user to have access in more than one network at a time. This results to the introduction of many services in different type of network systems. It is hence possible to conclude that:

*“A service in the future multi-domain environment can be anything and be introduced by anybody at anytime.”*

This leads to a large variety of names and would result to the following situations:

- 1) *The same service may have different names and in different languages.*

- **Requirement:** The future service discovery must be capable of handling the same service with different names in multi-languages.
- 2) *The same service name or word has several meanings and denotes different services*
  - **Requirement:** The future service discovery must be capable of handling different services with same name without confusion.
- 3) *The service found may not be an exact match of the requested service but is a superset i.e. have extra functions in addition to the requested ones.*
  - **Requirement:** The future service discovery must be capable of discovering the services which are partially matches of the requested one.

IV. FUTURE SERVICE DISCOVERY SYSTEM

In this section the overall architecture of the proposed future service discovery and its development environment is described. The call flow on each method used is also explained in the sequence diagrams.

A. Service naming and description convention

According to the requirements described in the previous section, the service name and service type should not be restricted to any length or format. The service provider must have the freedom to the name and type as pleased. This requirement poses problem for the service matching to succeed.

To solve the problem we propose to introduce more semantics in the service description as Figure 1. In addition to regular parameters like Name, Type, State Variable and Actions which provide the syntaxes of the service additional parameters like Keywords, ParentType, EquivalenceClass are introduced to provide more semantics to the service. They will be used in the service matching to find the requested services in an efficient and unambiguous manner. Their usage will be elucidated by the use scenarios in later section.

**Service type description template**  
 A service type description has the following items:

- **Name:** The name can be in any language and less than 64 characters
- **Keywords:** Some words that can be used in the first round discovery
- **ParentType:** The name of service type that the current type is derived from
- **ParentType URL:** The URL of the parent type
- **EquivalenceClass:** All aliases in any language are given here
- **StateVariables:** The state variables determine the state of the services. They are left empty in the service type description
- **Action:** Actions are the methods that can be called by clients or other services
  - Each action has a name and a set of parameters
    - Each parameter has a type, allowable values (for enumerated types), and direction (in or out)
- **Event:** Enable clients to subscribe to the occurrence of a particular event

Figure 1. Service description template

B. Overview of the system architecture

Figure 2 illustrates the system architecture for future service discovery.

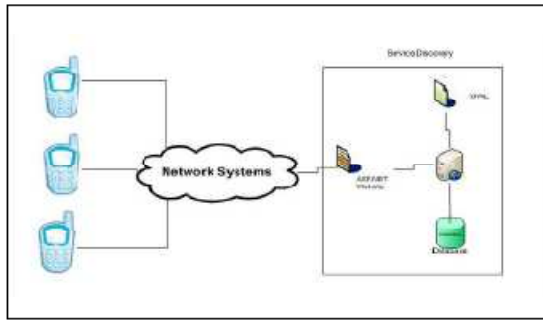


Figure 2. Future service discovery system architecture

The major components of the architecture are:

- 1) *Web Service Client* – This component is an application running on the remote mobile terminal. The Client on behalf of the user can ask/search for a service. It can also on behalf of the Service Provider introduce and register a new service. The protocol used between the Client and the actual XML Web Service is SOAP (Simple Object Access Protocol) [4].
- 2) Components of the service discovery system consisting of:
  - i) *ASP .Net [5] Website (XML Web Service)* – This service acts as an interface between the Client/Service Provider and the System Server. The IIS Internet Information Services 5.01 or later is customized in order to publish the ASP.Net pages on the Internet.
  - ii) *Database* – This component acts as a service repository which stores the services registered by the service provider.
  - iii) *OWL [6] file* – is used for semantic matching with the OWL Lite (EquivalenceClass and ParentType).

#### B. Service discovery tested

The hardware as well as the software components used for the system implementation are described below:

- 1) *Hardware* - A computer with installed Windows XP is used as a server. The server can process several request from multiple clients at the parallel. The Client can request the server by using any platform or browser to retrieve the desired services.
- 2) *Software*
  - a) *Server side:* Since the application is developed using .Net technology, the .Net framework version 2.0 is installed on the server. Microsoft Windows based operating system is the minimum requirement for this

version of .Net framework. Therefore Microsoft Windows XP was installed on the server.

b) *Client:* The Client is independent of any architecture and hence can access the services by using any industry standard Internet browser from any operating system that has access to the Internet.

c) *Programming languages:* For the implementation of this project Microsoft Visual Studio 2005 IDE for C# was used. The application was developed in C# whereas the website was developed using ASP.Net technology.

- *C#:* C# was mainly used as an implementation language because of its convenient programming capabilities, the object oriented paradigm it supports, type-safety and wide range of libraries available as a part of .Net framework [7]. Another reason to use C# was the OwlDotNetApi that was mainly used to access OWL file in order to define parent as well as equivalent classes for service types. The detail of this API can be found on [8].

- *ASP.Net:* ASP.Net is used for the XML web service and the web form interface for accessing the web service. Reason for using ASP.Net:

- o for building XML web service because it makes exposing and calling web services very simple.

- o for Client website because of the flexibility of interface it provides and support for the mobile devices. By building a website in ASP.Net one has to write the code once and the ASP.Net automatically generates pages based on the device they are called.

d) *Support for ontologies.* As the semantic meaning in this project of service discovery is being achieved by ontologies they were continuously monitored manually by using Protégé 4.0.2 [9]. OWL Lite is chosen in the implementation of this project due to its simplicity and the dynamic nature of the project even though it is known that it has some limitations as to compare with the OWL DL and OWL Full. Since the services can be introduced by anybody and there is no control mechanism on introducing the services, therefore it is required to avoid complexity in the system usage and for that only the EquivalenceClass and the ParentType properties is used which can be accomplished by the OWL Lite.

#### C. Sequence Diagram

The call sequence diagrams [10] are used to explain the flow of the four main methods in the future service discovery.

- 1) *Registration a service.* This method is used for the Service Provider to register a new service.



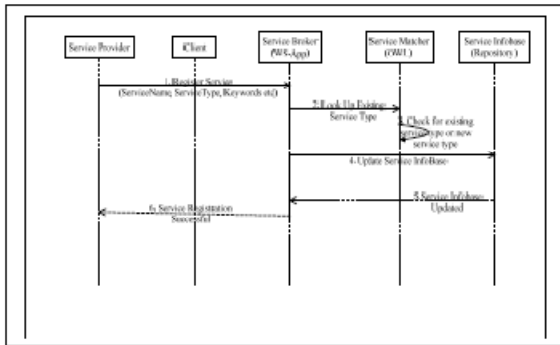


Figure 3. Call sequence diagram for Registration a service

- i) When a Service Provider wants to register a service, he/she will first type the particular details of the service (e.g. the Service Name, Service Type and Keyword).
- ii) The Service Broker which is the Web Service Application will then lookup for Equivalence Service Type.
- iii) The Service Matcher which in this case is the OWL Lite will check for existing Service Type or new Service Type.
- iv) The Service Broker will then update the Service Infobase which is the Service Repository of this system.
- v) After that the Service Infobase is updated.
- vi) The service registration is successful and Service Provider can see the new service registered in the service list.

2) *Discovering services*: This method is used for the Client which wants to find all available services.

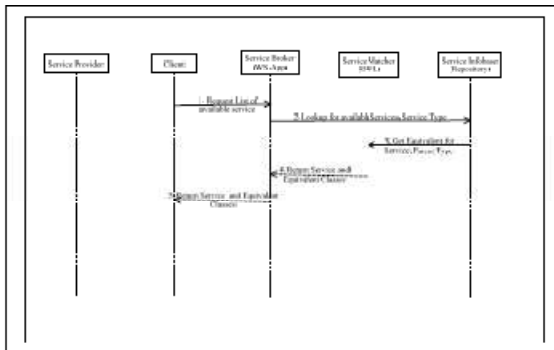


Figure 4. Call sequence diagram for Discovering services

- i) When the Client wants to discover available services, it may request for list of services to the

- ii) Service Broker.
- iii) The Service Broker then will lookup for available services from the Service Infobase.
- iv) The Service Infobase will then get the Equivalent for Service and Parent Type from the Service Matcher (OWL).
- v) The Service Matcher will return the available services and all Equivalent Classes to the Service Broker.
- vi) The Service Broker will return the services available and its Equivalent Classes to the Client.

3) *Service Request*: This method is used if the Client wants to find a particular service or set of services

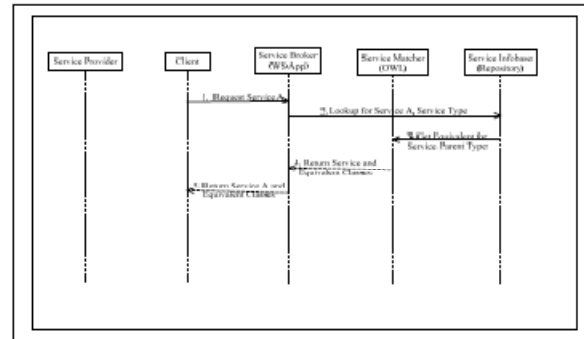


Figure 5. Call sequence diagram for Service Request

- i) When the Client wants to request for a particular service, he may find the service either by using the Service Name or the Service Type.
- ii) The Service Broker will Lookup for a particular service A and its Service Type in the Service Infobase.
- iii) The Service Infobase will then get the Equivalent for Service A and its Parent Type from the Service Matcher (OWL).
- iv) The Service Matcher will return the requested Service A and its Equivalent Classes to the Service Broker.
- v) The Service Broker will return the Service A and its Equivalent Classes to the Client.

4) *Find a service*: This method is used if the Client wants to find a service but knowing very little information about the service (for e.g. just having a Keyword).

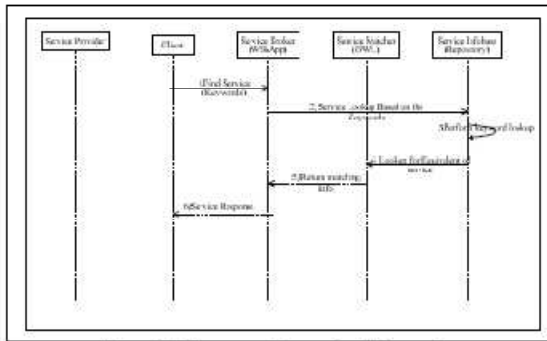


Figure 6. Call sequence diagram for Find a service

- i) The Client can also find a service via the Keyword if he is not sure about the service Name
- ii) The Service Broker will do the Service Lookup based on the Keyword and send it to the Service Infobase.
- iii) The Service Infobase will then perform the Keyword lookup.
- iv) The Service Matcher (OWL) will then Lookup for Equivalent services.
- v) Service Matcher will return the matching information to the Service Broker.
- vi) The Service Broker will finally send the service response to the Client.

V. USE SCENARIOS

Based on the requirements specified in Section III, three use scenarios are selected for described in details.

Use scenario 1

The network system has similar services with different names and different languages.

When a Client wants to find a service, such as for example - a *Restaurant* service he will use either a standard English or his native language. The *Restaurant* service can be registered under other different names in multiple languages for e.g.

Restaurant: {Restoran, Ristorante, Café, Bistro, Warung, ... }

If the Client enters the word *Café*, the future service discovery will return not only the *Café* service but also other similar service with *Café* in other names as depicted in Figure 7.



Figure 7. Use scenario 1a) - having similar service with different names

In addition, the future service discovery also supports for finding a service in multiple-languages as shown in Figure 8.



Figure 8. Use scenario 1b) having similar service in multiple languages

By having this feature the Client will not have to worry even though other names returned when he wants to find a service (especially if he is abroad and different languages is used to call a service) because the service discovery system returns only services equivalent to the one requested.

There are three options for the Client when it wants to find a service:

- i) *by Service Name* – if the Client knows exactly the service it wants to look for
- ii) *by Service Type* – if the Client is not sure about the Service Name but only knows the Service Type
- iii) *by Keyword* – if the Client only knows very little about the service and has some clues describing the service (for e.g Dining or Meal which refer to Restaurant)

Use scenario 2

The network system has different services with the same name

Since in the future ubiquitous communication systems the service can be anything and introduced by anybody, there can be a possibility of having different term and perspectives of knowledge of the service from the user and service provider. The future service discovery supports the function of having different services with the same name as depicted in Figure 9.



Figure 9. Use scenario 2 – having different services with the same name

As in the above example, the word Book can refer to a type of Book service (for e.g. buying online book or information about a book). There can also be other meaning of Book service which may refer to a Reservation service.

Even though the future service discovery allows having different services with the same name, the ambiguity and confusion can still be avoided via the details of the Service Description and Service Type returned during the service search.

*Use scenario 3*

The service discovery returns partially matches to requested service.

The future service discovery is using a semantic matching in addition to syntactical matching as the one used in the existing service discovery. This is very important especially in the situation where no equivalent service available but there exists service, which have additional functions to the requested ones. Such a service is called – partially match.

The future service discovery introduces the use of service subtyping by having ParentType and attributes ParentType in the Service Type description template. As illustrated in Figure 10, when the Client is asking for Telephony service, Skype, SIP and G1 are returned because they are grand children of Telephony (Skype and SIP are subtype of IP Telephony while G1 is subtype of GSM) and have inherited all the characteristics of Telephony. In this case the Skype, SIP and G1 are having similar functions (voice call – which is the generic Telephony features) but some of them have more or different additional functions (for e.g. Skype has video call feature) and they are also different in service components and service implementations.



Figure 10. Use scenario 3a) – having partially match service

However, if the Client is asking for a specific Skype service, only the Skype is returned and nothing else. This is as illustrated in Figure 11.



Figure 11. Use scenario 3b) – service subtype

VI. CONCLUSION

In this paper a service discovery for mobile multi-domain and multi-language environments is proposed. To allow any service provider to introduce service as anything at anytime and anywhere more semantics are introduced. In addition to the service name and type, the relationship Keywords, ParentType and EquivalenceClass are introduced to concretize the service classification. A prototype is successfully implemented and is working fine. The next step will be to carry out larger experiments and tests on the system, e.g. increase the number of parents, the number of equivalent services, etc. It is also quite interesting to move the system to a real mobile and wireless environment where the client is implemented on real mobile phones.

REFERENCES

- [1] Sun Microsystems. Jini™ Technology Core Platform Specification Version 2.0. [Internet] June 2003. <http://www.scribd.com/doc/2280880/Jini-Technology-Core-Platform-Specification>.
- [2] Microsoft Corporation. Universal Plug and Play Device Architecture Version 1.0 [Internet] 15 October 2008. <http://www.upnp.org/specs/arch/upnp-arch-devicearchitecture-v1.0.pdf>.
- [3] Kamal Basha, N.S, Jørstad, I & van Do, T: Service Discovery in Future Open Mobile Environments. Proc. the Fourth International

- Conference on Digital Society (ICDS2013) – February 10-16 2010. St. Maarten, Netherlands Antilles. ISBN: 9780769519539.
- [4] World Wide Web Consortium (W3C). SOAP Version 1.2 Part 1 – Messaging Framework (Second Edition). [Internet] 27 April 2007. <http://www.w3.org/TR/soap12-part1/>.
- [5] Onion, Fritz. "Essential ASP .Net with Examples in C#". Microsoft .Net Development Series. Boston, MA: Addison Wesley, 2004. ISBN 0201760401.
- [6] World Wide Web Consortium (W3C). OWL Web Ontology Language Overview. [Internet] 10 February 2004. <http://www.w3.org/TR/2004/REC-cwl-features-20040210/#s3>.
- [7] Blum, Richard. "C#™ Network Programming". Alameda, CA: John Wiley, 2003. ISBN. 0782141765.
- [8] OwlDotNetApi [Internet] <http://users.skynet.be/bpellens/OwlDotNetApi/documentation.html>.
- [9] Holger Knublauch, Ray W. Fergerson, Natalya F. Noy and Mark A. Musen. "The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications". The Semantic Web (ISWC 2004), 2004, pp. 229-243.
- [10] Fowler, Martin. "UML Distilled: A Brief Guide to the Standard Object Modeling Language". Third Edition. Boston, MA: Addison Wesley, 2004. ISBN 0321193687.

## **Bibliography**

1. **MarkWeiser.** *The Computer for the 21st Century.* s.l. : Scientific American, 1991.
2. **Patric Goering, Geert heijen,boudewijn haverkort,Robert Haarman.** *The effect of mobility on local service discovery in the ahoy ad-hoc network system, Proceedings of the 4th European performance engineering conference on Formal methods and stochastic models for performance evaluation.* Berlin,Germany : Springer-Verlag Berlin, Heidelberg, 2007. 978-3-540-75210-3 .
3. **Martin Bäckström, Andreas Havdrup, Tomas Nylander, Jari Vikberg,Peter Öhman Ericsson Review No. 2, 2005.** Mobile@Home—GSM services over wireless LAN. Review No. 2, 2005.
4. **Upkar vashney, Ron Vetter.** *Emerging mobile and wireless networks,Communications of the ACM,Pages: 73 - 81 .* New York : ACM, USA, 2000. ISSN:0001-0782 .
5. **Felkey, Brent I. Fox and Bill G** *PDA Interface, Expanding your hardware capabilities..* Facts and Comparisons 2003, Volume 38, Number 1, pp 90–92.
6. **Philipp Offermann, Olga Levina, Marten Schönherr, Udo Bub,** *Outline of a Design Science Research Process. Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology.* ACM 2009 :, Pages: 1-1.
7. **Stuart Cheshire, Daniel Steinberg.** *Zero Configuration Networking: The Definitive Guide .* s.l. : O'Reilly Media, December,2005. 0-596-10100-7.
8. **Droms, R.** *Dynamic Host Configuration Protocol, Request for Comments: 2131 .* s.l. : <http://www.ietf.org/rfc/rfc2131.txt>, March 1997.
9. **Y. Goland, T. Cai, P.Leach** *Simple Service Discovery Protocol.* s.l. IETF Draft, October 1999, Vols. draft- cai-ssdp-v1-03-.txt, .
10. **Jini Introduction.** *TTM47AC Laboratory in construction of self-configuring systems, Fakultet for informasjonsteknologi, matematikk og elektroteknikk,INSTITUTT FOR TELEMATIKK*[Online] [http://www.item.ntnu.no/fag/ttm47ac/Information/Jini\\_Introduction.pdf](http://www.item.ntnu.no/fag/ttm47ac/Information/Jini_Introduction.pdf)
11. **Jini Technology** *The community Resource for Jini Technology..* [Online] 2006. The community Resource for Jini [http://www.jini.org/wiki/Jini\\_Architecture\\_Specification](http://www.jini.org/wiki/Jini_Architecture_Specification) .
12. **E. Guttman, C. Perkins,Sun Microsystems,J. Veizades, M. Day** *RFC 2608, Service Location Protocol version 2.* s.l. : IETF,Network Working Group, 1998.
13. **Salutation Consortium** *White Paper: Salutation Architecture.* [www.salutation.org](http://www.salutation.org) :, 1999.

14. **Salutation Consortium**, *Salutation Architecture Specification Version 2.0* [Online] . <http://systems.cs.colorado.edu/grunwald/mobilecomputing/papers/salutation/sa20e1a21.pdf>, 1 June 1999.
- 15., **IrDA-Infrared Communication Interface revision. 6/04** s.l. : copyright Cambell Scientific, inc 2003-2004 .
16. **W3C**. web service. [Online] 2006. [www.webservice.com](http://www.webservice.com).
17. **Prof Do Van Thanh**,. *Semantic Web Lecture 1*. Slide 6-9 : Department of telematics, NTNU, Trondheim, 2009.
18. **Matteo Villa, Giovanni Di Matteo,Roberto Lucchi, Michel Millot, Ioannis Kanellopoulos**. *INSPIRE NETWORK SERVICES SOAP Framework* . s.l. : European Commission, Joint Research Centre. EUR 23635 - 2008.
19. **Martin Gudgin, Marc Hadley, Noah Mendelsohn,Jean-Jacques Moreau**. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). [Online]. <http://www.w3.org/TR/soap12-part1/> 27 April 2007.
20. **Erik Christensen, Francisco Curbera,Greg Meredith,Sanjiva Weerawarana**. Web Services Description Language (WSDL) 1.1. W3C. [Online] [http://www.w3.org/TR/wsdl#\\_introduction](http://www.w3.org/TR/wsdl#_introduction).
21. **W3C Recommendation**,. *XML Schema Part 0: Primer Second Edition*. 28 October 2004. [Online]: <http://www.w3.org/TR/xmlschema-0/> .
22. **W3School**.. *w3school*. [Online] [http://www.w3schools.com/wsdl/wsdl\\_documents.asp](http://www.w3schools.com/wsdl/wsdl_documents.asp)
23. **Erik Christensen, Francisco Curbera,Greg Meredith,Sanjiva Weerawarana**. Web Services Description Language (WSDL) Version 1.2. [Online] <http://www.w3.org/TR/2003/WD-wsdl12-20030611/wsdl12.pdf>.
24. **Introduction to UDDI Concepts,Important Features and Functional**. *Oasis UDDI*. October 2004. online : <http://www.uddi.org/pubs/uddi-tech-wp.pdf>.
25. **Newcome, Eric**. *Understanding Web services: XML, WSDL, SOAP, and UDDI, Chapter 5 Finding web services Page 151-162*. s.l. : Addison-Wesley, 2002. 0201750813, 9780201750812.
26. **Accenture, Ariba, Inc., Commerce One, Inc., Fujitsu Limited, Hewlett-Packard Company, i2 Technologies, Inc**. UDDI Technical white paper. 6th september 2000.
27. **Reynolds, Fred Hartman and Harris**. *Was the Universal Service Registry a Dream?* Web Services Journal,, Vols. Dec 2, 2004.

28. **Barbara Carminati, Elena Ferrari, Patrick C.K. Hung.** Exploring Privacy Issues in Web Services Discovery Agencies. *IEEE Security and Privacy*. Sep/Oct 2005, Vols. vol. 3, no. 5, pp. 14-21.
29. **IEEE Systems, Man, and Cybernetics Society.** *Different Interpretations by Syntactic Search*. s.l. : Page 270-274, IEEE International Conference on Systems, Man, and Cybernetics ,1996.
30. **WHITEPAPER INTERDigital Media Independent Handover : WHITEPAPER (C).** s.l. InterDigital Inc, King of Prussia, PA 19406 USA. [online] [http://www.interdigital.com/images/id\\_pubs/InterDigitalMIHWhitePaper\\_Apr09.pdf](http://www.interdigital.com/images/id_pubs/InterDigitalMIHWhitePaper_Apr09.pdf).
31. **Lampson, Butler W.** *Designing a Global Name Service*, Digital Equipment Corporation. Digital Equipment, 130 Lytton Avenue, Palo Alto, CA 94301 : s.n. ACM 0-89791-198-9/86/0800-0001.
32. **Gnutella.** <http://gnutella.wego.com>. [Online]
33. **Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan.** *Chord: A Scalable Peer-to-peer Lookup Service for Internet*. s.l. : IEEE Press Piscataway, NJ, USA, FEBRUARY2003.
34. **Cliff Kettemborough,** *Introduction to Use Case Diagrams*. March 5, 1999.
35. **(W3C), World Wide Web Consortium.** SOAP Version 1.2 Part 1 – Messaging Framework (Second Edition). <http://www.w3.org/TR/soap12-part1>. [Online] 27 April 2007.
36. **Onion, Fritz.** "Essential ASP .Net with Examples in C#". Microsoft .Net Development Series. [book auth.] MA: Addison Wesley. Boston : ISBN 0201760401, 2004.
37. **Microsoft Internet Information Services (IIS) 6.0 Resource Kit.** ISBN:0735614202 : Microsoft Press, 2003 .
38. **(W3C), World Wide Web Consortium.** OWL Web Ontology Language Overview. <http://www.w3.org/TR/2004/REC-owl-features-20040210/#s3>. [Online] 10 February 200.
39. **David S Platt,** *The Microsoft Platform Ahead*. ISBN 0735620644 : Microsoft Press Redmond, WA, USA , 2004.
40. **Louis Columbus.** *The Microsoft Windows XP Professional Handbook*. ISBN:1584502193 : Charles River Media, Inc. Rockland, MA, USA , 2002 .
41. **Brian Johnson, Criag Young, Marck Skibo.** *Inside Microsoft Visual Studio .NET*. s.l. : Microsoft Press Redmond, WA, USA .

**42. OWIDotNetApi.**

OwIDotNhttp://users.skynet.be/bpellens/OwIDotNetApi/documentation.html. [Online] [Cited: 06 10, 2010.]

**43. Holger Knublauch, Ray W. Fergerson, Natalya F. Noy and Mark A. Musen** “*The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications*”. .. The Semantic Web (ISWC 2004), 2004, pp. 229-243. , s.l. : The Semantic Web (ISWC 2004), , 2004, Vols. pp. 229-243.

**44. Robert M Colomb,** *Frontiers in Artificial Intelligence and Applications; Vol. 156, Proceeding of the 2007 conference on Ontology and the Semantic Web.* s.l.: IOS Press Amsterdam, The Netherlands, The Netherlands, 2007. ISBN:0922-6389 , 978-1-58603-729-1 .

**45. Phillips, Jeremy J. Carroll and Addison.** Multilingual RDF and OWL : s.n., Vol. copied from 10.1.1.85.143.

**46. Ray Rankins, Paul Bertucci,Chris Gallelli, Alex T Silverstein.** *Microsoft(R) SQL Server 2005 Unleashed.* s.l. : Sams Indianapolis, IN, USA, 2006. 0672328240 .