Multi-ontology web services discovery

Le, Duy Ngan

2008

Le, D. N. (2008). Multi-ontology web services discovery. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/13591>

<https://doi.org/10.32657/10356/13591>

NANYANG TECHNOLOGICAL UNIVERSITY

**MULTI-ONTOLOGY WEB SERVICES
DISCOVERY**

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

By

Le Duy Ngan

School of Computer Engineering

May 8, 2008

Acknowledgements

First and foremost, I would like to express my special gratitude to my supervisor and mentor, Professor Angela Goh for her invaluable advice and enthusiastic help. In the past few years, I have not only learnt to conduct academic research but also many good characteristics from her.

Second, I wish to express my thanks to people who shared their ideas, gave their comments, and spent their time in discussion with me. They are Lee Ken Yoong, Do Tien Dung, and Zhou Chen in NTU; all members of the Integrated Manufacturing and Service Systems (IMSS) pilot project in the Singapore Institute of Manufacturing Technology (SIMTech). Special thanks go to Goh Chong Minsk, Tan Siew Poh, Ramasamy, and Hoang Huu Hanh; all from SIMTech and Michael C.Jaeger of Berlin University.

Next, I would like to acknowledge the scholarship support granted by the Nanyang Technological University (NTU). My gratitude also goes to staff and students in the Parallel Distributed Computing Centre (PDCC) including Assoc Prof Stephen Turner (director of the PDCC), Ng-Goh Siew Lai and Tay Siew Eng. My thanks also go to Vietnamese teachers who recommended me for the scholarship which has changed my life.

Finally, I wish to express my deepest gratitude to my parents, my younger sister Le Thuy Ngoc, and my closest friend, Nguyen Hong Khoat in Vietnam, for their love and encouragement which has helped me overcome many difficulties faced during the

research. They are the most important people in my life and this work is my gift to them. Thanks to my parent who have tried their best to give my younger sister and I a good education.

Singapore, October 15th 2007

Le Duy Ngan

Table of Contents

<i>Chapter 1</i>	Introduction.....	1
1.1	The Semantic Web.....	1
1.1.1	Web Services	3
1.1.2	Semantic Web Services.....	3
1.2	Motivation and Scope	4
1.2.1	Motivation.....	4
1.2.2	Scope.....	6
1.3	Major Contributions.....	7
1.4	Outline of the Thesis.....	9
<i>Chapter 2</i>	Related Technologies.....	11
2.1	The Semantic Web.....	11
2.1.1	Semantic Web Architecture	11
2.1.2	Ontology	13
2.2	Web Services	20
2.2.1	Web Services Standards.....	20
2.2.2	Web Services Model.....	23
2.3	Semantic Web Services.....	25
2.4	Summary	31
<i>Chapter 3</i>	Web Services Discovery: the State of the Art	32
3.1	A Survey of Web Services Discovery Systems	32
3.1.1	Categories and a Taxonomy of Web Service Discovery Systems	33
3.1.2	Discovery of Web Services using the Same Ontology	36
3.1.3	Discovery of Web Services using Different Ontologies.....	44
3.2	Web Services Discovery Problems and Approaches	46
3.2.1	Matching Web Services Using Different Ontologies	46
3.2.2	Matching Semantic Web Services against Non-Semantic Web Services.....	47
3.2.3	Matching Semantic Web Services that Use Different Description Languages	49
3.3	Summary	50
<i>Chapter 4</i>	Comparing Two Ontologies.....	52
4.1	Introduction.....	52
4.2	Ontology Comparison Algorithm	55
4.2.1	Ontology Relationship Definitions	55
4.2.2	Ontology Comparison Algorithm Description	57
4.2.3	Complexity of the Algorithm.....	60
4.3	Prototype, Example, and Experiment	61
4.3.1	Ontology Comparison Examples	62
4.3.2	Experiment and Results	68
4.4	Related Work	70
4.5	Summary	74
<i>Chapter 5</i>	Computing Ontological Concept Similarity	76
5.1	Introduction.....	76

5.2 Related Work	78
5.2.1 Concept Similarity in the Same Ontology	79
5.2.2 Similarity between Concepts from Different Ontologies	81
5.3 Concept Similarity in the Same Ontology	84
5.3.1 Concepts are Subsumed	84
5.3.2 Concepts are not Subsumed	87
5.4 Concept Similarity in Different Ontologies	88
5.4.1 Syntactic Similarity	90
5.4.2 Property Similarity	96
5.4.3 Domain Similarity	99
5.4.4 Neighborhood Similarity	101
5.4.5 Discussion	103
5.5 Prototype and Testing	104
5.5.1 Prototype of the Algorithm	104
5.5.2 Concept Similarity Measurement by Oundhankar <i>et al.</i>	106
5.5.3 Examples of Concept Similarity Tests	109
5.5.4 More Tests	117
5.6 Summary	119
Chapter 6 MOD Framework	121
6.1 Introduction	121
6.2 MOD Algorithm	122
6.2.1 MOD Algorithm Description	122
6.2.2 Matching Components	125
6.2.3 Complexity of the Algorithm	128
6.3 MOD Engine	130
6.4 Summary	132
Chapter 7 MOD Application and Experiment	133
7.1 A Supply Chain Scenario	133
7.2 Prototype and Oundhankar <i>et al.</i> 's System	136
7.2.1 MOD Prototype	136
7.2.2 Discovery System by Oundhankar <i>et al.</i>	137
7.3 Experiments	139
7.3.1 Sample Ontologies used	139
7.3.2 Examples	142
7.3.3 More Tests	148
7.4 Summary	149
Chapter 8 Conclusions and Future Work	150
8.1 Conclusions	150
8.2 Future Work	153
8.2.1 Supporting Non-Semantic Web Services and Different Description Languages	153
8.2.2 Applying Fuzzy Logic	153
8.2.3 Applying the Concept Similarity Measurement to Ontology Operations	155
Publications Derived from this Research	173
Appendix A	- 1 -
Appendix B	- 3 -

List of Figures

Figure 2.1: The Semantic Web Architecture [109].....	12
Figure 2.2: RDF model and example.....	15
Figure 2.3: A graphical view of <i>equipment</i> ontology using Protégé	18
Figure 2.4: A representation of a portion of the <i>equipment</i> ontology.....	19
Figure 2.5: Web service model	24
Figure 2.6: Top level of the service ontology	28
Figure 2.7: The representation of a partial OWL-S service profile	30
Figure 3.1: Taxonomy of Web service discovery systems based on the use of semantics.....	35
Figure 4.1: Cases of using ontology	53
Figure 4.2: Data structure declaration.....	57
Figure 4.3: Prototype of the ontology comparison algorithm.....	61
Figure 4.4: Technology, Equipment, and Computer ontologies using Protégé.....	64
Figure 4.5: Ontology common parts	65
Figure 4.6: <i>agent</i> ontology, common ontology, and <i>fipa-agent</i> ontology using the SWOOP Editor [44]	67
Figure 4.7: Example of a local approach for ontology alignment [1].....	74
Figure 5.1: The taxonomy of approaches to compute concept similarity	78
Figure 5.2: The relationship of concepts.....	84
Figure 5.3: An example of measuring concept similarity	90
Figure 5.4: Undirected graph to represent synonym set relations [96]	92
Figure 5.5: Prototype of the algorithm.....	105
Figure 5.6: Users customize parameter weights and semantic similarity values.....	106
Figure 5.7: Summary of concept similarity results	116
Figure 5.8: Concept similarity results of 300 concept pairs and the ground truth.....	117
Figure 6.1: A schematic description of the matching algorithm between two Web services	123
Figure 6.2: MOD Engine	130
Figure 7.1: A supply chain model using MOD.....	135
Figure 7.2: Administrator Interface	136
Figure 7.3: User Interface	137
Figure 7.4: Ontology in eBusiness domain.....	139
Figure 7.5: Ontology for computer device domain.....	140
Figure 7.6: Ontology for price domain	141
Figure 7.7: Requested and advertised services	142
Figure 7.8: Ontology for computer hardware device domain.....	146
Figure 7.9: Toshiba request service	147
Figure 8.1: Fuzzy numbers	155

List of Tables

Table 3.1: A comparison between the related works based on the stages	41
Table 3.2: Advantages and disadvantages of three approaches to Semantic Web discovery based on the same ontology	44
Table 4.1: Partial results of comparing 10 ontology pairs	69
Table 4.2: More testing for real world ontologies	69
Table 5.1: Weight mapping table.....	89
Table 5.2: Concept label similarity examples.....	94
Table 5.3: Concept description similarity examples.....	95
Table 5.4: Concept label similarity examples.....	96
Table 5.5: Range similarity values when range is primitive data type	97
Table 5.6: Similarity values when range is primitive data type.....	99
Table 5.7: <i>SupSim</i> and <i>SubSim</i> similarity measurement.....	102
Table 5.8: Clusters and their information	110
Table 5.9: Concept pairs in a common ontology	111
Table 5.10: The influence of domain on the concept similarity measurement.....	113
Table 5.11: Concept pairs from different ontologies and different domains.....	114
Table B.1: Completed results of comparing 100 ontology pairs. Each ontology is represented by its name and root. Two ontologies have “sub ontology” relationship has one common part.....	- 3 -

Abstract

The Semantic Web promotes machine understandability in the Web through the use of ontologies. An ontology is a specification of classes, properties, and their relations, from which new knowledge can be inferred. By enhancing traditional Web services with Semantic Web technology, Semantic Web services have emerged as a promising means of enhancing e-business and e-commerce operations. As a result, Semantic Web services have been rapidly developed. Though the numbers may not be as large as current Web pages and non-semantic Web services, the increasing use of Semantic Web service description languages such as OWL-S indicates that this number will increase further in future. This has led to a demand for a discovery mechanism.

Researchers have developed discovery systems to discover Semantic Web services which satisfy the requirements of a requester. Current discovery systems locate Semantic Web services that are based on the same ontology but they do not support the discovery of Semantic Web services that use different ontologies. Thus, even if the Semantic Web service providers meet the requirements of a Semantic Web service requester, such a match may be missed. Unfortunately, in the real world, it is highly probable that requesters and providers utilize different ontologies to describe their services. Therefore, a discovery system that supports Web services using different ontologies is important.

We have developed MOD, a **M**ulti-**O**ntology web services **D**iscovery system, which matches Semantic Web services described by the same as well as different ontologies. Techniques related to ontology and Web service matching such as ontology comparison and concept similarity measurement algorithm have been investigated. Ontology comparison determines if two ontologies are related. It is employed in measuring concept similarity. If it can be determined that two concepts, which originate from two different ontologies, actually belong to a common ontology, the accuracy of the measurement can be enhanced. Moreover, considering domain similarity has avoided mismatches and improved the matching of the overall concept similarity. In turn, concept similarity measurement is employed to measure Web service similarity which is the core of MOD. The proposed techniques are not only used in MOD but are also applicable to other ontology and Semantic Web service activities. MOD is superior to existing Web service discovery systems as it supports multiple ontologies and enhances the accuracy of Web service discovery.

Glossary

DAML	- DARPA Agent Markup Language
DAML-S	- DAML-based Web Service Ontology
MOD	- Multi-Ontology Web services Discovery system
OWL	- Web Ontology Language
OWL-S	- OWL-based Web service ontology
RDF	- Resource Description Framework
RDFS	- RDF Schema
SOAP	- Simple Object Access Protocol
UDDI	- Universal Description, Discovery and Integration
URI	- Uniform Resource Identifier
W3C	- World Wide Web Consortium
WSDL	- Web Service Description Language
WSMO	- Web Service Modeling Ontology
XML	- Extensible Markup Language

Chapter 1

Introduction

1.1 The Semantic Web

The World Wide Web (WWW) is considered one of the greatest inventions of the twenty-first century. It is called “a revolution of information” since it provides new ways to share, search, and publicize information easily and more conveniently than ever before. HyperText Mark-up Language (HTML) [109] is one of the most important reasons behind its success. Most early Web documents were formatted using HTML, enabling documents to be linked to each other. Currently, the World Wide Web has billions of Web pages and continues to grow rapidly. As a consequence, there has been an enormous demand to have mechanisms to retrieve information. Search engines such as Google (www.google.com), Yahoo (www.yahoo.com), Excite (www.excite.com), and so on have been developed to satisfy this demand.

However, the documents which are formatted in HTML are unable to capture the semantics of the contents. As a result, three main disadvantages in Web searching are encountered. Firstly, searches produce a large number of results, many of which are inaccurate. Additionally, users must manually select from a large number of links in the results. Furthermore, without semantic considerations, synonyms of keywords are

ignored and hence, the results may be incomplete. Moreover, by using HTML, applications are unable to exploit the rich information sources available in the Web.

To overcome these problems, Tim Berners-Lee, the “inventor” of the World Wide Web, has coined the term “Semantic Web” [5]: *“The Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.”* The Semantic Web is the next generation of the Web and is an extension of the current World Wide Web. It attempts to overcome the drawbacks of the current Web by using metadata to describe the semantics of the data. When metadata is used to mark-up data, the data is not only human readable but also understood by machines. An ontology is a specification of concepts, properties, and relationships that can be used to represent and infer knowledge, and shared by communities or programs. These ontologies are used as metadata in the Semantic Web and provide many advantages such as knowledge representation and reasoning. By using ontologies to describe knowledge, the Web can be brought to its full potential.

To enable the Semantic Web to become a reality, more ontologies need to be created to store knowledge of different domains. Moreover, ontology activities such as comparing, merging, aligning, integrating and so on must be fully supported in order to promote its usage. These will take a considerable amount of effort. In the meanwhile, attempts have been made in existing Web applications to provide intelligence by annotating keywords on the Web.

1.1.1 Web Services

A Web service is an essential enabler of e-business and e-commerce through *discovery*, *composition*, *invocation*, *monitoring*, and other functions. Web services allow companies to reduce cost by fast, effective, and reliable services to customers, suppliers, and partners over the Internet. They also enable business operations to function more efficiently via the Web and enhance business opportunities to companies.

Web services which are supported by W3C - the World Wide Web Consortium - (www.w3.org) include three standard technologies, namely, Web Service Description Language (WSDL), Universal Description, Discovery and Integration (UDDI), and Simple Object Access Protocol (SOAP). WSDL [12,113] is used to describe the services. UDDI [74,113] is used to publish and search Web services. Web services communicate with each other via SOAP [6,113]. However, based on these standard technologies, the Web service cannot fully satisfy the requirements of business applications because of limitations in automatic discovery and composition. For example, a Web service is unable to “know” that *notebook* and *laptop* refer to the same thing. The reason for this shortcoming is the lack of semantic understanding. To overcome this problem, Web services require a method to incorporate semantics.

1.1.2 Semantic Web Services

Just as the Semantic Web is an extension of the current World Wide Web, the Semantic Web service is an extension of the Web service. It overcomes the Web service limitations by using knowledge representation technology from the Semantic Web. Specifically, it

uses ontologies to describe its service instead of using WSDL. Such ontologies can be understood by machines and can be reasoned upon. In other words, Semantic Web service is a novel technology that combines the advantages of Semantic Web and overcomes disadvantages of Web service. By employing Semantic Web techniques to present data, a Semantic Web service has the following advantages over traditional Web services:

- A software agent or a computer program is able to understand the semantics of Semantic Web services described by metadata.
- Ontologies describing Semantic Web services can be shared and reused among different software agents or computer programs.
- A Semantic Web service is able to be described by a rich knowledge-expressive model so that the knowledge can be managed effectively.
- New knowledge can be inferred from the existing knowledge making Semantic Web service more intelligent.

The above benefits enable a fully automatic *discovery*, *composition*, and *invocation* of Semantic Web services.

1.2 Motivation and Scope

1.2.1 Motivation

Web service discovery is an important task in the entire Web service oriented architecture. A Web service is dysfunctional if it cannot be discovered. UDDI [74,113] provides

standard registration and discovery of Web services. We can search by service name, service category, and other information through UDDI. However, the search provided by UDDI is only limited to keyword search. These keyword searches cannot recognize the Web services with different keywords but have the same or similar meaning. Web services discovery has been used in e-business, workflow, e-learning, etc. In the workflow field, Web services discovery is used to automatically locate Web services, in order to fulfill some services that are required in the workflow.

Researchers have developed matchmakers and discovery systems [47,48,56,62,76,98,101-103] to match Semantic Web service requesters against Semantic Web service providers. The Web service providers advertise their information in the discovery systems. When a requester with a certain requirement wants to find a service, the system will attempt to match the request with the advertised information of providers in its database and produce a list of providers, sorted by degree of relevance.

Current discovery systems are adequate when the Semantic Web service requester and provider use the same ontology. Unfortunately, researchers have not focused much on the situation where a Semantic Web service requester and provider use different ontologies. Consider the following scenario which illustrates the importance of supporting Web service using different ontologies: Some companies in Singapore provide computer solution services. They define their own ontology in the computer domain to describe their services. A company in Vietnam wishes to buy a computer and it defines its own ontology in the same computer domain to describe its services. There are no global ontologies in the domain and the provider and the requester do not know each other. So, the two ontologies they define in the same computer domain are different. Current

discovery systems are unable to match the Web service requester and provider in such a situation, though obviously, the providers can provide an exact fit to the requester's demands.

Thus, in the real world, a Web service provider can provide an exact service to the requester even though both services use different ontologies. Therefore, a discovery system that supports Web services based on different ontologies is important and provides the motivation behind this work. This thesis introduces a Multi-Ontology Semantic Web services Discovery system (termed MOD). The system supports the discovery of Web services which are based on the same ontology as well as on different ontologies.

1.2.2 Scope

There are many issues related to Semantic Web service such as discovery, invocation, composition, and monitoring. Each issue is an interesting research area and provides a challenge to the research community. The focus of this work is on Web service discovery. The other issues are outside the scope of our work. For example, we will introduce how Web services are invoked and composed but will not undertake to explore these areas in this thesis.

Additionally, as mentioned above, the Web service provider and Web service requester do not know each other and they may use different ontologies. We assume that a group of providers offer services in a specific domain, and they develop an ontology to describe their services. Similarly, a requester develops an ontology to describe his Web service.

Since there is no standard test data or benchmark available, we have developed 500 Web service profiles. The Singapore Institute of Manufacturing Technology (SIMTech) (www.simtech.a-star.edu.sg) was chosen to determine the “ground truth” in the tests because of their expertise in the Business to Business (B2B) domain.

1.3 Major Contributions

We have proposed a number of novel techniques related to ontology and Semantic Web service discovery. The major contributions of this research are summarized as follows:

- *Comparing two Ontologies.* Once an ontology is created, it can be used by many different applications. Such applications may use the entire ontology or only a part of it. Furthermore, they may use a part of the ontology and then extend it by adding more knowledge. Since ontologies which are used by different organizations, may be related, there is a need to check these relationships. We have introduced an algorithm that compares two ontologies to determine their relationship. The algorithm can be applied to many applications. In this research, it is used to measure concept similarity.
- *Measuring Concept Similarity.* Ontologies are used in applications related to knowledge management, natural language processing, e-commerce, knowledge engineering, cooperation and integration of information systems, and so on. In order to promote the use of ontology, there are many operations such as mapping, merging, alignment, integration, and evolution that must be supported. Determining the semantic similarity between concepts is the core of these

operations. The thesis presents an algorithm for measuring concept similarity. It improves on other existing work by supporting measurement of concepts from different ontologies and adding the domain similarity dimension in the measurement. Moreover, by checking whether the two concepts belong to a common part of the two ontologies, the computation of concept similarity is more accurate.

- *MOD – A Multi-Ontology Web Services Discovery System.* Current discovery systems are adequate when Semantic Web service requester and provider use the same ontology. Unfortunately, researchers have not focused much on the situation where a Semantic Web service requester and provider use different ontologies. We have designed and developed MOD, a system which supports Web services based on the same ontology as well as on different ontologies.
- *A Taxonomy of Current Web Service Discovery Systems.* Web service discovery systems have been developed increasingly. A survey of Web service discovery systems has highlighted the advantages and disadvantages of each system. A taxonomy of Web service discovery systems has been proposed, focusing specifically on the aspect of semantic support. We also elaborate on open issues relating to such discovery systems.
- *A Scenario and Benchmark for Web Service Discovery systems.* Since there is a lack of related scenarios for Web service discovery systems, we introduce a scenario in the e-supply chain domain that highlights the motivation of MOD and how it works. As there is no standard test data or benchmark available, we have

developed a hundred Web service profiles based on the scenario. The scenario and test data should be useful for other discovery systems as a test bed.

1.4 Outline of the Thesis

This chapter has introduced the Semantic Web, Web service, and Semantic Web service as a backdrop to the motivation of the work and its scope. The rest of the thesis is organized as follows:

Chapter 2 “Related Technologies” presents a background of related technologies utilized in the discovery system, an important aspect of which is the Semantic Web service. Web service architecture which is the interaction of three parties, namely, a requester, providers, and a discovery system is introduced.

Chapter 3 “Web Service Discovery Systems: State of the Art” presents a survey of Web service discovery systems. We propose a taxonomy to classify these discovery systems. The advantages and disadvantages of each system are highlighted. Shortcomings of current Web service discovery systems are pointed out.

Chapter 4 “Comparing two Ontologies” introduces an algorithm to compare two ontologies. The chapter begins with a discussion on the motivations for comparing two ontologies and how it is employed in MOD, followed by the description of the algorithm. Examples consisting of developed ontologies as well as real world ontologies are introduced. Finally, testing of the algorithm and an analysis is given.

Chapter 5 “Measuring Concept Similarity” introduces the concept similarity measurement. The chapter begins with a brief discussion on the current approaches to concept similarity measurement. The advantages and disadvantages of each approach are discussed. The proposed concept similarity algorithm which overcomes the disadvantages of current approaches is presented. Examples and experiments based on developed ontologies as well as real world ontologies are presented and discussed.

Chapter 6 “MOD Framework” presents the algorithm and architecture of MOD. The chapter first introduces the motivation and advantages of MOD. Next, the MOD algorithm is described, followed by its architecture.

Chapter 7 “MOD Application and Experiment” demonstrates the application and validity of MOD. The chapter first introduces an e-supply chain scenario in an *equipment* domain. Next, examples based on the scenario are presented to illustrate how MOD works. Finally, experimental results are presented.

Finally, *Chapter 8 “Conclusions and Future Work”* concludes the thesis with a summary and suggests directions for further research.

Chapter 2

Related Technologies

This chapter introduces further details of the Semantic Web, Web service, and Semantic Web service technologies which were mentioned briefly in chapter 1. Firstly, the architecture of the Semantic Web is introduced, followed by an elaboration of the concept of ontology in section 2.1. Web service standards such as WSDL, UDDI, and SOAP are briefly explained and Web service model which has been recommended by W3C is introduced in section 2.2. Next, Semantic Web service functions such as automatic discovery, composition, and monitoring are presented in section 2.3. Finally, the chapter is summarized in section 2.4.

2.1 The Semantic Web

2.1.1 Semantic Web Architecture

Berners-Lee, the ‘inventor’ of the World Wide Web, proposed the Semantic Web architecture (figure 2.1) [109]. The architecture includes seven layers, namely, *Foundation*, *XML Schema*, *RDF Schema*, *Ontology*, *Logic*, *Proof*, and *Trust layer*.

Figure 2.1: The Semantic Web Architecture [109]

- Foundation layer consists of Uniform Resource Identifier (URI) and Unicode. URI is used to identify the Web resource and Unicode standard is used to encode the documents.
- The second layer is XML Schema which is an XML based alternative to DTD (Document Type Definition) and is used to describe the structure of an XML document.
- The third layer is RDF Schema which includes RDF + rdfschema. RDF which stands for Resource Description Framework is a language for representing resource information in the Web. RDF schema is a language for declaring basic class and types for describing the terms used in RDF. The second and third layers define objects and classes, their relationships and constraints.
- The next layer is the Ontology vocabulary which provides a way to express more semantics which cannot be represented in the above schema layers. Since

ontology is used in Semantic Web service, it will be described in detail in the following subsection.

- Logic layer enables inference from existing knowledge. This layer is usually integrated with the Ontology layer.
- Proof Layer confirms the truth of a statement. The Trust layer resolves conflicts between knowledge carried by the Semantic Web.
- The Trust layer has not been extensively developed. Digital Signature is used to secure documents. Currently, both the Proof and Trust layers are less established compared with the other layers.

2.1.2 Ontology

This section presents an overview of ontology including the motivations, definition, and categories as well as a framework of ontology languages. Next, the OWL Web Ontology Language [110] which is currently the de-facto standard recommended by the World Wide Web Consortium (W3C) for ontology description in the Semantic Web is introduced.

2.1.2.1 An Overview of Ontology

Ontologies are used in a broad number of applications related to knowledge management, natural language processing, e-commerce, knowledge engineering, cooperation and integration of information systems. This is because they provide a facility to describe knowledge expressively as well as present a means of sharing, reuse, and common understanding of knowledge that can be communicated between people, agents, and

systems. There are many definitions of ontologies, but a well-known definition from Gruber is as follows: “An ontology is a formal, explicit specification of a shared conceptualization” [33]. Hence, an ontology provides an abstract, simplified view of the world that we wish to represent for some purposes.

In order to represent knowledge of a certain domain of discourse, a language to express such knowledge is required. The choice of language is an important step in the ontology development process. The selection is usually based on the inference mechanisms and platforms that the application is deployed in. Many ontology languages have been proposed and we classify them into three categories, namely, first-order logic, frame-based logic, and description logic. The first category uses first-order logic theory to represent and infer knowledge. KIF (Knowledge Interchange Format) [30] and CycL [55] are typical examples of this kind of language. The second category uses frame-based model to represent the knowledge. Examples of this group include Ontolingua [24], Frame logic [50], and GFP (Generic Frame Protocol) [46]. The third category describes the knowledge in terms of concepts information and relationship. RDF [111], DAML [15], and OWL [110] are examples of the last category.

Several ontology languages which are classified into the third category have been developed to support the Semantic Web. The languages are classified into three groups: HTML-based, XML-based, and RDF-based. Simple HTML Extension (SHOE) [35] and Ontobroker [25] are two typical examples of the HTML-based group. In this group, knowledge is represented by HTML tags. Therefore, ontologies in this group do not support schema definition and therefore it is difficult to be defined. To overcome this shortcoming, XML-based ontology description languages which were proposed support

XMLS (XML Schema) or DTD (Document Type Description). Semantic information can be embedded directly into XMLS or DTD. However, ontologies in this group have limitations in expressing semantics. XOL (XML-Based Ontology Exchange Language) [45] and OML (Ontology Markup Language) [49] are typical examples of XML-Based languages.

Figure 2.2: RDF model and example

RDF-based languages overcome the semantic limitations found in XML-based languages. RDF (Resource Description Format) [111] which is based on XML syntax, represents data semantically through a triple: subject, predicate, and object. Semantics of data are presented through a relation between a subject and an object through a predicate. Figure 2.2 shows the RDF model (figure 2.2a), an example of the model (figure 2.2b), and the representation of the example (figure 2.2c). The example describes the relationship between a “Computer” (subject) “made in” (predicate) a “Country” (object). The RDF Schema (RDFS) can be used to define class hierarchies. Hence, RDF can be used to represent ontology more effectively. DARPA Agent Markup Language (DAML) [15] and Web Ontology Language (OWL) [110] are two popular RDF-based ontology languages. DAML is based on RDF enhanced with the object-oriented approach. OWL is an extension of DAML, allowing various types of relationships between classes to be defined.

Generally, an ontology language describes two types of entities, namely, classes and relationships. Different languages may use different terms and detailed description for defining the entities. Classes are described through a set of features called properties and instances. Relationships are used to represent the connection between concepts or properties. In other words, an ontology includes descriptions of classes, instances of classes, properties, as well as range and domain constraints on properties. It also contains various types of relationships between classes or properties. These relationships form the structure of the ontologies. The most common and important relationship is sub or super class relationship which determines the hierarchical structure of an ontology.

2.1.2.2 OWL - Web Ontology Language

The OWL Web Ontology Language [110] which is a semantic markup language for publishing and sharing ontologies on the World Wide Web has been proposed by W3C [112]. As with other ontology languages, OWL has many elements which constitute the language but the four main elements are concept, property, relationship, and instance. Each aspect is elaborated upon as follows. In order to clarify the description, an *equipment* ontology using the Protégé system [99] is used (figure 2.3). A portion of its representation is shown in figure 2.4.

A class, which is also called concept, defines a group of instances that share common features. For example, *Desktop Computer*, *Laptop*, and *Notebook* in the *equipment* ontology are classes. Instances corresponding to actual entities can be grouped into these classes. For example, *Dell XPS Desktops*, *HP Pavilion Laptop* and *IBM r40 notebook* are instances of classes *Desktop Computer*, *Laptop*, and *Notebook*, respectively.

Classes can be organized in a specialization hierarchy using `subClassOf` relation. For example, the classes *Laptop* and *Notebook* are subclasses of the classes *Portable Computer*. *Desktop Computer* and *Portable Computer* are subclasses of the class *Personal Computer*. With these relationships, it could be inferred that if an instance is a *Desktop Computer* or *Portable Computer*, then it is also a *Personal Computer*. In this case, it can also be inferred that *Dell XPS Desktops* is *Personal Computer*. A class may have super classes. A class must have an identity and may have a name and a short description. It may optionally have one or more properties.

Figure 2.3: A graphical view of *equipment* ontology using Protégé

```

<?xml version="1.0"?>
<rdf:RDF xml:base="http://www.owl-ontologies.com/Technology.owl"
xmlns="http://www.owl-ontologies.com/Technology.owl#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Mother_Board">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Hardware"/>
    </rdfs:subClassOf>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">the main circuit board for a computer</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="Equipment">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">an instrumentality needed for an undertaking or to perf
  </owl:Class>
  <owl:Class rdf:ID="Mouse">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      a hand-operated electronic device that controls the coordinates of a cursor on your computer screen as you move it around on a
    </rdfs:comment>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Input_Device"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Unix">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Operating_System"/>
    </rdfs:subClassOf>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Unix is an operating system with a graphical user inter
  </owl:Class>
  <owl:Class rdf:ID="Webcam">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Input_Device"/>
    </rdfs:subClassOf>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">a digital camera designed to take digital photographs
  </owl:Class>
  <owl:Class rdf:ID="Office">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Office Software</rdfs:comment>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Software"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Spread_Sheet">
    <rdfs:subClassOf rdf:resource="#Office"/>
  </owl:Class>
  <owl:Class rdf:ID="Intel">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="CPU"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Personal_Computer">

```

Figure 2.4: A representation of a portion of the *equipment* ontology

Properties of a class are used to describe relationships between individuals or from individuals to data values. There are two types of property, namely, data type property and object property. For example, *Dell XPS Desktops* has *hasSoftware* relation with

Microsoft Windows Vista which is an instance of the class *Windows*. *Dell XPS Desktops* has *made_in* property where data type is a string. In the *equipment* ontology, the class *Computer* has four properties, namely, *hasSoftware*, *hasHardware*, *made_in*, and *year_of_manufacture*. A property may have sub or super properties. Values of instances are limited by a range of properties. In other words, range is a restriction on an instance. For example, the class *Computer* would have a cardinality restriction of **1** on the *year_of_manufacture* and **n** on *hasSoftware* property. A property has a name and a description. Additional vocabulary for describing properties and classes include: relationship between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties and characteristics of properties (e.g. "symmetry"), and enumerated classes.

2.2 Web Services

2.2.1 Web Services Standards

A Web service is a software component representing a specific business function that can be described, published and invoked over the network (typically Internet) using open-standards. It uses a remote invocation method based on XML [7] for its binding, and supports both asynchronous and synchronous interaction. A Web service has the following features: *platform independence*, *Internet scoped*, *loosely coupled*, and *easy interaction*. Web services are able to interact without running on the same platform. Web services can run on different networks (Internet, Intranet ...) and different operating systems (Windows, Linux ...), and can be implemented using different programming

languages (Java, C, C++...). A Web service has a global scope and can be invoked over the Internet, including computers that are fire-walled.

Prior to the availability of Web services, mechanisms such as Common Object Request Broker Architecture (CORBA) [69,70], Distributed Component Object Model (DCOM) [84,90], and Electronic Data Interchange (EDI) [42,78] had to be used in order to invoke some functionalities of another organization's computer. However, these techniques have limitations. Firstly, these distributed technologies use specialized, uncommon description languages. For example, CORBA uses Interface Definition Language (IDL) [18,58]. In contrast, Web services use WSDL [12,113] which is an XML based language and therefore, possesses the advantages of an XML document such as being an open standard, self-describing, and both human and machine readable. Secondly, these distributed techniques require exact function names to be called, whereas, Web services are loosely coupled. For example, a computer sales service has a configuration of the computer as the input, and price as the output. To invoke this service in the distributed techniques, the input must be a specific computer configuration which was described by the offered service. In the Web service, the input does not need to describe the specific computer configuration. It could be general or specific. Finally, those mechanisms do not operate well over the Internet in the presence of firewalls. Web services have overcome the shortcomings of the previous distributed mechanisms.

There are three standards used to describe the Web services, to store advertised Web service information, and to communicate between Web services in the Web service model. They are Web Service Description Language (WSDL), Universal Description, Discovery and Integration (UDDI), and Simple Object Access Protocol (SOAP).

- WSDL [12,113] is used to describe how to access a Web service and what operations (methods) it offers. It is also used to locate a Web service. A WSDL document defines a Web service through four major elements, namely, *port types*, *message*, *types* and *binding*.
- UDDI [74,113] is an industry specification for describing, publishing, and finding Web services. It allows developers to describe and classify their services with the technical details about the interfaces of the Web services which are exposed. UDDI also enables developers to consistently discover services, or interfaces of a particular type, classification, or function. It also defines a set of Application Programming Interfaces (APIs) that developers can use in order to interact with UDDI data directly. The UDDI scheme uses *White*, *Yellow*, and *Green pages* as data categories. Without the categorization, locating data within repositories would prove to be a very difficult task. Therefore, as in any repository, UDDI registration data must have the ability to assign category information. The UDDI schema uses *White*, *Yellow*, and *Green pages* as data categories. The *White* pages contain the overall background information including products, contact information, and so on about the company providing the Web service. The *Yellow* pages make it easy to locate similar Web services by categorising them into industrial standard taxonomies such as North American Industry Classification System (NAICS) or United Nations Standard Products and Services Code (UNSPSC). The *Green* pages provide the technical information needed such as a URI address of a SOAP or WSDL file describing the service and its behaviours in order to use the service.

- SOAP [6,113] is a communication protocol based on XML that allows applications to communicate over the Internet. SOAP does not define any programming model or implementation specific semantics. It describes a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. SOAP is independent of platform and language. A SOAP message is an XML document containing four elements, namely, *envelope*, *header*, *body*, and *fault*. A SOAP message is an XML document containing four elements *envelope*, *header*, *body* and *fault*. Among the elements, *envelope* and *body* are mandatory while *header* and *fault* are optional. The *envelope* element which is the top-level element in the SOAP document identifies an overall framework for expressing what is in a message. The *header* provides a flexible mechanism for extending a message in a decentralized and modular way without prior knowledge between the communicating parties. The *body* element provides a simple mechanism for exchanging mandatory information intended for the ultimate recipient of the message. The *fault* element is used to carry error and/or status information within a SOAP message.

2.2.2 Web Services Model

The Web service model in figure 2.5 shows the interaction between a service requester, service providers, and a service registry which is a Web service discovery system.

Figure 2.5: Web service model

- The providers offer Web services which provide functions or business operations. They are created by companies or organizations. In order to be invoked, the Web services must be described. This will facilitate discovery and composition. WSDL or service profile of Semantic Web service is used to carry out this function.
- The Web service requester describes requirements in order to locate service providers. Service requesters usually contain a description of the Web service, though it is not a Web service which can run on the Internet. The requirements are usually described by WSDL, service template or service profile.
- The Web service discovery system or service registry is a broker that provides registry and search functions. The service providers advertise their service information in the discovery system. This information will be stored in the registry and will be searched when there is a request from service requester. UDDI is used as a registry standard for Web service described by WSDL.

Semantic Web service discovery systems are developed to discover Semantic Web services.

The above three components interact with each other via publishing, discovery, and binding operations. These operations are elaborated upon as follows:

- *Publishing*: the Web service providers publish their service information through the discovery system for requesters to discover. Through the publishing operation, the Web service provider stores the service description in the discovery system.
- *Discovery*: the Web service requesters retrieve service providers from the service registry. Based on service description, which describes the requirements of the Web service requesters, the discovery system will output a list of Web service providers which satisfy the requirements.
- *Binding*: The Web service requester invokes the discovered Web service provider. The binding occurs at runtime. The Web service requesters and Web service providers will communicate via SOAP protocol [6,113] which is an XML based protocol for Web service information exchange.

2.3 Semantic Web Services

As mentioned in chapter 1, Semantic Web services are an extension of Web services by using knowledge representation technologies of the Semantic Web. For Semantic Web

services to become a reality, the markup languages must be sufficiently descriptive in order to be machine understandable. The requirements of such a language include:

- *Automatic Discovery.* Automatic Web service discovery is an automated process for locating Web services whose capabilities satisfy the requestor's requirements. For example, a user wishes to find a service that sells computers and accepts a particular credit card. Currently, this task must be performed by a human who might use a search engine to find a service. The user accesses the results through a Web page, and executes the service manually, once it has been determined that the constraints are met. Even if UDDI were used to locate Web services, the lack of semantics results in possible matches being overlooked. With Semantic Web services, the information necessary for Web service discovery could be specified as computer-interpretable semantic markup, and a service registry or agent could be used to locate the services automatically.
- *Automatic Invocation.* A software agent or a computer program must be able to automatically determine how to invoke or execute the service. A Semantic Web service provides a description of what an agent needs in order to execute the service. A software agent should be able to interpret the description to determine what input(s) is/are necessary to invoke the service, and what information will be returned.
- *Automatic Composition.* This task involves the automatic selection, composition, and interoperation of Web services to perform some complex task or a high-level description provided by requesters. Currently, the user must select the Web services and specify the composition manually. In Semantic Web services,

software agents are expected to select and combine a number of Web services to complete a complex objective.

- *Automatic Monitoring.* While the system is executing, software agents need to be able to verify and monitor the service properties automatically.

There are different Semantic Web service description languages and each uses different ontology description languages. For example, DAML-S [17], a semantic markup for Web service language, uses DAML [15] ontology to describe the services. OWL-S [16], which supersedes DAML-S [16], uses Web Ontology Language (OWL) [110]. OWL-S was created to replace DAML-S and has a more expressive description capability. Another such language is Web Service Modeling Ontology (WSMO) [89]. WSMO together with Web Service Modeling Framework (WSML) [54] and Web Service Execution Environment (WSMX) [94] presents a framework for Semantic Web services, combining Semantic Web and Web service technologies. Currently, WSMO and OWL-S are the most expressive description languages. However, OWL-S is more popular and widely-used since it provides a facility and convenience for discovery, composition, and invocation [53].

Figure 2.6: Top level of the service ontology

A Semantic Web service described by OWL-S includes a *Resource* which is a specification of the Resources used by the service including computational aspects such as bandwidth or disk space.

Web service discovery uses only *ServiceProfile* because it contains the description of the service to be discovered. The *ServiceProfile* represents the service through the class *Profile*, which has three basic types of information, namely, contact information, functional description of the service, and additional properties.

The contact information is meant for human users. It includes *serviceName* which is the name of the service, and *contactInformation* which shows the address, telephone number, etc. of the owner of the service. The *textDescription* provides a brief description of the service. It summarizes what the service offers, or describes what service is requested.

The functional description of the service is the most important declaration in the *Profile* that is used for matching. It specifies the following parameters: inputs, outputs, preconditions, and effects of the service. The inputs are parameters required by the service and the outputs are the parameters which are generated by the service. The precondition indicates the condition necessary before execution of the service and the effect is the status after execution of the service. For example, to invoke a Web service to buy a computer and pay via credit card, the input is the computer configuration, the output is price, the precondition is that the credit card must be valid, and the effect is that the credit card must be charged. Inputs and outputs point to ontological concepts.

Additional properties which describe features of the service are *serviceParameter*, *qualityRating*, *serviceCategory*, etc. The Profile also declares an operation of the Web service. The operation is declared by a *profileHierarchy* property which points to a concept in an ontology. Matching two operations of two Web services is one of the four stages of the matching algorithm. This stage is called *operation matching* which will be elaborated in chapter 6.


```

1  <?xml version="1.0"?>
2  <!DOCTYPE rdf:RDF [
3      <ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
4      <ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
5      <ENTITY owl "http://www.w3.org/2002/07/owl">
6      <ENTITY actor "http://localhost/Actor.owl">
7      <ENTITY profile "http://localhost/WSRequester.owl">
8      <ENTITY profileHierarchy "http://localhost/eBusiness.owl">
9      <ENTITY DEFAULT "http://localhost/WSRequester.owl">
10 ]>
11
12 <rdf:RDF xmlns="http://localhost/WSRequester.owl#"
13     xmlns:actor="http://localhost/Actor.owl#"
14     xmlns:owl="http://www.w3.org/2002/07/owl#"
15     xmlns:profile="http://localhost/WSRequester.owl#"
16     xmlns:profileHierarchy="http://localhost/eBusiness.owl#"
17     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
18     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
19     xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
20
21     <profileHierarchy:BuyComputer rdf:ID="BuyComputer">
22     <profile:serviceName> Buy Computer </profile:serviceName>
23     <profile:textDescription>
24     This is a service provides computer
25     </profile:textDescription>
26     <profile:hasInputrdf:resource="http://localhost/Tech.owl#Computer"/>
27     <profile:hasOutput rdf:resource="http://localhost/pri.owl#Lesthen100"/>
28     </profileHierarchy:BuyComputer>
29
30 </rdf:RDF>

```

Figure 2.7: The representation of a partial OWL-S service profile

Figure 2.7 shows the profile of a simple “Buy Computer” Semantic Web service described in OWL-S. This is a service by a requester wishing to buy a computer with a price which is less than 100 units of currency. Therefore, the service has input which is *Computer* concept from a *Technology.owl* ontology. The output of the service is *Lessthen100* concept which is from a *price.owl* ontology. The operation of the service is “*BuyComputer*”, declared via *profileHierarchy* by a *BuyComputer* concept which belongs to an *eBusiness.owl* ontology.

2.4 Summary

This chapter has described the technologies that form the foundation to this thesis, namely, the Semantic Web, Web services, and Semantic Web services. Among these, Semantic Web service, which combines the Semantic Web and Web service, is a core component in this work. An understanding of such technologies is needed not only to appreciate the evolution of Web services but to determine the choice of languages to utilise in the implementation of MOD. Chapter 3 presents a survey of Web service discovery systems which employ these techniques.

Chapter 3

Web Services Discovery: the State of the Art

This chapter contains a survey of approaches and open problems relating to Web service discovery systems. It first starts with a survey of existing Web service discovery systems in section 3.1. The advantages and disadvantages of each system will be highlighted. Arising from the survey, three open issues are introduced in section 3.2. An overview of approaches to resolve the three problems are also presented. Finally, the chapter is summarized in section 3.3.

3.1 A Survey of Web Services Discovery Systems

As many Web service discovery systems have been developed, a survey of the systems is needed to explore existing techniques and to highlight the advantages as well as disadvantages of each system. Garofalakis *et al.* [29] presented a survey on these systems. However, their work mainly focused on aspects and approaches of Web service architecture. In Semantic Web services, the usage of semantics is the most important factor. This section presents a survey of Web service discovery systems which mainly focuses on the use of semantics.

3.1.1 Categories and a Taxonomy of Web Service Discovery Systems

Current Web services discovery systems can be categorized into categories, namely: distributed systems, supporting QoS systems, and applying fuzzy logic to the systems.

Details of each category are given as follows:

- *Distributed Discovery Systems:* A large number of distributed Web services discovery systems based on Peer to Peer (P2P) technologies have been proposed to exploit the advantages of decentralized characteristics of a distributed system. Most of the systems built on P2P network use ontologies to publish and discover the Web services. Some typical systems are found in [28,57,116,118,119]. These systems present approaches for distributed Web services organization by combining the capabilities of Semantic Web services with the dynamics and real-time search capabilities of (P2P) networks. Typically, an ontology is used to organize services based on semantic classification of domains.
- *QoS Discovery Systems:* Web service discovery can be divided into two phases. The first phase is matching functional aspects of the Web service such as input, output, precondition, effect, operation and so on. The output of the first phase may result in many advertised Web service satisfying the requirement. To reduce the number of advertised Web service providers and to find the most appropriate services, the second phase uses QoS. QoS refers to *Quality of Service* which considers availability, accessibility, integrity, performance, reliability, regulation, and security. Several discovery systems supporting QoS have been developed

such as [27,38,61,91,122,123]. However, current Semantic Web services seldom support QoS because of its immaturity and complexity.

- *Applying Fuzzy Logic to Discovery*: Since the model-theoretic semantics of the languages used in the Semantic Web are crisp, the need to extend them to represent fuzzy data arises [11,40,64,83,100]. As ontology languages are based on Description Logics which lack the ability to encode and reason with imprecise knowledge, Stoilos *et al.* [100] extended the DL language with fuzzy set theory and provided sound and complete reasoning algorithms for the extended language. This work focused only on extending the description language to facilitate Semantic Web Service activities.

As mentioned, in Semantic Web services, the method of using semantics is the most important factor. Therefore, our survey of discovery systems focuses on the use of semantics in the systems.

Current discovery systems can be categorized into a taxonomy based on the use of semantics. Figure 3.1 presents a taxonomy of Web service discovery systems, which can be divided into systems for matching Semantic Web services and systems for matching non-Semantic Web services. The Semantic Web service discovery systems include systems that support matching Web services using different ontologies and systems that support matching Web services using the same ontology. There are three main approaches for matching Semantic Web services using the same ontology, namely, dividing the matching process into several stages, matching two profiles directly and supporting UDDI.

The system described by Liang *et al.* [59] is an example of a non-Semantic Web service discovery system. The authors introduced a system that can provide semi-automatic discovery and composition by indexing service advertisements. Currently, most systems use UDDI for discovering such services which are usually described in WSDL. Furthermore, these services are registered in UDDI, which is then used to locate service providers. The shortcoming of UDDI, which has been mentioned in chapter 1, is that it lacks semantic understanding. Semantic Web services overcome the mismatch of Web services. This is done by adding more information to the Web service descriptions.

Figure 3.1: Taxonomy of Web service discovery systems based on the use of semantics

3.1.2 Discovery of Web Services using the Same Ontology

Almost all Web service discovery systems assume the Web services are based on the same ontology. Some systems divide the matching into several stages; others match two service profiles directly, while some support UDDI.

3.1.2.1 Direct Matching

This category of Semantic Web services discovery systems matches two service description “profiles” directly instead of dividing the matching into several stages. As mentioned in chapter 2, the service profile consists of information about the services for matching purpose. In these systems, users are not permitted to manage the degree of similarity of matching. Direct matching systems are accurate but are time-consuming since all information of the profiles is matched. Furthermore, they do not allow users to manage the trade-off between performance and quality of matching.

InfoSleuth [3,37] is one of the earliest matching agent systems. The matching agent enables the querying agent to locate all available agents that provide appropriate services. It supports syntactic and semantic matchmaking. In InfoSleuth, the service capability information is written in LDL++ [39,95], a logic deduction language. Agents use a set of LDL++ deductive rules to support inferences about whether an expression of requirements matches a set of advertised capabilities. The system provides syntactic and semantic matching. However, LDL++ is outdated and limited in its expressivity. LDL++ is not a standard supported by W3C. When InfoSleuth was developed, OWL-S and DAML-S which are currently widely used as Web services description languages were unavailable.

Similarly, Trastour *et al.* [105] did not use DAML-S or OWL-S for describing Web services because their work was carried out before the existence of these languages. Therefore, they developed their own specification for Web services description by converting services into an RDF graph. Services are nodes of the graph. Two nodes are considered matched if one node is a subtype of the other. Trastour *et al.* extended this work by using DAML+OIL as the Web description language [10]. The matching is based on the subsumption relationship to find matches between concepts. DL (Description Logic) reasoner is used to determine the similarity between concepts. The system has the drawback that it only considers the relationship between direct sub-concepts and super-concepts. It does not consider the relationship between concepts that are more distantly related.

Benatallah *et al.* [4] presented an approach for Web service discovery based on Description Logic. They formalized service discovery based on rewriting concepts using terminologies. This is called the *best covering problem*. They presented a formalization of the *best covering problem* in the framework of DL-based ontologies and proposed a hypergraph-based algorithm to effectively compute best covers of a given request. In this system, matchmaking is based on DAML-S. Similarly, Li and Horrocks [56] developed a framework for matching Web service based on DAML-S. Their Matchmaker uses Racer [34] as DL (Description Logic) reasoner to determine the semantic match between services. It tries to match two profiles of the requested service and advertised service, instead of dividing the matching into several stages. Without dividing the matching into several stages, it will be difficult for these systems to automate reasoning techniques to compute semantic matching and the matching will be inefficient.

Web Service Modeling Ontology (WSMO) provides a conceptual model for service discovery that exploits WSMO formal descriptions of goals and Web services. WSMO Discovery Engine [88] can match WSMO goals, and hence, matches a requester Web service's description against provider Web service's description. WSMO has two major advantages. Firstly, Web services can be expressed in rich semantics. Secondly, the system proposes a distributed discovery framework. Since the semantic reasoning is computationally resources intensive, the distributed execution may improve system performance especially if there are a large number of Web services. However, WSMO is not a standard Web services. Compared with OWL-S, the latter is more mature in some aspects, such as the definition of the process model and the grounding of Web services. Therefore, WSMO is not as popular as OWL-S, which is widely supported by the Web community [53].

3.1.2.2 Dividing the Matching Process into Several Stages

The challenge of dynamic matchmaking in the Internet is to optimize the trade-off between performance and quality of matching. Complex matching has to be restricted to allow meaningful semantic matches of requests and advertisements in a reasonable time. Direct matching systems are more accurate than the stage-dividing matching systems but they are time-consuming. They do not allow users to manage the trade-off between performance and quality of matching. To overcome this, some systems divide the matching process into several stages. These include TU-Berlin [62], LARKS [101-103], the Matchmaker from the collaboration between Toshiba and Carnegie Mellon University [47,48], and a matchmaker proposed by Paolucci *et al.* [76].

As heterogeneous Web services use different description languages to describe the services and do not understand each other across distributed networks, the LARKS (Language for Advertisement and Request for Knowledge Sharing) project [101-103] defined an expressive common language called ACDL (Agent Capability Description Language). The LARKS matching agent uses this language to match Web services requesters with Web service providers. It includes five different filters, namely, context matching, word frequency profile comparison, similarity matching, signature matching, and constraint matching. LARKS is expressive and capable of supporting inferences. Its knowledge is specified as local ontologies in the concept language ITL. It supports matching syntactic and semantic similarity among Web services by using techniques from information retrieval and AI. It also allows users to restrict the result of matching. However, all services in LARKS must be described in ACDL which is not a standard and is quite difficult to write. Furthermore, LARKS does not support other commonly used Web service description languages such as DAML-S and OWL-S.

Paolucci *et al.* [76] have developed an algorithm for matching Semantic Web services using DAML-S. Their work considers only input and output matching. Input and output are concepts based on an ontology. Therefore, the similarity of input, output matching is treated as the similarity of concepts from the same ontology. Paolucci *et al.* defined four levels of similarity: *Exact*, *Plug In*, *Subsumes*, and *Fail*. This definition is based on concepts that have a direct relationship either as: sub-concept or super-concept. Matching two concepts should also include the properties of both concepts. It should consider situations when these concepts are not related directly. It also should consider syntactic matching which is based on service name and description of the service.

The Semantic Web service matching system from the collaboration between Toshiba and Carnegie Mellon University [47,48] is based on LARKS and the algorithm from [76]. To overcome the drawback that requires ADCL, the system supports Semantic Web services which are described in RDFS, DAML-S, and OWL-S. Similar to LARKS, the system provides a set of filters and allows users to configure these filters to achieve the desired tradeoff between performance and matching quality. There are four filters which are independent of each other. These are Namespace Filter, Text Filter, I/O Type Filter, and Constraint Filter. The system can be installed as an add-on to UDDI; therefore, it supports matching of both Semantic Web services and non-Semantic Web services. When it receives a request for a service, it performs the matching process. If it receives a non-Semantic Web service request, it redirects the request to UDDI to perform the matching process.

The Matchmaker from TU-Berlin [62] is a good example of a matching algorithm which is divided into four stages. The four stages are input matching, output matching, profile matching, and user-defined matching. By dividing the matching algorithm, the Matchmaker not only avoids the drawback in the direct matching systems described in section 3.1.2.1 but also helps users to choose the degree of matching from each stage in order to control the number of results and accuracy. An earlier version supported matching DAML-S Semantic Web services, whereas the current version supports matching OWL-S Semantic Web services. The user-defined matching stage allows users to describe more constraints to restrict the number of results and to increase the accuracy of the results. However, the system is too simplistic. It only presents a demonstration of matching two Web services; it is not really a discovery system since it does not store the

advertisement into a database. It is also very difficult for users to utilize user-defined matching as users must define their rules and constraints without user-friendly graphical user-interfaces.

In summary, some systems divide the matching process into several stages that allow users to manage the trade-off between performance and quality of matching by turning on or off the matching stages. These include Paolucci [76], LARKS [101-103], Toshiba-CMU [47,48], and TUB [62]. The term “several stages” can only be quantified based on a particular work. We can categorise the work into a number of stages: 2-stage (Paulucci), 3-stage (LARKS, Toshiba-CMU), and 4-stage (TUB, MOD). Table 3.1 is a comparison between these work based on the number of stages.

Table 3.1: A comparison between the related works based on the stages

Project	Stages				Approach	Language Supported
	Input	Output	Operation	User-Defined		
Paolucci	√	√	-	-	Based on <i>subsumption</i> relationship	DAML-S
LARKS	√	√	√	-	Based on <i>subsumption</i> relationship	ACDL
Toshiba-CMU	√	√	√	-	Based on <i>subsumption</i> relationship	DAML-S
TUB	√	√	√	√	Based on <i>subsumption</i> relationship	OWL-S
MOD	√	√	√	√	Support <i>different</i> ontologies	OWL-S

The maximum stages that a system could have are input, output, operation, and user-defined matching. The minimum stages that a system must have are input and output matching since the two stages match the most important and fundamental information of the Web services. For each matching stage, different systems may have different algorithms/approaches. In particular:

- Paolucci: based on *subsumption* relationship. The matching is classified into: *Exact*, *Plug In*, *Subsumes*, and *Fail*. The project supports Web services described by DAML-S using the same ontology.
- LARKS: The matching is classified into: *Exact*, *Plug-in*, and *Relaxed*. Its ontologies are described by Information Terminological Language (ITL) and its Web services are described by ACDL (Agent Capability Description Language) which is not a well supported language.
- Toshiba-CMU: similar to LARKS but it has improved on LARKS by supporting DAML-S.
- TUB: supports 4 stages. The matching is similar to Paolucci but it supports OWL-S based on the same ontology.
- MOD: supports 4 stages. Web services are described by OWL-S. It is different from the above systems as it supports references to the same ontology as well as different ontologies.

3.1.2.3 Supporting UDDI

Using UDDI for discovery has limitations. UDDI is XML-based and since XML lacks semantics, the discovery mechanism is limited. To overcome this problem, attempts have been made to add semantics to UDDI. The project from Carnegie Mellon [98] is a typical

example. It has added OWL-S profile to UDDI. Kawamura *et al.* [47,48] added advertiser's Web services in WSDL to UDDI and advertiser's DAML-S profile to its Matchmaker. If the requested Web services use WSDL, the discovery system will access UDDI to look for suitable service providers. Otherwise, the discovery system will revert to the Matchmaker. By combining the Matchmaker and UDDI, the system can support matching both Semantic and non-Semantic Web services, but in a separate manner. In other words, the two types of Web services are not considered as alternatives. Matches are to either Semantic or non-Semantic Web services. Another example is from [97] which has added semantics to WSDL using DAML+OIL ontologies. It also uses UDDI to store these semantics and searches for Web services based on these semantics. By adding semantics to WSDL and UDDI, the project in [97] achieves sufficient expressiveness to automate the discovery process.

The registration and discovery with UDDI is based on a centralized design whereas Web service providers developed their own stores called private registries. As Web services support distributed operation execution, this has led to a demand for a combined UDDI registry with the private registries. Thader *et al.* [107] have developed a distributed peer-to-peer infrastructure. They created a virtual global registry by connecting all private registries in a P2P network. Furthermore, they also used DAML-S to enhance semantic search capabilities.

3.1.2.4 Comparing the three Sub Categories

The advantages and disadvantages of the three methods that use the same ontology are summarized in table 3.2:

Table 3.2: Advantages and disadvantages of three approaches to Semantic Web discovery
based on the same ontology

Systems	Advantages	Disadvantages
• Direct matching	+ Matches all information of the two Web services	- Time consuming to match all information of the two Web services. - Does not support user-definition in each part of matching (e.g. input, output matching, etc)
• Divide the matching process into several stages	+ Can specify the degree of the similarity + Can define more constraints to enhance the result.	- Some information of Web service may not be matched; therefore, the accuracy may be lower than direct matching.
• Supporting UDDI	+ Can support both semantic matching and non-semantic matching	- Incorporating semantics to UDDI but not to a full extent.

3.1.3 Discovery of Web Services using Different Ontologies

The above mentioned Web service discovery systems only support matching Web services requesters and Web services providers who use the same ontology. This assumption implies that if different ontologies are used, matching cannot be carried out. This is a major limitation since Web services are heterogeneous, autonomous, and developed independently. It is necessary to discover Web services that are based on different ontologies.

Cardoso and Sheth in “Semantic e-Workflow Composition” [8] have addressed this problem. The system supports syntactic matching, quality of service (QoS) matching, and semantic matching. Syntactic matching involves computing the syntactic similarity of Web services requesters and Web services providers based on their service names and service descriptions. They use “string-matching” as a way to calculate how closely service names and service descriptions of the two Semantic Web services resemble each others. QoS is used to match Web services requesters and Web services providers based on three factors, namely, time, cost and reliability. Semantic matching is computed based on concepts and their properties. Property matching is based on domain, name, and range of properties.

However, the system is part of a work-flow project [8]. Therefore, this discovery system has been developed mainly to support the project and is limited as a discovery system. It supports user defined degree of similarity but it does not allow users to declare more rules or constraints to restrict the matching result. Semantic matching based on only concepts and their properties is insufficient. It must also include the sub-concept, super-concept and properties of these concepts. The domain of concepts also should be considered. Moreover, it is unclear if the system determines whether two ontologies are the same or different.

Oundhankar *et al.* [75] presented an extension of the algorithm in [8] with two new measures: context and coverage similarity. The discovery technique is based on METEOR-S [77] which is a Web service discovery infrastructure. This infrastructure provides a facility to access registries that are divided into business domains and grouped into federations. The discovery algorithm used input, output of Web services for

matching. The matching is divided into *syntactic* and *semantic* matching. The syntactic matching is similar to Cardoso's work which uses n-gram [2,93,121]. But in semantic matching, the algorithm improved Cardoso's work by considering the context and the coverage information of concepts.

By considering the concept and the coverage information of the matched concepts, the algorithm is able to find good matches and eliminate false matches. Moreover, based on METEOR-S, indexing and retrieving information in the registries is facilitated. However, the system does not allow users to intervene during the matching process. It also does not allow users to define more rules or constraints to restrict the matching result. This may be a significant problem since the number of Web services from the matching result is huge. Furthermore, the approach to measure coverage similarity is incomplete since it is only based on the similarity and disjointness relationships.

3.2 Web Services Discovery Problems and Approaches

This section introduces three Web service discovery problems. They are: matching Semantic Web services using different ontologies, matching Semantic Web services against non-Semantic Web services, and matching Semantic Web services using different description languages.

3.2.1 Matching Web Services Using Different Ontologies

As mentioned in chapter 1, current discovery systems are adequate when the Web service requester and provider use the same ontology to determine how the provider satisfies the

requester's requirement. Unfortunately, most of them do not support the situation where a Web service requester and provider use different ontologies. Only one group from the University of Georgia [8,75] has addressed this problem. However, there are shortcomings which were mentioned in section 3.1.3. These drawbacks should be addressed. In the real world where the Web service requester and provider operate independently, each defines their own ontologies to describe their services. However, a Web service provider can provide an exact service to the requester even though both services use different ontologies. Therefore, a discovery system that supports Web services using distinct ontologies is necessary.

Since this urgent research issue has yet to be addressed, this thesis focuses on solving this problem. The algorithm and engine proposed to address this problem is presented in chapter 6.

3.2.2 Matching Semantic Web Services against Non-Semantic Web Services

As mentioned in earlier sections, UDDI and some discovery systems have been developed for matching non-Semantic Web services. They support searching by service names and service categories. They also support matching by inputs and outputs of the Web services but focus only on syntactic aspects. The development of Semantic Web services has led to discovery mechanisms for Semantic Web services. Some Semantic Web services discovery systems have been developed to meet this need. However, these systems only search for Semantic Web services, and not for non-Semantic Web services.

In short, we lack systems which can support matching a Semantic Web service requester with non-Semantic Web service provider and vice versa.

For example, a company A developed a Web service to sell computers online. At that time, Semantic Web service technology was not available. Hence, WSDL was employed to describe the service. Another company B, which is interested in purchasing computers, started recently. It developed an ontology in OWL and uses OWL-S to describe their request service. Company A is able to provide the service to company B. However, with the current Web service discovery, company A cannot be found to match the requirement of the company B because they use different Web service discovery languages: company A uses a non-Semantic Web services description language while company B uses Semantic Web service description language.

The inputs, outputs, and operation of the Semantic Web services are found in the *ServiceProfile* (section 2.3, chapter 2) while those of the non-Semantic Web services are in the messages of WSDL. On the other hand, the operations (methods) of the service are in the profile of the Semantic Web services. The operations of the non-Semantic Web services are described in WSDL. To perform matching, we propose the matching of inputs, outputs, and operations of Semantic Web services and non-Semantic Web services. As textual data is used in the two types of Web services, text matching is used. This is elaborated as follows:

1. Inputs and outputs of the Semantic Web services profile are matched against input and output parameters of the WSDL messages, respectively.

2. Operations of Semantic Web services profile are matched against operations in WSDL.
3. Matching two Web service description texts.

Matching could be divided into two levels: syntax matching and semantic matching. Syntax matching includes matching the names (labels) of inputs, outputs, operations of Semantic Web services against non-Semantic Web services and matching two Web service description texts. This matching could use WordNet and Jaro technologies (section 5.4.1, chapter 5).

Semantic Web services use ontologies to describe the service. Therefore, at the semantic level, we match the WSDL elements with not only the inputs, outputs, operations concepts but also with super/sub-concepts of the concepts. This could help determine the relationship between the WSDL elements through the semantic similarity of concepts and WSDL elements.

3.2.3 Matching Semantic Web Services that Use Different Description Languages

Semantic Web services have been developed increasingly to contribute to e-business and e-commerce applications. There are a large number of Semantic Web services based on different description languages, including DAML-S, OWL-S, WSMO, etc. Existing Semantic Web services discovery mechanisms support matching based on specific description languages but they cannot perform matching on different description languages.

Suppose a Web service uses OWL-S while another service uses WSMO. OWL-S uses OWLJessKB reasoner [31] while WSMO uses WSMO reasoner [89]. The service provider which is described by OWL-S can meet the requirements of a Web service requester which is described by WSMO. However, these two services cannot be matched using current Semantic Web services discovery systems because the two different reasoners do not interact with each other.

To solve this problem, the reasoners must interoperate. This is difficult because each reasoner supports one ontology description language and they have different vocabularies. A possible approach is to include multi-reasoners where each reasoner is responsible for an ontology description language. Then we could use the technique to be described in chapter 5 to compute the concept similarity.

3.3 Summary

The chapter has presented a survey of Web services discovery systems which focus on the use of semantics in the systems. A taxonomy of discovery systems was introduced. The advantages and disadvantages of various systems were highlighted. There are two categories of the discovery systems that cater to Semantic Web services and non-Semantic Web services, respectively. UDDI is a typical registry and discovery system for non-Semantic Web services. For Semantic Web services, the systems can be categorized as systems that support matching based on the same ontology or on different ontologies. In systems supporting the same ontology, we can further classify them as systems that

divide the matching process into several stages, directly matching two Web service profiles, or supporting UDDI.

We have introduced three open problems arising from current Web service discovery systems, namely, matching Semantic Web services against non-Semantic Web services, matching Semantic Web services using different description languages, and matching Semantic Web services using distinct ontologies. A new Web services discovery mechanism is required to overcome the three shortcomings. When designing a Web service discovery system, the main problem of dynamic matchmaking in the Internet, namely, the trade-off between performance and quality of matching should be borne in mind. The heterogeneities of Web services should also be considered.

Amongst the above problems, supporting matching Web services using different ontologies is the most important and urgent. This is because more and more Web services have been developed based on different ontologies. Moreover, when addressing this issue, several novel algorithms of ontology operation are required to be tackled. These algorithms are not only used in MOD but also contribute to ontology functionalities in general. Therefore, we focus on the problem of matching Web services using different ontologies, and the remaining issues are left for future consideration.

Chapter 4

Comparing Two Ontologies

4.1 Introduction

There are a large number of ontologies in the Internet that have been developed. Such development is both tedious and expensive. Hence, once ontologies are developed, they can be shared by many companies. A created ontology may be very large. It is unlikely that many applications use the entire ontology. The following three situations may arise. The entire ontology is used; only a part of the ontology is used; or a part of the ontology is extended to become a different ontology by adding more knowledge. Therefore, arising from an original ontology, there may be many replicas as well as variations deployed in different locations (URL) in the Internet. In addition, these locations may change frequently. In these situations, ontologies which are used by different companies are related.

The relationship variations that may arise from an ontology are summarized in figure 4.1. In this figure, each node represents a concept; each arrow represents a relationship. To simplify the figure, we do not show properties and other information. Furthermore, concepts and relationships form the skeleton of an ontology. Figure 4.1a depicts an original ontology. Case 1 (figure 4.1b) shows the entire ontology used in a different location in the Internet and so the two ontologies are exactly the same. Case 2 (figure

4.1c) illustrates only a part of the original ontology. In this case, the ontology is a subset of the original one. Lastly, case 3 (figure 4.1d) shows a part of the original ontology which has been extended with additional knowledge. As a consequence, the modified ontology contains some common concepts and relationships with the original one.

Figure 4.1: Cases of using ontology

In ontology operations such as ontology mining, mapping, aligning, and integrating, it is necessary to determine the relationship of two ontologies. For instance, in ontology classification within ontology mining, we may need to classify ontologies in predefined

clusters such as: *identical*, *sub ontology*, *intersection*, or *different*. In a Web service discovery system, determining the semantic similarity of concepts from ontologies is a key function in the system. Currently, the semantic similarity of concepts from different ontologies is measured based on information such as labels, properties, and neighborhood concepts of the concepts. However, such information is insufficient. For instance, if the two ontologies are related and the two concepts belong to a part which is common to both ontologies, then the concepts can be considered as belonging to the same ontology. Computing concept similarity in the same ontology is more accurate than in different ontologies. More details of this issue will be presented in chapter 5.

Checking the relationship between two ontologies bears some similarities to ontology versioning and ontology alignment. However, ontology versioning focuses mainly on the changes of ontology with time. In other words, it considers cases when ontologies are changed to adapt to new situations. Such changes are typically minor and incremental. It may involve a change in concept, relationship, domain, or data type. Ontology versioning checks the delta amendments of one ontology compared to another. It does not focus on determining common ontologies and does not check if one ontology is a subset of the other ontology. Ontology alignment is the process of determining semantic correspondences between concepts. Its purpose is finding a *map* between two different ontologies without the presence of an original ontology. Our assumption is that one ontology evolved from another and the two ontologies may involve significant differences. More details of related work are presented in section 4.4.

This chapter presents an algorithm to check whether two ontologies are related. If the ontologies are related, the algorithm will return the parts that are in common. The chapter

started with the motivation for checking the relationship between two ontologies. The ontology comparison algorithm is introduced in section 4.2. To illustrate and highlight the validity of the algorithm, examples and an experiment to compare two ontologies are presented in section 4.3. Related work is presented in section 4.4 and the chapter concludes in section 4.5.

4.2 Ontology Comparison Algorithm

4.2.1 Ontology Relationship Definitions

An ontology contains a wide range of information. In this chapter, only concepts, relationships, and properties are involved. Ontological instances are ignored since we focus on matching the structure of two ontologies. Relationships and properties of concepts in an ontology are represented in set $\{R, P\}$, where R represents a set of relationships and P represents a set of properties. Each relationship $r_i(C_p, C_q)$ in R represents a binary association between concepts C_p and C_q . As a consequence, concepts in the ontology are captured in R . $P_j^{C_i}$ which is an element of P represents the j^{th} property of concept C_i . Each concept C_i may have several properties (or none).

Assume we have two ontologies O and O' with components $\{R, P\}$ and $\{R', P'\}$, respectively and we need to compute the relationship between O and O' . When comparing two ontologies, one of the following situations may occur: both ontologies are identical; one ontology is a sub ontology of the other; two ontologies have some common parts; the two ontologies are different. An elaboration of these relationships follows:

- *Identical ontology.* The two ontologies are identical if all concepts, their relationships, and their properties are the same. It means that every element $r_i \in R, p_j \in P$ in ontology O must have a corresponding $r_i \in R', p_j \in P'$ in ontology O' and vice versa.

$$\{R, P\} \subseteq O, \{R', P'\} \subseteq O', O \text{ and } O' \text{ are the same}$$

$$\Leftrightarrow \begin{cases} \forall r_i \in R, p_j \in P \Rightarrow r_i \in R', p_j \in P' \\ \forall r'_i \in R', p'_j \in P' \Rightarrow r'_i \in R, p'_j \in P \end{cases}$$

- *Sub ontology.* Ontology O is a complete subset of ontology O' if all concepts, their relationships and properties of ontology O can be found in ontology O' but not vice versa.

$$\{R, P\} \subseteq O, \{R', P'\} \subseteq O', O \text{ is sub ontology of } O'$$

$$\Leftrightarrow \forall r_i \in R, p_j \in P \Rightarrow r_i \in R', p_j \in P'$$

- *Different ontologies.* Two ontologies are different if they do not have any common part. This requirement is satisfied if the two ontologies do not have any common relationship between two concepts.

$$\{R, P\} \subseteq O, \{R', P'\} \subseteq O', O \text{ and } O' \text{ are different}$$

$$\Leftrightarrow \begin{cases} \forall r_i \in R \Rightarrow \neg \exists r'_i \in R' \\ \forall r'_i \in R' \Rightarrow \neg \exists r_i \in R \end{cases}$$

- *Intersection.* Two ontologies may have one or more common parts. A common part must have at least two concepts and a relationship between the concepts.

$$\{P, R\} \subseteq O, \{P', R'\} \subseteq O'$$

$$\Leftrightarrow \begin{cases} \exists r_i \in R, \text{ and } \exists r_j \in R' \\ r_i = r_j \end{cases}$$

- *Ontology similarity.* Some applications need to measure the percentage similarity of two given ontologies. The percentage similarity of two ontologies is computed based on the number of common concepts, their properties, and their relationships. The formula to compute the similarity of the two ontologies is as follows:

$$\text{ontSim}(O, O') = \frac{|R \cap R'| + |P \cap P'|}{|R \cup R'| + |P \cup P'|}$$

4.2.2 Ontology Comparison Algorithm Description

Figure 4.2 presents a declaration of various data structures used in the algorithm. *interOnt* is an array containing common ontologies. *concept1* and *concept2* are vectors to store all the concepts of ontology 1 and ontology 2, respectively. *intersect* is the intersection of the two vectors of concepts.

```
array interOnt of common ontologies;
vector concept1 = get all concept of ontology 1 ;
vector concept2 = get all concept of ontology 2 ;
// 'intersect' stores common concepts of the two ontologies. It is archived by comparing
// each concept in ont1 with concept in ont2.
vector intersect = get all common concepts from two vectors, namely, concept1 and concept2 ;
```

Figure 4.2: Data structure declaration

The algorithm functions as follows. It first determines all the concepts found in the two ontologies and stores them in two vectors *concept1*, and *concept2*, respectively. Then, the two vectors are compared to determine the intersection concepts, namely, *intersect*.

Based on the *intersect* vector and the *isPropertiesMatch* function, we will determine the relationship between the two ontologies. In case, if the two concepts are matched by name, the *isPropertiesMatch* function will check if their properties are matched. If they are, the two concepts are identical. If the two ontologies have their entire concept pairs are identical, then the two ontologies are identical. The similar reasoning for the case sub ontology and different ontology relationship.

In case, if the two ontologies have common part, the algorithm will iterate through each *concept* in the *intersect* vector. Each *concept* is checked to determine if it has a relationship with classes in the array of common ontologies *interOnt* by using the function *isRelation*. If it has a relationship with any class in *interOnt*, it will be added to that ontology by using the *addConcept* function. If the *concept* is related to more than one common ontology, these related ontologies will be merged to become one common ontology by using the *mergingCommonOntology* function. If the *concept* does not have relationship with any class in common ontologies, the new common ontology of *interOnt* is created and the *concept* is added to the new common ontology by using the *createOntology* function. The process is repeated for all *concepts* in *intersect*. As a consequence, the *interOnt* is created to store common ontologies. The algorithm is summarized in pseudo-code as follows:

```
FUNCTION getOntRelationship(ontology ont1, ontology ont2):
    Flag = TRUE; // Flag is used to determine if the properties of two
                  //concepts are matched
    interOnt[0] = NULL; //Vector stores common ontologies

    // Case 1: two ontologies are identical
    IF (concept 1 == intersect) AND (concept 2 == intersect) THEN
        //Checking if the properties of two concepts are identical
        FOR (i = 1 to concept1.size()) DO
            FOR (j = 1 to concept2.size()) DO
                IF (concept1[i]== concept2[j]) THEN
```

```

        Flag = isPropertiesMatch(concept1(i), concept2(j));
    END IF
END FOR
END FOR
IF (Flag == TRUE) THEN
    PRINT    "The two ontologies are identical";
    RETURN   interOnt[0] = Ont1;
END IF
END IF

//Case 2: ontology 1 is a sub ontology of ontology 2
IF (concept 1 == intersect) THEN
    //Checking if the properties of the two concepts are identical
    FOR (i = 1 to concept1.size()) DO
        FOR (j = 1 to concept2.size()) DO
            IF (concept1[i]== concept2[j]) THEN
                Flag = isPropertiesMatch(concept1(i), concept2(j))
            END IF
        END FOR
    END FOR
    IF (Flag == TRUE) THEN
        PRINT    "Ontology 1 is a sub ontology of ontology 2";
        RETURN   interOnt[0] = Ont1;
    END IF
END IF

//Case 3: ontology 2 is a sub ontology of ontology 1
IF (concept 2 == intersect) THEN
    //Checking if the properties of the two concepts are identical
    FOR (i = 1 to concept2.size()) DO
        FOR (j = 1 to concept1.size()) DO
            IF (concept2[i]== concept1[j]) THEN
                Flag = isPropertiesMatch(concept2(i), concept1(j))
            END IF
        END FOR
    END FOR
    IF (Flag == TRUE) THEN
        PRINT    "Ontology 2 is a sub ontology of ontology 1";
        RETURN   interOnt[0] = Ont2;
    END IF
END IF

// Case 4: the two ontologies are different
IF (intersect.size = 0) THEN
    PRINT    "The two ontologies are different";
    RETURN   interOnt = NULL;
END IF

// Case 5: the two ontologies have common parts
ELSE
    i=0;
    WHILE (intersect.size <> 0) DO
        commonConcept = get an element from intersect;
        intersect.delete(commonConcept);
        j=0;Flag=FALSE;
        WHILE (j ≤ interOnt.size) DO
            IF(isRelation(commonConcept, interOnt[j]))DO
                addConcept(commonConcept, interOnt[j]);
                mergingCommonOntology(interOnt[i]);
                Flag=TRUE;
            END IF
            j = j + 1;
        END WHILE
        IF (Flag=FALSE) THEN

```

```

        i = i + 1;
        createOntology(relationship, interOnt[i]);
    END IF
END WHILE
PRINT      "The two ontologies have common parts";
RETURN     interOnt;
END IF

END FUNCTION

```

4.2.3 Complexity of the Algorithm

We assume the two ontologies have a size m and n , respectively. Assume the maximum number of properties in concepts of the two ontologies is p and q , respectively. The function *getOntRelationship* which is presented above is suitable to demonstrate how the algorithm works but it is difficult to determine the time complexity of the algorithm. Therefore, the algorithm is re-written as follows without changing the gist of the solution in order to measure the time complexity.

```

FUNCTION getOntRelationship (ontology 1, ontology 2)
    FOR i = 1 to m DO// m times for m concepts of ontology 1;
        p := number properties of concept  $i^{\text{th}}$ ;
        FOR j = 1 to n DO// n times for n concepts of ontology 2;
            q := number properties of concept  $j^{\text{th}}$ ;
            FOR x = 1 to p DO// p times;
                FOR y = 1 to q DO// q times;
                    Compare (string1, string2) // O(max(k,z)); k and z are
                                                //the size of the string1 and
                                                //string2, respectively
                END FOR
            END FOR
        END FOR
    END FOR
END FUNCTION

```

The function Compare (string1, string2) which checks if the two strings are identical has a complexity of $O(\max(k, z))$ where k and z are the sizes of the two strings. We assume

that $k > z$ and therefore, $O(\max(k, z)) = O(k)$. As a result, the complexity of the algorithm will be $(m * n * p * q * k)$.

Normally, the complexity of an algorithm reflects the time performance of the algorithm. However, it should be noted that the algorithm depends much on external factors such as the size of the ontologies, the network speed, and power of the server in which the ontologies are stored.

4.3 Prototype, Example, and Experiment

Figure 4.3: Prototype of the ontology comparison algorithm

This section introduces the prototype based on the algorithm. This is followed by examples describing how the proposed algorithm is applied. The examples provide a detailed description of the ontologies and how the comparison is carried out. Lastly, testing is undertaken with one hundred ontological concept pairs. Prototype based on the Algorithm

Figure 4.3 shows a screenshot of the algorithm prototype. The prototype provides a facility for users to perform an ontology comparison and to trace the results. It displays detailed results of the comparison algorithm, including the relationship, percentage of similarity, and the first common ontology in *interOnt*. The next common ontology (if available) can be displayed by clicking on the “Next Common Ontology >>” button. The common ontologies, if any, are displayed as a tree.

4.3.1 Ontology Comparison Examples

Consider the following example. A company in Singapore developed an ontology in the *Technology* domain for its particular application. The *Technology* relates the practical application of science to commerce or industry. It includes *Electronic Technology*, *Automotive Technology*, *Chemical Technology*, etc. The former includes *Computer*, *Hardware*, and *Software* technology. The company adopted portions of *Electronic Equipment* from the *Equipment* ontology (section 2.1.2) to become sub ontology portions of *Electronic Technology*. The three portions are *Computer*, *Software*, and *Hardware*. Hence, the two ontologies which are located in different URLs have common ontologies. This situation is depicted in case 3 of figure 4.1. *Technology* and *Equipment* ontology are represented in figure 4.4a and figure 4.4b, respectively. As another example, a company

in Vietnam needs an ontology in a *Computer* domain for its specific purpose. Again, the *Equipment* ontology (section 2.1.2) is adopted. In this case, the *Computer* portion becomes this company's *Computer* ontology. This situation is depicted in case 2 of figure 4.1. *Computer* ontology is represented in figure 4.4c. In figure 4.4, only taxonomies of the ontologies are represented. The comparison of the two ontologies is as follows:

- concept1 = {Equipment, Electronic_Equipment, Auxiliary_Equipment, Computer, Analog_Computer, Personal_Computer, Desktop_Computer, Portable_Computer, Laptop, Notebook, Server, Hardware, CPU, AMD, Intel, Intel_Core, Intel_Pentium, Input_Device, Joystick, Keyboard, Microphone, Mouse, Webcam, Mother_Board, Output_Device, Audio, Headset, Speaker, Monitor, Printer, Storage, Software, Sound_System, Naval_Equipment, Sports_Equipment}.
- concept2 = {Computer, Analog_Computer, Client, Digital_Computer, Personal_Computer, Desktop_Computer, Portable_Computer, Laptop, Notebook, Server}.

Based on the two lists of concepts, the 'intersect' which is a list of common concepts is as follows:

- intersect = {Computer, Analog_Computer, Client, Digital_Computer, Personal_Computer, Desktop_Computer, Portable_Computer, Laptop, Notebook, Server}.

a) Technology ontology

b) Equipment ontology

c) Computer ontology

Figure 4.4: Technology, Equipment, and Computer ontologies using Protégé

- Common part between Equipment and Technology ontologies
- Common part between Equipment and Computer ontologies

Figure 4.5: Ontology common parts

Figure 4.5 shows the resultant common ontologies arising from the comparison algorithm. A comparison between *Equipment* and *Technology* ontology reveals that there are three common portions whose roots are *Computer*, *Hardware*, and *Software*, respectively. Figure 4.5a shows these common portions in subparts (i), (ii), and (iii). Figure 4.5b is the common part between *Equipment* and *Computer* ontologies. In this case, only one common portion of the ontology is found.

Another example is based on real world ontologies. *Creative Commons* is a company which created an *agent* ontology to describe its agent for an application. Then, for another application, they created another agent ontology called *fipa-agent*. To create this ontology, the company has adopted parts of the original agent ontology and added some other concepts such as *FIPAAgent*, *FIPA_Platform*, etc. Hence, the interaction between the two applications will be more accurately depicted if common ontologies are defined. Basing on the original agent ontology, other ontologies such as action, role, location, etc. have been developed. *Agent* ontology and the other related ontologies are available at: <http://daml.umbc.edu/ontologies/cobra/0.4/>. Figure 4.6 presents *agent* ontology and *fipa-agent* ontology in figure 4.6a and 4.6c respectively. The ontology comparison algorithm described in this chapter has determined the common ontology between *agent* and *fifa-agent* as shown in figure 4.6b.

Fig. 4.6a: Agent ontology

Fig. 4.6b: Common ontology

Fig 4.6c: Fipa_Agent ontology

Figure 4.6: agent ontology, common ontology, and fipa-agent ontology using the

SWOOP Editor [44]

4.3.2 Experiment and Results

Testing has been carried out on one hundred pairs of ontologies. Among them are 30 pairs in the *Equipment* domain, 30 pairs in *Technology* domain, 10 pairs between *Equipment* and *Technology* domains, 7 pairs in the *e-Business* domain, 3 pairs in the *Price* domain, and 20 pairs real world ontologies. For each domain, we first created an ontology as a “template”. The *Equipment* and *Technology* templates have been mentioned in the previous section. The *e-Business* and the *Price* templates can be found in Appendix A. Subsequently, based on the *Equipment* “template”, 10 ontologies were created by deleting, adding, or modifying classes, properties, and relationships. 10 ontologies in *Technology* domain were created in the same manner. Similarly, based on the *e-Business* and *Price* templates, other 5 ontologies were created. Real world ontology templates are *agent*, *computer*, *construction*, *context aware*, *bibliographic*, and *academia*. Based on these ontologies, many other variations have been developed.

Table 4.1 lists partial test results of 10 ontology pairs. The remaining pairs are given in Appendix B. As the ontologies were created manually from the original templates, the expected results are known. The results of the testing with 100 pairs confirm the accuracy of the algorithm. Index (test) 4 and 5 shows percentage similarities of 68.75 % and 70.96 %, respectively. This implies that ontology “Equipsub01” is more similar to ontology “Technology” than ontology “Equipment”. It is clear that the higher the percentage similarity, the more similar the two ontologies. In measuring the concept similarity, one is not concerned about the “percentage similarity” since the system is only based on the common ontology. We only need to know if the two concepts that we wish to measure the similarity belong to a common ontology.

Table 4.1: Partial results of comparing 10 ontology pairs

Index	Ontology 1	Ontology 2	Relationship	Percentage Similarity
1	Equipment	Equipsub01	Sub Ontology (1 Common)	96.15%
2	Equipment	EquipSame	The Same (1 Common)	100%
3	Equipment	Equipsub09	Sub Ontology (1 Common)	26.92%
4	TechSub04	TechSub01	Different Onts (0 Common)	0%
5	Equipment	Technology	Intersection (3 Commons)	68.75%
6	Equipsub01	Technology	Intersection (3 Commons)	70.96%
7	Agent	Broker-admin	Different Onts (0 Common)	0%
8	Agent	foaf-basic	Intersection (1 Common)	8.64
9	context	contextbroker	Intersection (1 Common)	90%
10	BuyTechSub04	BuyTechSub03	Sub Ontology (1 Common)	31.42

More real world ontologies have been utilised as shown in table 4.2:

Table 4.2: More testing for real world ontologies

Index	Ontology 1	Ontology 2	Relationship	Percentage Similarity
1	Beer	Beer_Partition1	Sub Ontology	42.0%
2	Beer	Beer_Partition2	Sub Ontology	14.0 %
3	Beer_Partition2	Beer_Partition3	1 Common Ont	8.0 %
4	Hotel A	Hotel B	Same Ontology	100%
5	Network A	Network B	Same Ontology	100%

Ontology's location:

- Beer: <http://www.purl.org/net/ontology/beer.owl>
- Beer_Partition1:
http://www.mindswap.org/2004/multipleOnt/FactoredOntologies/FactoredBeer/beer_partition1.owl
- Beer_Partition2:
http://www.mindswap.org/2004/multipleOnt/FactoredOntologies/FactoredBeer/beer_partition2.owl
- Beer_Partition3:
http://www.mindswap.org/2004/multipleOnt/FactoredOntologies/FactoredBeer/beer_partition2.owl
- Hotel A: <http://www.atl.lmco.com/projects/ontology/ontologies/hotel/hotelA.owl>
- Hotel B: <http://www.atl.lmco.com/projects/ontology/ontologies/hotel/hotelB.owl>
- Network A:
<http://www.atl.lmco.com/projects/ontology/ontologies/network/networkA.owl>
- Network B:
<http://www.atl.lmco.com/projects/ontology/ontologies/network/networkB.owl>

4.4 Related Work

Schema matching, which involves the determination of a set of correspondences that identify similar elements in two different schemas, is a critical issue in data exchange, data integration, data restructuring and schema evolution. Typical work on this area are

[7] [10] [11] and [14]. A survey on schema matching with classification of schema matching approaches is available at: <http://research.microsoft.com/~philbe/VLDBJ-Dec2001.pdf>. These works differ from our work since ontology captures more knowledge and possess different features from schemas.

Information Retrieval [1,6] is the science of information storage and searching documents, metadata which describe documents, or searching within databases. ‘Semantic search’[4] which is a field within Information Retrieval bears some relationship to our work. Semantic Search attempts to improve traditional searching technologies by employing semantic data to disambiguate semantic search queries and web text in order to increase relevancy of results. They are different from ontology comparison since their work mainly focuses on ‘storing’ and ‘querying’ information.

Klein *et al.* [51] discussed the topic of ontology versioning and change detection on the Web by looking at the characteristics of the version relationships between ontologies. They proposed a Web-based system that helps users to manage changes in ontologies. Users can use an adaptable rule-based mechanism to find and classify changes in RDF-based ontologies. In another work [52], Klein *et al.* presented a mechanism to find and specify relationships between versions of an ontology. The system enables ontology engineers to compare versions and to specify the relationships between the different versions of concepts.

Also in ontology versioning, Noy *et al.* [73] introduced PROMPTDIFF: a fixed-point algorithm for comparing ontology versions. The algorithm integrates different heuristic matchers such as same type, same name, single unmatched sibling, siblings with same

suffixes, etc. for comparing ontology versions. It combines these matchers in a fixed-point manner, using the results of one matcher as an input for others until the matchers produce no more changes. In another work, Heflin and Pan [36] discussed the problem of ontology versioning and then presented a model theoretic semantics for ontology versioning. They discussed how the theory model can be applied to RDF and OWL and how to compute perspective-based entailment using existing logical reasoners.

Wang and Ali [114,115] developed the first available online measurement tool for efficient ontology comparison based on a proposed senses refinement algorithm. The algorithm builds a senses set to accurately represent the semantics of the ontology. It automatically extracts senses from WordNet, the electronic lexical online database, and then removes unnecessary senses based on the relationship among the entity classes of the ontology, and specifies relationships and constraints of the concepts in the refined senses set. The senses refinement algorithm ensures the efficiency and accuracy of the ontology comparison by converting the measurement of ontology difference into simple set operations based on set theory. However, the proposed senses refinement algorithm only supports “is-a” relations to discover the semantics of the ontology. Although “is-a” is the most common relationship used in an ontology, the “part-whole” and other relationships should also be supported.

A large amount of work and tools on ontology versioning have been developed and the above are only typical examples. Ontology versioning bears some similarity to our work. However, as mentioned in section 4.1, ontology versioning focuses mainly on incremental changes in the ontology involving concept, relationship, or data type. Our focus is on the relationships between two ontologies within a specific application. The

difference between the two ontologies is not limited to incremental changes but may involve significant differences. In some cases, only a small part of the original ontology may be used to develop a new ontology. The proposed algorithm can determine that common part of the two ontologies. It can determine the relationship of the two ontologies as identical, sub ontology, intersection, or different. This is outside the scope of ontology versioning.

Maedche and Staab [63] presented an algorithm to measure ontology similarity by considering ontologies as two-layered systems, consisting of a lexical and a conceptual layer. Matching two ontologies therefore involve matching the two layers of the two ontologies. The lexical layer uses a string matching method to measure string similarity. The conceptual layer compares semantic structures of the two ontologies. Our work is different since we focus on comparing the structure of the two ontologies instead of considering the name changes of the concepts, properties, or relationships. As far as the authors are aware, there is only one work which compares two RDF graphs presented by Carroll [9] that is similar to our work. This is only one function within Jena which checks whether two ontologies are identical. It is unable to identify the sub ontology or intersection relationship. Moreover, the algorithm is based on graph isomorphism which is very time consuming. It is unnecessary to use graph isomorphism techniques since the concepts and relationships of the two ontologies are named.

Ontology alignment is a process of finding correspondences between semantically related entities in a pair of ontologies. There are two main approaches to ontology alignment, namely, local model and global model. In a local model, the alignment consists of methods which measure the correspondence at a local level, namely, only comparing one

concept with another and not comparing at the global scale of ontologies. Typical work in this approach includes [82], [1], [32] and [23]. Figure 4.7 presents an example of the local approach. In a global model, the alignment contains complex methods which enable correspondences at a global level to be measured. The methods to measure concept similarity may require a combination of several methods. [26] and [41] are typical examples of this global model. Ontology alignment focuses on *mapping* two ontologies from different ontologies. Our work is different since its purpose is determining if two ontologies share some common portions.

Figure 4.7: Example of a local approach for ontology alignment [1]

4.5 Summary

Ontologies may be used and modified by different applications. Therefore, ontologies used by such applications may be related. This leads to a need to check the relationship between concepts from different ontologies. The chapter has presented an algorithm which compares two ontologies to determine their relationship and where applicable, to

return their common portions. The proposed algorithm can be applied to applications related to ontology mining, mapping, merging, integration, and alignment. An example was introduced to illustrate how the algorithm works.

The algorithm will be employed in measuring concept similarity, which is the core of the proposed discovery system. By employing the proposed algorithm, the accuracy of the concept similarity measurement is enhanced. This is because if two concepts belong to a common part, they are considered to be in the same ontology. Concept similarity in the same ontology can be measured more accurately than if they belong to different ontologies. More details are presented in chapter 5.

Chapter 5

Computing Ontological Concept Similarity

5.1 Introduction

Semantic ontological concept similarity is a quantity determining how *similar* or how *close* the two concepts are semantically based on ontologies. It is cast as a real value ranging between 0 and 1. Concept similarity plays an important role in ontology activities. For example, in ontology mapping [21,43], alignment [22,73], and merging [60,72] applications, it is used to find the corresponding concepts of the two ontologies. In ontology integration [79], it is employed to determine if and how two ontologies are able to be integrated. In Web services discovery systems, it plays a core role in matching two Web services. As introduced in chapter 2, input, output, and operation of a Web service are described by ontological concepts. Matching two Web services involves matching inputs, outputs, and operations of the two Web services, respectively. So, it is basically computing ontological concept similarity. Matching Web services is a core operation of Web Service discovery, and hence measuring concept similarity is a key component of the discovery systems.

Different applications may have different definitions of concept similarity, which reflects the approaches used to compute the similarity. In the majority of approaches, the similarity is based on concept information such as name, property, and ontology

hierarchical structure as well as neighborhood concepts. Some approaches only use the ontological hierarchical structure while others only use names and properties. A minority of approaches uses the specific features of the ontology languages. These approaches have drawbacks in that they use only a part of the ontology information. This can cause mismatches. Moreover, most previous work only supports measuring concept similarity within the same ontology. Oundhankar *et al.* [75] is the sole exception that supports measuring similarity of concepts in the same ontology as well as in different ontologies. More details of the related work are introduced in section 5.2.

This chapter presents an algorithm to compute the concept similarity in the same ontology as well as in different ontologies. It improves Oundhankar's work [75] by checking if two concepts belong to a common ontology which was introduced in chapter 4. If the two concepts are within a common ontology, the similarity is computed as if they are in the same ontology. Measuring the similarity of two concepts within the same ontology yields a more accurate result as compared to estimating the similarity of concepts which are from different ontologies. If they are from different ontologies, the algorithm uses four components: syntactic, property, neighborhood, and domain similarity. The proposed algorithm improves previous work by adding the domain similarity dimension. With this added dimension, we improve our understanding about the concepts in the domain of the ontology. More detailed results of the proposed algorithm are introduced in section 5.5.

The chapter has introduced the semantic ontological concept similarity. Section 5.2 presents related work. Section 5.3 introduces an approach to measure the similarity between concepts in the same ontology, while section 5.4 discusses measuring the

similarity between two concepts from different ontologies. Section 5.5 describes a prototype and experiments to illustrate the methods discussed, followed by a conclusion of the chapter in section 5.6.

5.2 Related Work

Figure 5.1: The taxonomy of approaches to compute concept similarity

Since concept similarity plays an important role in ontology applications, it is expected that there is existing work in this area. This section presents a literature survey on this issue. Figure 5.1 presents a taxonomy of the approaches, which can be divided into methods for concepts in the same ontology and methods for concepts from different

ontologies. The former is further classified based on concept subsumption. If the concepts are subsumed, the approaches are sub-classified into methods based on description logic and methods based on concept information such as properties. The approaches that support concepts from different ontologies are separated into methods that support ontologies described by different languages and methods that support ontologies described by the same language.

Almost all existing work assumes that the concepts are either in the same ontology or from different ontologies but not both. Only Oundhankar *et al.* 's work [75] which will be introduced in section 5.5, handles concepts that are in the same ontology as well as from different ontologies.

5.2.1 Concept Similarity in the Same Ontology

There are two approaches to compute the concept similarity within the same ontology depending on whether the two concepts are subsumed.

5.2.1.1 Concepts are Subsumed

Under such a situation, the concept similarity can be computed using description logic or concept information.

Using description logic: As discussed in chapter 2, there are three methods to describe an ontology, namely, *first-order logic* [30], *frame-based logic* [50], and *description logic* [110]. With ontologies described by *description logic*, researchers have used features of the logic to compute concept similarity. By exploiting the strong points of description logic, the approach is more accurate, faster, and hence, achieves a higher performance.

The drawbacks are they can only be applied to ontologies using description logic and they can only compute concept similarity if the two concepts are subsumed. They are unable to consider the other concept information as well as other relationships except for *Subsumes* and *Inverse-subsumes*. Examples of this approach are algorithms developed by Benatallah *et al.* [4], Li and Horrocks [56], and Trastour *et al.* [10].

Using concept information: Typical examples of work in this category are [76], [75] and [92]. Paolucci *et al.* [76] defined four levels of similarity: *Exact*, *Plug-In*, *Subsumes*, and *Fail* in a Web Service application. This definition is based on concepts that have a direct relationship: super-concept (generalization) or sub-concept (specialization). However, the use of these terms is crucial. The most challenging aspect is that they open to different interpretations. The above terms are unable to be defined numerically. Said *et al.* [92] overcame this drawback by mapping these terms into integers. For example, *Exact* = 4, *Subsumes* = 3, *Inverse-subsumes* = 2, and *Fail* = 1. However, the values of these terms may vary, depending on particular contexts and applications.

5.2.1.2 Concepts are not Subsumed

Wu and Palmer's work [117] and Ehrig and Sure's work [21] are typical examples of this category. Ehrig and Sure [21] measure similarity based on the following components: the semantic distance of labels of two concepts; the semantic distance between the properties of concepts; the semantic distance of super-concepts or sub-concepts. Finally, the similarity of two concepts A and B is the summation of the above components. This approach is based on various features (labels, properties, etc.) of concepts. Each component consists of a method to match a feature. The disadvantages of this work are

that it ignores the ontology taxonomy and requires complex computation. In contrast, Wu and Palmer [117] compute concept similarity based on the taxonomy of the ontology. It is computationally efficient as the algorithm just needs to locate the classes in the hierarchy in order to compute the distances between the nodes.

5.2.2 Similarity between Concepts from Different Ontologies

The computing of concept similarity in different ontologies can be sub-divided into approaches that support ontologies described by the same language and approaches that support ontologies described by different languages.

5.2.2.1 Ontologies Described by the Same Language

An *et al.* [19,20] have developed the GLUE system that uses machine learning techniques to find a mapping between ontologies. For each concept in one ontology, GLUE finds the most similar concept in the other ontology by using joint probability distribution of the concepts involved. Given two concepts A and B, the joint distribution consists of $P(A,B)$, $P(A,\bar{B})$, $P(\bar{A},B)$, and $P(\bar{A},\bar{B})$ representing the probability of instances belonging to concepts. $\frac{P(A \cap B)}{P(A \cup B)}$ which is known as the Jaccard coefficient [85] is used to compute the similarity between two concepts A and B. To compute the joint distribution more accurately, a large number of instances are needed. Moreover, using machine learning techniques is time consuming. Therefore, this method is unsuitable for commercial Internet services which require good performance.

The LARKS (Language for Advertisement and Request for Knowledge Sharing) project [101-103] is a matchmaker agent which matches agents based on their descriptions. Concept similarity is a core component of the matching process. LARKS's knowledge is specified as local ontologies in ITL ontology language. However, since the expressivity of ITL is restrictive, there is a need to express additional associations between concepts. For this purpose, a weighted *associative network* is used. The *associative network* is a semantic network with directed edges between concepts as nodes. Each edge denotes a binary relation between two concepts, and is labeled with a numerical weight. The weight represents the strength of belief in that relation. The *semantic network* consists of three kinds of weighted binary relationships: super-class, sub-class, and a *positive association* among concepts. The *positive association* is the most general relationship among concepts in the network indicating that they are synonyms in some context. The similarity between two concepts in an *associative network* is then computed as the length of the shortest path between them. This method supports the ITL language which may not be widely used in the future since there are many alternative ontology languages.

5.2.2.2 Ontologies Described by Different Description Languages

Almost all approaches to measure concept similarity assume the ontologies are based on the same ontology language. However, Rodriguez and Egenhofer [87] presented an approach to compute similarity between concepts from ontologies described by different languages. The proposed algorithm can be applied to any ontology specification. Examples of various specifications are WordNet [67] and Spatial Data Transfer Standard (SDTS) [108]. The similarity function which is determined by using *word matching*, *distinguishing features*, and *semantic neighborhoods* is defined by the weighted sum of

the similarity of each specification component. The function is shown in the following formula:

$$S(a^p, b^q) = w_w \cdot S_w(a^p, b^q) + w_u \cdot S_u(a^p, b^q) + w_n \cdot S_n(a^p, b^q)$$

$$\text{for } w_w, w_u, w_n \geq 0 \text{ and } w_w + w_u + w_n = 1.0$$

where S_w , S_u , and S_n are the *word matching*, *distinguishing features*, and *semantic neighborhoods* similarity between concept a of ontology p and concept b of ontology q . w_w , w_u , and w_n which are the weights of the respective components S_w , S_u , and S_n are assigned values depending on the characteristics of the ontologies.

This approach covers all information of concept and ontology taxonomy. However, the domain of the two ontologies should be considered since it affects the meaning of the concept. This approach is useful to improve the retrieval and integration of information. Arising from the large number of heterogeneous agents in the Internet which use different specification languages, the approach is needed to coordinate across distributed networks of information.

We have presented a survey of approaches to compute concept similarity. A taxonomy presenting various approaches was introduced. Research has focused mainly on computing similarity of concepts from the same ontology. It has only been in recent years that computing similarity of concepts from different ontologies has emerged. Section 5.3 introduces our approach to measure concept similarity which supports concepts within the same ontology as well as in different ontologies based on OWL specification.

5.3 Concept Similarity in the Same Ontology

As mentioned in the previous section, there are two approaches to compute concept similarity within the same ontology, depending on whether the two concepts possess a subsumed relationship.

5.3.1 Concepts are Subsumed

Figure 5.2: The relationship of concepts

Two concepts have a subsumed relationship if one is a sub-class or super-class of the other. For example, *Electronic Equipment*, *Computer*, *Portable Computer*, and *Laptop* in figure 5.2 display different subsumed relationships. The similarity is defined as follows.

The terms *Exact*, *Subsumes*, *General*, *Inverse*, and *Specific* given below are real numbers ranging between 0 and 1. Their values depend on particular applications and contexts.

- *Exact match (Computer, Computer)*: It is the most accurate match. It happens when the two descriptions are semantically equivalent. The similarity degree for this match is given a value *Exact* and this is the highest similarity.
- *Subsumes match (Computer, Personal Computer)*: *Personal Computer* is subsumed by *Computer*. As *Computer* is a direct super-class of *Personal Computer*, it is more general than *Personal Computer*. This match is less accurate than exact match. The similarity degree for this match is *Subsumes*.
- *General match (Computer, Laptop)*: *Computer* is more general than *Laptop* but *Computer* is not a direct super-class of *Laptop*. We assume that there are x levels (with $x \geq 2$) between *Computer* and *Laptop*. The similarity degree for this match is “*Subsumes* – $a^{*(x-1)}$ ”. ‘ a ’ is a reducing coefficient whose value is given depending on the particular application. We reduce the general similarity for super-concepts based on the level of ancestry. The similarity is reduced by a value of $a^{*(x-1)}$ for ancestors above the direct parent, where $x \geq 2$, until it is reduced to zero. The similarity will only reach zero when $x = \frac{\text{Subsumes}}{a} + 1$. The value of ‘ a ’ is chosen such that the similarity is greater than zero. Therefore, ‘ a ’ must satisfy $a \leq \frac{\text{Subsumes}}{\text{Length of the Ontology} - 1}$.

- *Inverse-Subsumes match (Personal Computer, Computer)*: *Personal Computer* is more specific than *Computer* and *Personal Computer* is a direct sub-class of *Computer*. The similarity degree for this match is *Inverse-Subsumes*.
- *Specific match (Laptop, Computer)*: this is the inverse of the general match. *Laptop* is more specific than *Computer* but *Laptop* is not a direct sub-class of *Computer*. We assume that there are x levels (with $x \geq 2$) between *Computer* and *Laptop*. The similarity degree for this match is “*Inverse-Subsumes* – $b^{*(x-1)}$ ”. Similar to *General match*, we reduce the specific similarity for sub-concepts based on the level of descendants. The similarity is reduced by a value of $b^{*(x-1)}$ for descendants below the direct child concept, where $x \geq 2$, until it is reduced to zero. The similarity must be greater than zero. Therefore, ‘ b ’ must satisfy
$$b \leq \frac{\text{Inverse} - \text{Subsumes}}{\text{Length of the Ontology} - 1}.$$

We use levels of two concepts to compute general and specific similarity. The two values ‘ a ’ and ‘ b ’ are usually different in the context of matching semantic Web services since concepts represent how the two services satisfy each other. General concepts satisfy a requirement more than the specific concepts. Therefore, ‘ a ’ is usually less than ‘ b ’.

The above terms and values *Exact*, *Subsumes*, *Inverse-Subsumes*, ‘ a ’ and ‘ b ’ are defined depending on the particular applications since different applications may have different approaches in designing ontology. For example, with the ‘Electronic Equipment’ ontology in figure 5.2, a designer may introduce more levels of granularity by adding a ‘Digital Computer’ concept between the two concepts ‘Computer’ and ‘Personal Computer’. However, another designer may opt for a coarser level of granularity by

deleting the ‘Personal Computer’ concept. As a result, the ‘Portable Computer’ concept will be the direct sub-class of the ‘Computer’ concept. This leads to different applications having different definitions on these terms and values. Therefore, the users are allowed to determine and change the values for their particular applications.

5.3.2 Concepts are not Subsumed

Two concepts do not have a subsumed relationship if they do not have super/sub-class relationships. Under this circumstance, the two concepts are connected through a common ancestor. *Portable Computer* and *Analog Computer* in figure 5.2 is an example of a concept pair connected to each other through *Computer*. To calculate the similarity of the two concepts in this situation, we adopt a technique to compute similarity in taxonomy [117]. The concept similarity between *Portable Computer* and *Analog Computer* is:

$$\text{conceptSim}(\text{Portable Computer}, \text{Analog Computer}) = \frac{2 * N3}{N1 + N2 + 2 * N3}$$

Computer is the least common super-concept of *Portable Computer* and *Analog Computer*. $N1$ is the number of edges on the path from *Portable Computer* to *Computer*. $N2$ is the number of edges on the path from *Analog Computer* to *Computer*. $N3$ is the number of edges from path *Computer* to root *Equipment Electronic*. Hence, $\text{conceptSim}(\text{Portable Computer}, \text{Analog Computer}) = 0.4$. This can be compared with another example involving *Analog Computer* and *Personal Computer*:

$$\text{conceptSim}(\text{Analog Computer}, \text{Personal Computer}) = \frac{2 * N3}{N1 + N2 + 2 * N3} = \frac{2 * 1}{1 + 1 + 2 * 1} = 0.5$$

In the situation where the two concepts are not connected to each other through any ancestor, the similarity is measured in the same manner as they are from different ontologies.

5.4 Concept Similarity in Different Ontologies

This section introduces the method for computing the semantic similarity of two concepts from different ontologies, assuming that no common part is found between both ontologies. The formula to measure similarity between concepts from different ontologies includes four components: syntactic, property, neighborhood, and domain similarity as shown in the equation (1):

$$CS = \frac{w_s * synSim + w_p * proSim + w_c * conSim + w_n * neiSim}{w_s + w_p + w_c + w_n} \quad (1)$$

where w_s , w_p , w_c , and w_n are weights defined by users. A weight indicates the importance of a corresponding component to the users. The weights are real values ranging between 0 and 1. Since the users developed the requested Web service, they are able to gauge how necessary each component is. However, it is unrealistic to expect the users to assign a value to represent the importance of a component [106]. Hence, we propose a mapping table that establishes a correspondence between “importance” and quantitative values based on [65]. Therefore, instead of specifying quantitative values, the users can choose qualitative terms, as shown in table 5.1. In the following discussion, weights in other equations are set in a similar manner. In the examples and testing that follows, all the weights are assigned default values of 0.5 since all the components are assumed to be equally important. In the case when the similarity component produces a value “nil”, the

component will be omitted in the computation. A similarity component returns a “nil” value if the information is unavailable.

Table 5.1: Weight mapping table

Term	Quantitative
<i>Redundant</i>	0.0
<i>Very Unimportant</i>	0.1
<i>Unimportant</i>	0.2
<i>Low</i>	0.3
<i>Slightly Below Average</i>	0.4
<i>Average</i>	0.5
<i>Slightly Above Average</i>	0.6
<i>Above Average</i>	0.7
<i>Important</i>	0.8
<i>Very Important</i>	0.9
<i>Mandatory</i>	1.0

Figure 5.3 demonstrates how the similarity between two concepts is measured: *Computer* and *Computing Device* belong to *Technology* and *Equipment* ontologies, respectively. The similarity between *Computer* and *Computing Device* is calculated based on syntactic, property, neighborhood, and domain similarity of the two concepts. The following sections provide details of the four similarity components.

Figure 5.3: An example of measuring concept similarity

5.4.1 Syntactic Similarity

Syntactic similarity computes the similarity between the concept label and concept description of the two concepts, respectively. As introduced in chapter 2, each concept in an ontology is labeled (concept name) and has a short text (concept description) which describes its function. The concept label in OWL is defined as a word or a set of words. Syntactic similarity for two concepts, A and B, is computed as follows:

$$SynSim(A, B) = \frac{w_l * LabSim(A, B) + w_d * DesSim(A, B)}{w_l + w_d}.$$

where $LabSim()$ and $DesSim()$ are label similarity and description similarity, respectively.

Weights parameters w_l and w_d are both assigned in the same manner as discussed above.

5.4.1.1 Concept Label Similarity

A concept label contains a set of words representing the name of the concept. There are some techniques to compute word similarity such as n-gram [2,93,121] and token Matcher [81] but most of these only consider a word as a string of characters. The semantic relationship of the words, which is very important in computing word similarity, has been ignored. To overcome this drawback, WordNet [66] is used.

WordNet is a semantic lexicon for the English language. It was designed to provide the relationship between four types of Parts of Speech (POS) - noun, verb, adjective, and adverb. The smallest unit in a WordNet which is used to describe the meaning of words is synset. Synset includes the words, its explanation and its synonyms. The exact meaning of one word under one type of POS is called a sense. Therefore, the same word may have different senses in different synset. In other words, synsets are equivalent to senses which are sets or terms with synonymous meanings. For example, consider the words *computer*, *computing machine*, and *computing device*. They have a common synset that defines “a machine for performing calculations automatically”. Notice that one word may belong to different synsets because of its different meanings. For instance, *computer* also belongs to the synset which contains *calculator*, *reckoner*, *figurer*, and *estimator* having meaning: “an expert at calculation (or at operating calculating machines)”.

The taxonomy in WordNet is presented as an undirected graph and word similarity is measured as "The shorter the path from one node to another, the more similar they are" [108]. The path length is measured based on the number of nodes/ vertices rather than links/edges where the length between two members from the synset is 1 (synonym

relation). Consider the graph [96] in figure 5.4, the length between *car* and *auto* is 1, *car* and *truck* is 3, *car* and *bicycle* is 4, and *car* and *fork* is 12.

Figure 5.4: Undirected graph to represent synonym set relations [96]

The WordNet database only stores root words. Therefore, before using WordNet, there is a preprocessing step to convert words to their roots. The database contains about 150,000 words organized in over 115,000 synsets for a total of 207,000 word-sense pairs. It should be noted that abbreviations and acronyms are not stored in WordNet database. The similarity between two words is given a value 0 if either one is not found in the database. Unfortunately, in the real world, abbreviations and acronyms are frequently used to describe the label. For example, if one concept label is “*prchsOrder*” and the other concept label is “*purchaseOrder*” then the similarity is given a value 0 although both labels are obviously very similar.

If one of the words in a label pair is not in WordNet, Jaro [80] is used to compute the similarity of these two words. Jaro [80] introduced a method to compare two strings by using insertion, deletion, and transposition processes. The three main steps in this algorithm are, compute the string lengths, find the number of common characters in the two strings, and find the number of transpositions. Common characters in the strings are the matching characters and must be within $\frac{1}{2}$ the length of the shorter string. The similarity between two strings s_1 , s_2 with the lengths as str_len1 and str_len2 is calculated as:

$$\text{Jaro}(s_1, s_2) = \frac{1}{3} \left(\frac{\#common}{str_len1} + \frac{\#common}{str_len2} + \frac{\#common - \#transpositions}{\#common} \right)$$

For example, consider the similarity between the words “*prchsOrder*” and “*purchaseOrder*” whose lengths are 10 and 13, respectively. The number of common characters is 10. The number of transpositions is 6. Hence:

$$\text{Jaro} ("prchsOrder", "purchaseOrder") = \frac{1}{3} \left(\frac{10}{10} + \frac{10}{13} + \frac{10 - 6}{10} \right) = 0.723$$

Jaro is a simple but efficient algorithm that can measure the similarity between two strings based on the position of characters within these strings. However, it is unable to capture the meaning of the words. Therefore, WordNet is used first, and if one of the words is not found in the database, Jaro is used.

Table 5.2 shows the label similarity examples. Concept labels are taken from the two ontologies *Technology* and *Equipment* as presented in figure 5.3. In case 2, “*Electronic Technology*” cannot be found in the WordNet database. Therefore, Jaro is used. Cases 3,

4, and 5 present the advantage of using WordNet. Actually, these word pairs are synonyms and thus they have the same sense in WordNet. As a consequence, the similarity results are 1. If other string matching approaches are used, the similarity value will be low since they do not capture the meaning of the words and only consider the words as a string of characters.

Table 5.2: Concept label similarity examples

Cases	Concept label in Technology ontology	Concept label in Equipment ontology	Similarity	Method
1	Technology	Equipment	0.563	WordNet
2	Electronic Technology	Electronic Equipment	0.811	Jaro
3	Computer	Computing Device	1.0	WordNet
4	Server	Host	1.0	WordNet
5	Personal Computer	PC	1.0	WordNet

5.4.1.2 Concept Description Similarity

Concept description is a string of words and thus, measuring concept description similarity is basically measuring similarity between two strings. There are several string matching approaches. They include *Monge Elkan* string-edit distances [68], TFIDF distance metric [13], supervised learning for string-edit distance metrics [86], and combining the results of different distance functions [104]. In a comparison of these

methods, Cohen *et al.* [14] pointed out that the use of *Monge Elkan* string-edit distances [68] is one of the best choices. This method is adopted for measuring concept description.

Table 5.3 shows concept description similarity examples using TFIDF distance metric. Concept descriptions are taken from the concepts in the two ontologies *Technology* and *Equipment* as presented in figure 5.3 and their corresponding labels were shown in table 5.2.

Table 5.3: Concept description similarity examples

Cases	“Description” of [Concept] in Technology ontology	“Description” of [Concept] in Equipment ontology	Similarity
1	“the practical application of science to commerce or industry” [Technology]	“an instrument needed for an undertaking or to perform a service” [Equipment]	0.1
2	“Technology that involves the controlled conduction of electrons” [Electronic Technology]	“equipment that involves the controlled conduction of electrons” [Electronic Equipment]	0.854
3	“a machine for performing calculations automatically” [Computer]	“an electronic device for performing calculations automatically” [Computing Device]	0.850
4	“a computer that provides client stations with access to files and printers as shared resources to a computer network” [Server]	“a computer that provides client stations with access to files and printers as shared resources to a computer network” [Host]	1.0
5	“a medium-sized computer which is used mainly by people at home rather than by large organizations” [Personal Computer]	“a small digital computer based on a microprocessor and designed to be used by one person at a time” [PC]	0.129

Table 5.4: Concept label similarity examples

Cases	Concept Technology ontology	Concept Equipment ontology	Similarity
1	Technology	Equipment	0.331
2	Electronic Technology	Electronic Equipment	0.832
3	Computer	Computing Device	0.925
4	Server	Host	1.0
5	Personal Computer	PC	0.564

Based on the results of concept labels and concept descriptions in tables 5.2 and 5.3, respectively, the final syntactic similarity of concept pairs are shown in table 5.4.

5.4.2 Property Similarity

A concept may or may not have properties. The properties are matched on a one-to-one basis in such a way so as to maximize the average property similarity. It is measured similar to the system by Oundhankar *et al.* [75]. Similar to concept, a property also has a name and description. In addition, it contains range and cardinality. To compute the *property similarity*, all the information of the two properties should be matched. The method to compute property name and description similarity is the same as that of *syntactic similarity*.

Range of a property is either a primitive data type or another concept. If both ranges are primitive data types, then the similarity between two ranges is as shown in table 5.5. Note that for the sake of simplicity, table 5.5 does not show a comprehensive set of all data types. If one range property has a primitive type and the other has a concept, the ranges are incompatible; therefore the range similarity is 0. If two range properties are concepts, the matching is carried out recursively as with the computing of two concepts with all the four components.

Table 5.5: Range similarity values when range is primitive data type

	Range of Property 1					
		Integer	Long	Float	Decimal	String
Range of Property 2	Integer	1	0.9	0.8	0.7	0.3
	Long	0.9	1	0.8	0.7	0.3
	Float	0.8	0.8	1	0.7	0.3
	Decimal	0.7	0.7	0.7	1	0.3
	String	0.3	0.3	0.3	0.3	1

Cardinality of a property permits the specification of the exact number of elements in a relation. We define the cardinality similarity between two properties A and B, namely, $\text{CarSim}(A,B)$, as follows:

- If both A and B are *functional properties* or if the cardinality of property A is equal to the cardinality of property B then $\text{CarSim}(A,B) = 1$.

- If the cardinality of property A is less than the cardinality of property B, then $\text{CarSim}(A,B) = 0.9$.
- If the cardinality of property A is greater than the cardinality of property B, then $\text{CarSim}(A,B) = 0.7$.
- Otherwise. $\text{CarSim}(A,B) = 0.5$.

The final *property similarity* is the weighted combination of the three components: *syntactic*, *range*, and *cardinality* similarity.

The values are determined heuristically. For example, the best matches of an Integer value should be another Integer value. In this case, the similarity is 1.0. In matching an Integer value with a Long value, the similarity should be lesser, compared with the best match since the Long value has a large range. Therefore, we chose the similarity is 0.9. In matching an Integer value with a Float value, the similarity should be lesser, since the Float value is even more different when compared with an Integer and Long because Float contains real values while Integer and Long do not. Therefore, we chose the similarity is 0.8. The same reasoning applies to the other similarity matching pairs. The $\text{CarSim}(A,B)$ values chosen are based on the paper by “Oundhankar, S.; Verma, K.; Sivashanugam, K.; Sheth, A.; and Miller, J.: Discovery of web services in a Multi-Ontologies and Federated Registry Environment, *International Journal of Web Services Research*, 1(3), 2005. In their work, the values have also been determined experimentally. However, users are allowed to change the above values if it is necessary.

Table 5.6: Similarity values when range is primitive data type

Case	Property Pair		Syntactic Similarity	Range Similarity	Cardinality Similarity	Property Similarity
	Technology	Equipment				
1	hasOS	hasOS	1.0	1.0	1.0	1.0
2	hasOS	hasCPU	0.35	1.0	1.0	0.78
3	hasCPU	hasOS	0.35	1.0	1.0	0.78
4	hasCPU	hasCPU	1.0	1.0	1.0	1.0

Table 5.6 represents property similarity measurement based on the two concepts *Computer* and *Computing Device* in figure 5.3. Each concept has two properties, namely, *hasOS* and *hasCPU*. The two properties of the *Computer* are matched against the two properties of the *Computing Device*. Hence, we have four matched property pairs. The properties have the same “Range” which is “String”. Therefore, range similarity results in a value 1 for all the four cases. The “Cardinality” of both properties have the same values are 1 and therefore, the cardinality similarity obtains a value 1 for all the four cases. Each property has two mapping but cases 1 and 4 produce the best matches.

5.4.3 Domain Similarity

Domain which is also known as context of an ontology is important to determine if a concept belongs to a specific ontology. For example, assume that there is a *Vehicle* ontology which has a concept *Car*. Another *Asset* ontology also has concept *Car*. If we do not consider the domain of the two ontologies, the concept similarity of two concepts

from the two ontologies will be very high. However, the two *Car* concepts have different meanings in the two ontologies: one refers to a means of *transportation*; the other refers to a *financial asset*. Therefore, the similarity of the two *Car* concepts of the two ontologies is lower than first perceived. In short, domain must be considered in computing concept similarity.

To compute the domain similarity of the two concepts, we compute the similarity of the roots of the two ontologies since the root represents the context (domain) of the ontologies. The root is a special concept in an ontology, as it does not have super-concepts. Note that every OWL ontology has concept “Thing” as a root but this is not a “real” root. It is an artificial root and does not convey any meaning in the ontology. Therefore, the concepts just below the concept “Thing” are treated as roots. The root similarity is computed based on syntactic and property similarity which were presented above. An ontology may have one or more roots. In these cases, domain similarity is the highest similarity between root pairs.

As an example, it is noted that the root concepts of *Computer* and *Computer Device* in figure 5.3 are *Technology* and *Equipment*, respectively. In this case, domain similarity which is the similarity between the two concepts *Technology* and *Equipment* is measured based on their syntactic and property similarity. The syntactic and property similarity are computed as 0.331 and 1, respectively. As a consequence, domain similarity is 0.665, namely the mean of these two values. The syntactic and property similarity are measured as presented in section 5.4.1 and 5.4.2 above.

5.4.4 Neighborhood Similarity

Neighborhood concepts which are the direct super-concepts, sub-concepts, and equivalent concepts of a given concept, give further information regarding the concept. Therefore, to measure concept similarity, these neighborhood concepts should be involved. The neighborhood similarity is computed based on super/sub-concept similarity and equivalent similarity using the following equation:

$$neighborSim = \frac{w_s * supsubSim + w_e * equiSim}{w_s + w_e}$$

where *supsubSim* and *equiSim* are super/sub-concept similarity and equivalent similarity, respectively.

5.4.4.1 Measuring supsubSim Similarity

supsubSim is measured as follows:

$$supsubSim = \frac{w_p * supSim + w_b * subSim}{w_p + w_b}$$

where *supSim* and *subSim* are direct super-concept similarity and direct sub-concept similarity, respectively. The super-concept similarity is computed as follows:

$$supSim(C_p, C_R) = \frac{\sum Sim(C_{p'}, C_{R'})}{n}$$

where C_p and C_R are the two concepts whose similarity we want to measure. $C_{p'}$ and $C_{R'}$ are super-concepts of C_p and C_R , respectively. “n” is the number of pairs of matched

super-concepts. For all C_P and C_R , $\text{Sim}(C_P, C_R)$ is computed based on syntactic, property, and context similarity. *subSim* is computed in the same way as *supSim*.

For example, in figure 5.3, *Computer* and *Computer Device* have super-concepts *Electronic Technology* and *Electronic Equipment*, respectively; their sub-concepts are *Server*, *Personal Computer*, and *PC*, *Host*, respectively. The *supSim* and *subSim* are measured as follows:

- *supSim* is the similarity between *Electronic Technology* and *Electronic Equipment* based on their syntactic, property, and context similarity.
- *subSim* is the average of the similarities of two concept pairs: (Personal Computer and PC) and (Server and Host) which are measured based on syntactic, property, and context similarity.

Table 5.7: *SupSim* and *SubSim* similarity measurement

Case	Concept Pair		Syntactic Similarity	Property Similarity	Domain Similarity	SupSim/SubSim Similarity	
	Technology	Equipment					
1	Electronic Technology	Electronic Equipment	0.883	1.0	0.665	0.849	
2	Personal Computer	PC	0.565	1.0	0.665	0.743	0.816
3	Server	Host	1.0	1.0	0.665	0.888	

Table 5.7 represents *SupSim* and *SubSim* similarity measurement of the two concepts *Computer* and *Computing Device*, $\text{supSim} = 0.849$, $\text{subSim} = \frac{0.743 + 0.888}{2} = 0.816$. As a consequence, $\text{supsubSim} = \frac{0.849 + 0.816}{2} = 0.833$.

5.4.4.2 Measuring Equivalent Similarity

In OWL, *sameConceptAs* and *equivalentClass* relationships infer that the concepts have the same meaning. The two relationships are named equivalent relationship. Therefore, to match two concepts, we should also consider these relationships. Assume that we would like to compute the similarity between two concepts A and B from different ontologies. A has equivalent relationship with C, and B has equivalent relationship with D. We first compute the concept similarity between: A-B, A-D, B-C, and C-D. Assume that the similarity values are S_{A-B} , S_{A-D} , S_{B-C} , and S_{C-D} , respectively. The similarity between A and B is the maximum value of S_{A-B} , S_{A-D} , S_{B-C} , and S_{C-D} .

With the example mentioned above, the concepts do not have *sameConceptAs* and *equivalentClass* relationships. Therefore, the weight for the similarity will be set to 0. As a result, the final neighborhood similarity, namely *neighborSim*, is 0.833.

5.4.5 Discussion

Our algorithm improves previous work by supporting concepts within the same ontology as well as from different ontologies. We first consider concepts within the same ontology. In cases where they are subsumed, we define similarity terms numerically. Moreover, the values of these terms are able to be changed to adapt to specific contexts and applications. In cases where the concepts are not subsumed, if the two concepts are connected to each

other, the approach by Wu and Palmer [117] is adopted because of the advantages mentioned in section 5.2.1.2. If they are not, the similarity is calculated as if the two concepts are from different ontologies.

When two concepts are from different ontologies, we check if they belong to a common ontology. If they are, the similarity is computed as if the concepts belong to the same ontology. This leads to a more accurate value as it avoids the subjectiveness present in two separate ontologies and requires less computation. If there is no commonality in the two ontologies, four components, namely, syntactic, property, neighborhood, and domain similarity are employed. The algorithm improves the previous work by eliminating mismatches and improving matching by considering the domain of the two ontologies which is an important factor in the meaning of the concepts.

5.5 Prototype and Testing

This section introduces a prototype, examples, and experiments. The prototype is an implementation of the concept similarity algorithm. Examples with 15 concept pairs are presented to illustrate how the algorithm works. Experiments with 300 concept pairs were carried out. The purpose of presenting examples is to illustrate how the matching is carried out while the testing confirms the validity of the algorithm.

5.5.1 Prototype of the Algorithm

Figure 5.5 shows a screenshot of the prototype of the algorithm. Each side of the panel displays the taxonomy of an ontology whose concept similarity we would like to measure.

Super-concepts are parent nodes, while sub-concepts are child nodes in the tree. This representation illustrates the structure of the ontology and enables each concept of the two ontologies to be browsed. When two concepts are selected (one from each side of the frames) to be compared, the algorithm will check if the two concepts belong to a common ontology. If they are, the method to measure concept similarity within the same ontology is invoked. This leads directly to the final result. The four similarity components, namely, *syntactic*, *property*, *neighborhood*, and *domain* display a “nil” value. This is the case in figure 5.5. If the two concepts do not belong to a common ontology, the four similarity components are involved and the results of these components are displayed.

Figure 5.5: Prototype of the algorithm

As mentioned, users are able to customize the weights in equation (1) in section 5.4 and semantic definition values in section 5.3. Figure 5.6 shows the user settings for the various weights and parameters mentioned in the earlier sections.

Weight Parameters			
WsynSim	WproSim	WneiSim	WdomSim
0.25	0.25	0.25	0.25

Semantic Distance Definition			
Subsumes	InverseSub	General	Specific
0.85	0.5	0.01	0.05

Figure 5.6: Users customize parameter weights and semantic similarity values

5.5.2 Concept Similarity Measurement by Oundhankar *et al*

As mentioned in section 5.1, only one previous work is similar to ours in handling concept similarity measurement in the same as well as in different ontologies. The method to measure the similarity is employed in a multi-ontology Web service discovery system [75]. The similarity is divided into two cases: when the two concepts are in the same ontology and when they are from different ontologies.

5.5.2.1 Similarity between Concepts in the Same Ontology

If the concepts are linked to each other by a subsumption relationship, the similarity is the weighted average of syntactic and property similarity which are introduced below. This approach has a major drawback in that it ignores the hierarchical structure of the ontology which is very important in presenting the related meaning of the concepts. Our method, described in 5.3.1, exploits the hierarchical structure.

In the cases where the concepts do not possess a subsumption relationship, the similarity is calculated in the same manner as matching concepts from different ontologies. This approach has a drawback in that it ignores the non-subsumption relationship between the concepts which belong to the same ontology.

5.5.2.2 Similarity between Concepts from Different Ontologies

The similarity is computed based on four components: *syntactic similarity*, *property similarity*, *coverage similarity*, and *context similarity*. The details of each component are as follows:

- *Syntactic similarity* is calculated using various names and string algorithms. As mentioned in section 5.4.1, the algorithm ignores the semantics of the words. Our work uses WordNet to overcome the semantic issue.
- *Property similarity* is similar to our algorithm which is based on syntactic, range, and cardinality of the properties.
- *Coverage similarity* is calculated based on super-concepts and sub-concepts of the corresponding concepts. This similarity dimension is similar to *supsubSim* similarity which is a component of our neighborhood similarity.
- *Context similarity* focuses on the semantic *similarity* and semantic *disjointness* of the concepts based on relationships between concepts such as *sameClassAs*, *equivalentClass*, *disjointWith*, *complementOf*, etc. These concepts are divided into two sets, namely, *SameConceptSet* and *DifferentConceptSet*. This similarity is similar to equivalent similarity which is a component of our neighborhood similarity.

5.5.2.3 Comparison with Oundhankar *et al.*'s work

To measure concept similarity, the first task is to determine whether the two concepts are within the same ontology or from different ontologies. Oundhankar *et al.* do not clearly discuss this issue but only state the assumption “while matching concepts from the same ontology, ...” [75]. This implies that they have simply compared the URLs of the two ontologies. This is a major disadvantage of Oundhankar *et al.*'s work. As mentioned in chapter 4, this is a shortcoming since ontologies may be deployed in different URLs after they are first created. We have overcome this problem by comparing the two ontologies. The algorithm can determine if the two ontologies are the same although they are located in different URLs. Moreover, we have considered cases where the two ontologies have common portions. Computing the similarity of two concepts within the common portion yields a more accurate result as compared to measuring the similarity of concepts which are from different ontologies.

In cases when two concepts are from different ontologies, both Oundhankar *et al.* and our work use four components to measure the similarity. Oundhankar *et al.*'s work uses the *syntactic*, *property*, *coverage*, and *context* similarities. Our work uses *syntactic*, *property*, *neighborhood*, and *domain* similarities. Actually, the *coverage* and *context* similarities in [75] are the *supsubSim* and *equivalent* in our neighborhood dimension, respectively. Therefore, the four components of Oundhankar *et al.*'s work are equivalent to our three components: *syntactic*, *property*, and *neighborhood* similarities. Our approach has added a *domain* similarity dimension. By considering domain similarity, there is additional information about a concept. This dimension has improved the similarity by avoiding mismatches.

5.5.3 Examples of Concept Similarity Tests

This section introduces the concept similarity measurement of 15 concept pairs which are divided into three groups. Each group consists of 5 concept pairs.

- In the first group, concept pairs belong to some common ontologies. In this case, the intersection of two ontologies is non-null.
- In the second group, concept pairs do not belong to any common part. However, the concept pairs have the same name but they are from different domains.
- In the third group, concept pairs do not belong to any common part. In this case, the two ontologies may be different or they have common parts but the two concepts are not in the common parts.

The following acronyms have been adopted for the tests that were conducted:

- CS-MOD is the Concept Similarity measurement described in this chapter which forms the backbone of MOD. This Multi-Ontology Web services discovery system will be described in chapter 6.
- CS-O is the Concept Similarity measurement based on Oundhankar *et al.*'s work.

In the examples and the rest of the testing, the results of CS-MOD and CS-O are compared against the “ground truth”.

The “ground truth” of concept similarity is the expected value which has been derived manually based on the knowledge related to the concepts including the name, description, properties, sub- and super-concepts of both concepts. The domains and other information

of the two ontologies are also involved. Since defining ontologies is subjective, it is difficult to state the numeric value of the “ground truth”. The concept similarity is classified into eleven clusters as follows *Fail*, *Very Low*, *Low*, *Below Average*, *Slightly Below Average*, *Average*, *Slightly Above Average*, *Above Average*, *High*, *very High*, and *Identical*. The “Ground truth” is the mean of the cluster which the concept similarity belongs to. Each cluster is mapped into a value as shown in table 5.8. We collaborated with SIMTech, Singapore Institute of Manufacturing Technology, (www.simtech.a-star.edu.sg) to determine the “ground truth”. SIMTech was chosen because they have many experts in the supply chain domain and e-business area. In addition, they also have been asked to set weights and parameters mentioned in the earlier sections which are used in the following examples and experiments.

Table 5.8: Clusters and their information

Clusters	Ranges	Mean
<i>Fail</i>	[0.00 ... 0.05]	0.025
<i>Very Low</i>	(0.05 ... 0.15]	0.1
<i>Low</i>	(0.15 ... 0.25]	0.2
<i>Below Average</i>	(0.25 ... 0.35]	0.3
<i>Slightly Below Average</i>	(0.35 ... 0.45]	0.4
<i>Average</i>	(0.45 ... 0.55]	0.5
<i>Slightly Above Average</i>	(0.55 ... 0.65]	0.6
<i>Above Average</i>	(0.65 ... 0.75]	0.7
<i>High</i>	(0.75 ... 0.85]	0.8
<i>Very High</i>	(0.85 ... 0.95]	0.9
<i>Identical</i>	(0.95 ... 1]	0.97

5.5.3.1 Concept Pairs Sharing a Common Ontological Part

Consider the *Equipment* and *Computer* ontologies which were introduced in figures 4.4a and 4.4b, respectively. The two ontologies possess a common part which was presented in figure 4.5a. Consider five concept pairs from the two ontologies as shown in table 5.9. In the “concept pair” column, the first concept in each row belongs to the *Equipment* ontology, the second concept belongs to the *Computer* ontology. The “Comment” column shows the relationship between the two concepts in the common ontology.

Table 5.9: Concept pairs in a common ontology

Index	Concepts pair	Comment	CS-MOD	Ground Truth	CS-O
1	Computer - Personal_Computer	Subsumes	0.85	0.85	0.78
2	Personal Computer – Portable Computer	Subsumes	0.85	0.85	0.67
3	Personal Computer - Server	Non- Subsumes	0.5	0.5	0.6
4	Portable Computer - Laptop	Subsumes	0.85	0.85	0.45
5	Laptop – Portable Computer	Inverse-Subsumes	0.5	0.5	0.45

Since these concept pairs share the same common part of their ontologies, users can define exact values in cases where the concepts have a subsumption relationship. Therefore, the results of CS-MOD are the same as the ground truth. In contrast, CS-O uses the four components mentioned in section 5.5.2.2 since it considers the two concepts are from different ontologies. Cases 4 and 5 are two concept pairs “Portable Computer -

Laptop” and “Laptop – Portable Computer”. CS-O produces the same concept similarity values for both cases. This is because CS-O considers concept similarity as symmetric. However, this is obviously incorrect since the similarity is applied in matching input, output, and operation components in Web services which represent how the Web services satisfy each other. CS-MOD overcomes this problem by computing them as subsumed and inverse-subsumed cases.

5.5.3.2 Influence of Domain Similarity on the Concept Similarity Measurement

Consider the *Equipment* ontology in figure 4.4a with the focus on *Mouse*, *Software*, *Hardware*, *Server* and *Operation System* concepts. Assume there are three ontologies, namely, *Animal* ontology, *Biology* ontology, and *NewEquipment* ontology. The *NewEquipment* ontology is different from the *Equipment* ontology introduced in figure 4.4a. The three ontologies can be found in Appendix A. *Animal* ontology has concept *Mouse*. *Biology* ontology has concepts *Software* and *Hardware*. The *NewEquipment* ontology has concepts *Host* and *OS*. The comparison and results are presented in table 5.10.

In the first three cases, since the concept pairs have the same names, it is tempting to assume that they have a high similarity value. However, these concepts are from different domains and actually refer to different objects in the real world. *Mouse* concept in the *Equipment* ontology refers to a device in a computer while the *Mouse* concept in the *Animal* ontology refers to a kind of animal. Similarly, *Software* and *Hardware* concepts in the *Equipment* ontology refer to components making up a computer system while they refer to different objects in the *Biology* ontology.

Table 5.10: The influence of domain on the concept similarity measurement

Index	Concepts Pair		CS-MOD	Ground Truth	CS-O
	Concept from Equipment Ontology	Concept [Ontology]			
1	Mouse	Mouse [Animal]	0.354	0.1	0.465
2	Hardware	Hardware [Biology]	0.36	0.1	0.418
3	Software	Software [Biology]	0.395	0.1	0.463
4	Server	Host [NewEquipment]	0.704	1.0	0.606
5	Operation System	OS [NewEquipment]	0.611	1.0	0.482

In contrast, it is difficult to calculate the similarities in the last two cases since *Host* and *OS* have many senses, respectively. *Host* has ten senses such as “a person who invites guests to a social event”, “a vast multitude”, “a computer that provides client stations with access to files and printers as shared resources to a computer network”, etc. One of the senses, software that controls the execution of computer programs and may provide various services, is similar to one of the meanings of “server”. Similarly, *OS* has five senses one of them being “software that controls the execution of computer programs and may provide various services”, may mean “operating system”. It is not easy to determine which senses the concepts belong to. In these cases, the domain of the ontologies is used to confirm the sense. Obviously, these concept pairs have different names but they belong to the same *equipment* domain and actually refer to the same objects in the real world.

Therefore, to measure concept similarity, the domain of the two ontologies must be involved. The results in table 5.10 show that CS-O which ignores domains, produces

values which are further from the ground truth while CS-A, which considers the domain similarity dimension, gives values closer to the ground truth. In these cases, the weights of four components: syntactic, property, neighborhood, and domain similarities are set equally. However, the higher the domain similarity is set, the closer CS-A's results is to the ground truth.

5.5.3.3 Concept pairs from different ontologies

This section deals with concepts pairs from different ontologies. Consider the *Computer*, *Hardware*, and *Software* ontologies in fig. 4.5a(i), fig. 4.5a(ii), and fig. 4.5a(iii) in figure 4.5, respectively. They are treated as different ontologies. We use the *Personal Computer* concept in the *Computer* ontology to match against the five concepts from the *Hardware*, and *Software* ontologies. Table 5.11 presents the results on matching five concept pairs.

Table 5.11: Concept pairs from different ontologies and different domains

Index	Concept [Ontology]	CS-MOD	Ground Truth	CS-O
1	Operating_System [<i>Software</i>]	0.39	0.4	0.387
2	Windows [<i>Software</i>]	0.331	0.35	0.308
3	Audio [<i>Hardware</i>]	0.432	0.35	0.41
4	Linux [<i>Software</i>]	0.35	0.35	0.334
5	Unix [<i>Software</i>]	0.341	0.35	0.321

In these cases, both CS-MOD and CS-O use four components to measure the similarity. Therefore, the results of the two algorithms are quite similar. However, CS-MOD is

slightly more accurate than CS-O since it employs the domain similarity although the domains in these cases do not influence the similarity measurement result as strongly as in the previous cases.

The CS-MOD approach is better than the CS-O when matching the “Ground Truth” in most cases. This is particularly true for the cases when the two concepts are in the same ontology since CS-MOD uses the subsume relationship to measure the concept similarity while CS-O measures concept similarity based on three components: *syntactic*, *property*, and *neighbourhood* similarity. Details of the issues were presented in section 5.5.4.

However, CS-O is slightly better in a few cases such as in the index 3 in table 5.11. These may happen when the concepts belong to two different ontologies. In the case that two concepts belong to different ontologies, the CS-MOD approach uses 4 components, namely, *syntactic*, *property*, *neighbourhood*, and *domain* similarity while CS-O approach uses 3 components, namely, *syntactic*, *property*, and *neighbourhood* similarity to measure concept similarity.

As discussed in section 5.4.4, *domain* which is also known as *context* is an important factor to calculate concept similarity since the domain affects the meaning of the concepts. The experimental results presented in section 5.5.4 show that *domain* similarity has a good impact on the overall similarity computation. However, in a few cases such as in the case (index 3) shown in table 5.11 where CS-MOD, “Ground Truth”, and CS-O have values 0.432, 0.35, and 0.41, respectively; the *domain* similarity had a negative impact on the overall similarity. The reason behind this problem is because the roots which represent the domains of the ontologies were not designed properly.

5.5.3.4 Discussion

The comparison of the 15 test results is summarized in figure 5.7. Each group in the figure consists of three columns, each of which represents a concept similarity. The first column depicts matching results of the CS-MOD algorithm; the second column represents the ground truth; the third column depicts the matching results of the CS-O algorithm.

Figure 5.7: Summary of concept similarity results

In the comparison, the focus is on the accuracy of the CS-MOD and CS-O approaches against the expected result. In the first five concept pairs, CS-MOD's results are highly similar to the ground truth since CS-MOD considers the concept pairs as being in the same ontology. On the other hand, CS-O employs four components in its comparison and

so, the results are quite far from the ground truth. In the next five concept pairs, CS-MOD has a higher accuracy since it considers domain similarity, whereas CS-O ignores the domain similarity dimension. In the last five cases, the results of the two algorithms are quite similar since both use comparable components to measure similarity. However, CS-MOD is slightly more accurate than CS-O, once again, due to the advantage of the domain similarity. In short, the examples show that CS-MOD is more accurate compared with CS-O since it produces results closer to the ground truth.

5.5.4 More Tests

Figure 5.8: Concept similarity results of 300 concept pairs and the ground truth

As there are no standard test beds for concept similarity measurement, we use ontologies which were introduced in chapter 4 for testing. Based on the ontologies, many other variations have been developed. Moreover, for testing the influence of domains, we have developed ontologies from different domains. We created 300 concepts pairs based on the developed ontologies. Ground truth was determined by SIMTech as discussed in section 5.5.3. Figure 5.8 consolidates the overall results. Assume that CS_MOD and CS_O are quantities representing the accuracy of CS-MOD's results and CS-O's results compared against the ground truths, respectively. They measure the differences between the CS-MOD results and CS-O results compared against the ground truth respectively. It is obvious that the lower the values of CS_MOD and CS_O , the more accurate the CS-MOD and CS-O algorithms, respectively.

The concept pairs are divided into three groups as in the examples but there are 125, 50, and 125 concept pairs in the first, second, and third groups, respectively. Similar to the examples, in the first group (from cases 1 to 125), CS-MOD's results are almost the same with the ground truth since CS-MOD exploits the semantic relations between concepts within the same ontology. In contrast, CS-O's results are far from the ground truth since it is based on the similarity components and ignores the semantic relations. In the second group (from cases 126 to 175), both CS-MOD and CS-O have low accurate results to compare with the first and the third groups. The reason behind this result is the test data in this group including concept pairs which have the related names but they are from totally different domains and concept pairs which have totally unrelated names but they have the same domain. This kind of test data causes mismatches for every concept matching algorithm. However, CS-MOD eliminates the mismatches by considering the

domain similarity dimension. Therefore, compared with CS-O, CS-MOD produced more accurate results. In the remaining 125 concept pairs (from cases 276 to 300), the two algorithms use similar components, and hence their results are similar in most cases. However, CS-MOD is slightly more accurate than CS-O since it employs the domain similarity although the domains in these cases do not influence the similarity result.

In general, the graph shows that CS-MOD's results are closer to the ground truth. The accuracy of the results depends on the test data. The average accuracy in the 300 cases of CS-MOD and CS-O are $_{CS-MOD} = 0.05474$ and $_{CS-O} = 0.10829$, respectively. This demonstrates that CS-MOD is more accurate than CS-O.

5.6 Summary

This chapter has introduced an algorithm to measure concept similarity. The algorithm improves the previous work by supporting concepts that are in the same ontology as well as in different ontologies. By checking the relationship between two ontologies, the similarity measurement is more accurate in the cases where the two concepts are found within a common ontology. If the two concepts do not belong to a common ontology, similarity is measured based on four components, namely, syntactic, property, neighborhood, and domain similarity. By considering the domain dimension, the proposed method has avoided mismatches. Three cases with 15 examples have been introduced to illustrate how the algorithm works. The experiment employing 300 concept pairs confirms the validity of the algorithm. The concept similarity results have been compared with Oundhakar *et al.*'s work which is the only work similar to the proposed

algorithm. The experimental results show that the algorithm is more accurate than Oundhakar *et al.*'s system in most cases.

Concept similarity plays an important role within ontology applications. In a Web service discovery system, it is employed to measure the similarity between input, output, and operation of two Web services. By employing the proposed concept similarity, the discovery system, which will be described in chapter 6, is more accurate in the cases where the two Web services use ontologies that have common parts. Furthermore, the discovery system avoids mismatch in the cases where the two concepts have similar names but are in different domains.

Chapter 6

MOD Framework

6.1 Introduction

This chapter introduces a Multi-Ontology Discovery system for Semantic Web services (termed MOD) which supports the requested and advertised Web services based on the same ontology as well as on different ontologies. The core of the discovery system is a matching process which measures the similarity between the requested and advertised Web services. The process matches components including input, output, and operation matching between two services. Since input, output, and operation are nothing more than ontological concepts, matching uses the measured similarity between two concepts. By employing the method to measure concept similarity described in chapter 5, the system has the advantage that it considers the relationship as well as domain of the two ontologies. In comparison with the similar systems, MOD has the following advantages:

- Handling the requested and advertised Web services described by the same as well as different ontologies.
- By checking the relationship between two ontologies, the accuracy and performance of the similarity measurement is improved.
- Incorporating domain improves the concept similarity measurement, thereby improving the discovery accuracy.

The algorithm is described in section 6.2 and the architecture of MOD is described in section 6.3, respectively, followed by the conclusion in section 6.4.

6.2 MOD Algorithm

6.2.1 MOD Algorithm Description

As introduced in chapter 2, a Semantic Web service includes four basic classes, namely *service*, *service profile*, *service model*, and *service grounding*. For discovery purposes, only *service profile* is used since it contains a description about the service to be discovered. A *service profile* consists of a variety of information including information for users (humans) which can be used for matching Web services manually and machine understandable information which can be used for automatic discovery of Web services. However, for automatic discovery, only input, output, and operation are employed since other information is non-essential and possibly not available. These parameters describe the functions of the service in a semantic manner through ontologies. To be discovered, a Web service provider must advertise its information in the MOD database. Advertised information includes input, output, and operation of the service. When a requester sends a request for a service, the requested service is matched against advertised Web services in the MOD database. The matching of the requested service with advertised services is carried out as in the *match()* function below.

Each advertisement is referred to as *advService*. If the advertisement service is sufficiently similar to the requested service, we add the advertised information and its degree of match to the *resultMatch*. After the requested service is matched with all

advertisements, the *sortResult* function will return a sorted list with a number of advertisements which fits the requester, based on the degree of matching. The *matchService(reqService, advService)* which matches the requested service against an advertised service is the core of the function.

```

FUNCTION match(reqService)
  resultMatch = empty_list;
  FOR all advService in MODdatabase DO
    IF matchService(reqService, advService) THEN
      resultMatch.add(advService, degreeMatch)
    ENDIF
  END FOR
  RETURN sortResult(resultMatch);
END FUNCTION

```

Figure 6.1: A schematic description of the matching algorithm between two Web services

The matching of the two services is divided into four stages namely *input*, *output*, *operation*, and *user-defined* matching. The matching algorithm is shown schematically in figure 6.1. The algorithm first checks the *user-defined* matching, where users are able to set high priority requirements in this matching. Therefore, if the matching fails, the result of matching two services also fails and the matching process terminates. Otherwise, the matching process continues with *input*, *output*, and *operation* matching. The *user-defined* stage is optional in that the user may choose not to specify any additional requirements apart from the *input*, *output* and *operation* matching. The final matching result will be composed of the *input*, *output*, and *operation* matching. The core of the matching process is the “Concept similarity algorithm” component described in figure 6.1 which was introduced in chapter 5.

Users are able to set thresholds at *input*, *output*, *operation* and final result stages. These thresholds represent the minimum similarity values accepted at each stage. As the number of matching results may be huge, the thresholds reduce the number of results. The thresholds which are cast as a real value ranging between 0 and 1, are determined in the same manner as weights described in chapter 5 (section 5.4). By default, the system does not set the thresholds. In other words, the thresholds are left as 0. If users would like to reduce the number of matching results, they may set a higher value for the threshold at one of the stages or at the final result. The details of each matching component are described in the following sections.

6.2.2 Matching Components

In matching a Web service, the provider is more interested in input matching since they would like the input of their Web service to be satisfied while the requester is more interested in output and operation matching since they expect the output and operation which they request are satisfied.

6.2.2.1 User-defined matching

The users can declare constraints or rules to restrict the matching and increase the accuracy of the results of matching. For example, a requester wishing to buy a computer describes a Web service with configuration of the computer as input and price as output. They may also declare more constraints relating to the computer manufacturer. For example, they may prefer not to buy from a particular vender. These constraints or rules will be matched against the advertisement. Users may decide not to define any constraints or they may define many constraints. We assume that the result of a constraint matching is either “match” or “not match”. It is possible to define the distance of this matching but this is outside the scope of the work.

6.2.2.2 Input matching

In the input matching, we determine how inputs of the advertised services are satisfied by the inputs of the requested services. The input matching operates as follows: for each input of the advertised service, the algorithm tries to find an input of the requested service that is the most similar (the highest degree match). If none of the advertised input can match with the requested input, the input matching returns *fail*. Thus, the matching of the two Web services will fail. Similar to other matching components, the core of input

matching is concept similarity which was introduced in chapter 5, represented by the “concept similarity algorithm” in figure 6.1. Therefore, the algorithm first checks if the two concepts belong to a common part of the two ontologies. If they are, the similarity is measured as if they belong to the same ontology. Otherwise, it is measured as a combination of the four components: syntactic, property, neighborhood, and domain similarity. The input matching algorithm is elaborated as follows:

```

FUNCTION inputMatching (inputAdv, inputReq){
  IF (inputAdv is empty) THEN RETURN 1; //exact match END IF
  degreeInput = 0; //degree of input matching
  numInput = 0; //number of input of the advertised service
  FOR all inputAdvElement in inputAdv DO
    numInput = numInput +1;
    degreeMax =0;
    FOR all inputReqElement in inputReq DO
      IF (two ontologies are the same) THEN
        (degreeMatch = similarity which is computed similarity on
        the same ontology)
      ELSE
        (degreeMatch = similarity which is computed based on
        different ontologies)
      END IF
      IF (degreeMatch > degreeMax) THEN degreeMax=degreeMatch END IF;
    END FOR
  END FOR
  IF degreeMax = 0 THEN RETURN fail;
  degreeInput = degreeInput + degreeMax;
END FOR
degreeInput = degreeInput/numInput;
IF degreeInput < threshold THEN RETURN fail;
ELSE RETURN degreeInput;
END IF
END FUNCTION

```

The number of inputs of the advertised and requested services may be none or many. When the advertised service does not contain any input, it means that the provider does not need any input. In other words, any input of the requester is satisfied. Therefore, the input matching result is an exact match. *inputAdvElement* and *inputReqElement* denote elements of the input list of advertised and requested service, respectively. *degreeMatch* is the degree of the matching between an input pair (an input of advertised service and an input of the requested service). *degreeMax* is the maximum degree of an input of advertised service with every input of the requested service. In other words, *degreeMax* is the maximum of *degreeMatch* for every input of advertised service. *degreeInput* is the final result of input matching. *degreeInput* will be compared against an input threshold whose value depends on a specific domain. If it is less than the input matching threshold, then the matching fails. Otherwise, the matching returns the *degreeInput*. The input matching threshold which represents the minimum similarity expected by the users is set as discussed earlier in section 6.2.1.

6.2.2.3 Output, Operation, and Final Result Matching

Both output and operation processing are carried out in the same manner as input. However, in output matching, all output from the requested service will be matched with each output in the advertised service. In operation matching, service category operation of the requested and service category operation of the advertised service are matched. The results of output matching and operation matching are the degree of output matching and operation matching, respectively. The final matching result is a combination of the input, output, and operation matching as shown in equation (2).

$$S = \frac{(w_1 * inputMatch + w_2 * outputMatch + w_3 * operationMatch) * userMatch}{w_1 + w_2 + w_3} \quad (2)$$

where w_1 , w_2 , and w_3 are weights defined by users. A weight is defined based on how important and how confident the component is to the users. Weights are assigned based on the method described in chapter 5 (Section 5.4), indicating the ratio of importance between matching components. By default, the three values carry equal importance. After matching with all advertisements from the database, the system will sort the matched Web services by the degree of similarity and return the matched list to the requester.

6.2.3 Complexity of the Algorithm

Similar to chapter 4, we assume the sizes of the two ontologies are m and n , respectively. The maximum numbers of properties in concepts of the two ontologies are p and q , respectively. The number of advertised web services is z . The *match()* function given in section 6.2.1 is re-written as follows without changing the gist of the solution in order to measure the time complexity.

```

FUNCTION WebServiceDiscovery()
    FOR i = 1 to z DO // z times for matching with z advertised Web
                        //services
        OntologyComparison() // (m*n*p*q*k);
        InputMatching(); //O(g*h)
        OutputMatching(); // O(g*h)
        OperationMatching(); // O(g*h)
    END FOR
END FUNCTION

```

The complexity of the function OntologyComparison which is presented above is

$(m*n*p*q*k);$

The three functions *InputMatching*, *OutputMatching*, and *OperationMatching* are similar and therefore they have the same complexity. So, we measure only *InputMatching*:

```

FUNCTION InputMatching()
    SyntacticMatching(); //  $O(g*h)$ ;  $g$  and  $h$  are the sizes of the two
                        //strings which are used in concept descriptions.
    PropertyMatching(); //  $O(g*h)$ ;  $g$  and  $h$  are the sizes of the two
                        //strings which are used in property descriptions.
    NeighborhoodMatching(); //  $O(g*h)$ ;  $g$  and  $h$  are the sizes of the
                        //two strings which are used in concept
                        //(neighbourhood concept) descriptions.
    DomainMatching(); //  $O(g*h)$ ;  $g$  and  $h$  are the sizes of the two
                        //strings which are used in //two root descriptions.
END FUNCTION

```

The four function *SyntacticMatching*, *PropertyMatching*, *NeighborhoodMatching*, and *DomainMatching* are eventually string matching algorithms. Therefore, their complexity is $O(g*h)$ where g and h are the size of the two strings.

The complexity of the web service discovery system is $(\max((m*n*p*q*k), (g*h)))$. Comparing $O(g*h)$ and $(m*n*p*q*k)$, the latter is much larger. As a consequence, the complexity of the discovery system is $(z*m*n*p*q*k)$.

As discussed in chapter 4, normally, the complexity of an algorithm reflects the time performance of the algorithm. However, it should be noted that the algorithm depends much on external factors such as the size of the ontologies, the network speed, and power of the server in which the ontologies are stored.

6.3 MOD Engine

Figure 6.2: MOD Engine

This section introduces the engine of MOD which is presented in figure 6.2. Each component is referenced by a number which represents the order of invoking the component. The “Service Profile Parser” component (1) parses the requested and advertised *service profiles*. The “Store Functional Property” component (2) stores input,

output, and operation of a *service profile* to MOD database. The “Loading Ontology” component (3) loads the corresponding ontologies based on a selected advertised Web service. The “Comparing two Ontologies” component (4) compares two given ontologies based on the algorithm introduced in chapter 4. The “Measuring Concept Similarity” component (5) computes concept similarity based on the algorithm introduced in chapter 5. The “Matching Web Services” component (6) matches Web services as described in section 6.2. The “Sorting Result” component (7) sorts the matching results based on their degree of similarity. The operation of the engine is described as follows.

- As a starting point, a Web service provider advertises its information to the system. Its *service profile* will be parsed by “Service Profile Parser” to obtain the advertised information. Similarly, when a Web service requester makes a request, its *service profile* will be parsed by “Service Profile Parser” to obtain the requested information.
- Advertised information of Web service providers needs to be stored in MOD database. As mentioned, a *service profile* consists of a variety of information, but only input, output, and operation are involved in discovery. Therefore, “Store Functional Property” only stores the information of the profile into the MOD database.
- A user may choose to match against a specific advertised service or perform discovery against all advertised services in the MOD database. The ontology of the requested Web service and advertised Web service are loaded from “Service Profile Parser” and “Load Ontology”, respectively.

- The two ontologies are checked whether they possess common parts by “Comparing two Ontologies”.
- Based on the result of “Comparing two Ontologies”, “Measuring Concept Similarity” will determine whether the ‘same ontology’ algorithm or the algorithm measuring concepts in different ontologies is to be used.
- “Matching Web Services” uses the result of “Measuring Concept Similarity” to perform Web service matching, which includes input, output, operation, and user-defined matching.
- The results of the discovery will be sorted based on degree of similarity by “Sorting Result”. Finally, these results are displayed.

6.4 Summary

This chapter has introduced a discovery system termed MOD which matches Web services based on the same as well as on different ontologies. The algorithm is divided into four stages, namely, input, output, operation, and user-defined matching. The system has improved existing work in supporting multi-ontology discovery, enhancing the accuracy by comparing two ontologies, and eliminating mismatches by considering domain similarity. These advantages are obtained by employing the method to measure concept similarity in chapter 5. Web service matching examples and testing will be introduced in chapter 7. The examples illustrate how the Web service matching is carried out and the testing confirms the validity of the system.

Chapter 7

MOD Application and Experiment

7.1 A Supply Chain Scenario

A scenario in the e-supply chain domain is used to illustrate how MOD locates companies that supply computers and computer components. The business operation is described in figure 7.1 and is elaborated upon as follows:

- Assume customers are end-users who want to buy computers or computer components for their specific applications. They send their purchase order to a *Computer Manufacturer Corporation* (CMC) which plays an intermediary role between these customers and some suppliers. For simplicity, we assume that customers have already completed a Request for Quotation. A purchase order, which may be sent by email or facsimile, contains information about the computers or components and the prices of the products which they want to buy. Customers can obtain the price at the CMC Web site. The customers do not know anything about MOD or suppliers of computer components. Examples of such customers are Multi-National Corporations, laboratories, research institutions, etc.
- CMC does not produce any computer components but purchases these components from other companies in order to meet the customers' requests. After receiving the purchase order from customers, CMC system has a service called "generate Web

service profiles” which utilizes the information in the purchase order and the *local ontologies* which were developed by CMC. The Web service profile is used to locate partners who provide computer components by using MOD.

- Computer component suppliers advertise their Web services through MOD. These services contain information about the products which they want to sell. An advertised Web service includes input, output, and operation of the *service profile* which were created based on their own ontologies. These ontologies are typically different from *local ontologies* which were created by CMC, since the ontologies were created independently of each other. However, these ontologies may have common parts if they were created based on existing ontologies. In some cases, the ontologies may be the same since CMC and the suppliers may adopt existing ontologies.
- When MOD receives the advertised information from the suppliers, it will index and store the information. Upon a request from CMC, MOD searches suitable suppliers and sends a sorted list of their URLs, based on the degree of their similarity against the request to CMC.
- CMC will then invoke the services which are provided by component companies.

Hence, the transaction occurs automatically.

In the absence of the MOD system, CMC must locate the component suppliers manually.

In this case, the following difficulties arise:

1. CMC must know the computer component suppliers before the business transactions take place. This leads to a difficulty for a new company entering the

- market as it is unrealistic to know every supplier in the world. The scenario assumes that MOD has a crawler to collect new Web services and index them to its database.
2. The relationship between CMC and its business partners would be static. The business operations are therefore not dynamic and flexible. There are problems if either the suppliers or CMC change their business model since either party would need to modify their mode of operation. In the scenario, a company can easily locate another partner if a current partner changes its business model since the relationship is dynamic and flexible.
 3. Business operations are impeded due to the manual set up required before transactions between CMC and its business partners can take place.

Using MOD avoids the above difficulties. As a result, the business is more flexible and efficient.

Figure 7.1: A supply chain model using MOD

7.2 Prototype and Oundhankar *et al*'s System

7.2.1 MOD Prototype

As part of an experiment, an administrator function is used to manage the database of advertised service in MOD. Through this administrator function (figure 7.2), advertised Web services can be added, deleted, modified, and viewed.

Figure 7.2: Administrator Interface

The User Interface in figure 7.3 is used to capture requested service requirements. Thresholds are placed at each stage of matching as well as weights for input, output, and operation matching. These thresholds and weights were described in chapter 5. There are

also options to choose “Matching with selected service” to obtain the best-fit advertised service or “Web service discovery” to match with all the advertised services in the MOD database.

Figure 7.3: User Interface

7.2.2 Discovery System by Oundhankar *et al*

As mentioned in chapter 3, only one existing discovery system, namely METEOR-S Web Service Discovery Infrastructure, is similar to MOD in handling matching Web services

based on the same ontology as well as different ontologies [75]. The matching is based on matching input, output, and operation of Web services which employed concept similarity algorithm similar to CS-O described in section 5.5.2. To compare with Oundhankar *et al.*'s system, we have developed a system, namely, MOD-O which uses the CS-O algorithm to measure concept similarity. MOD has the following differences when compared with MOD-O:

- Matching Web services: Both systems use input, output, and operation for matching Web services. However, MOD allows users to intervene in the matching process. For example, users are allowed to restrict the matching results by placing a threshold at each matching stage or by using user-defined matching component.
- Concept similarity measurement: At the core of the Web service matching process, MOD and MOD-O employ concept similarity components CS-MOD and CS-O respectively. These two approaches were introduced in section 5.5.3. As discussed, CS-MOD checks if two concepts belong to the same common ontology and uses domain similarity to reduce mismatch. Hence, MOD possesses these advantages as well.

In the examples and the rest of the testing, the results of MOD and MOD-O are compared against the “ground truth” which is created by the same manner as introduced in section 5.5.3.

7.3 Experiments

This section introduces Web service matching examples and experiments. Examples with 3 Web service pairs are presented to illustrate how the algorithm works. Experiments with 500 Web service pairs were carried out. The purpose of presenting examples is to illustrate how the matching is carried out while the testing confirms the validity of the MOD system. The section first starts with ontologies used in the examples and experiments.

7.3.1 Sample Ontologies Used

Figure 7.4: Ontology in eBusiness domain

We assume that there are three companies: IBM is a computer company which buys *special components* to produce *computers*. Components include high speed CPU, or mass

storage, etc. Intel and AMD are examples of companies providing these components. The component manufacturers describe the services offered using an ontology.

Figure 7.4 describes an ontology in an eBusiness domain which provides Sell and Buy services. The Buy object in turn consists of *BuyComputer*, *BuyComponent*, and *BuyCamera*. *BuyComponent* provides *BuyCPU*, *BuyMemory* and *BuyStorage*. This is a simple example of an ontology which is used in the *operation* of Web services which will be described in figures 7.7 and 7.9.

Figure 7.5 describes an ontology in the *Computer Component* domain which includes software and hardware components. The hardware contains memory, CPU, and storage. This ontology is used by the output of the services which are described in figure 7.7.

Figure 7.5: Ontology for computer device domain

Business transactions involve monetary exchange. Hence, we need to define the concept of price. Figure 7.6 describes a simple example of an ontology that has some concepts used to compute prices. The price concept has two data-type properties, namely, *hasValue* and *hasCurrency* which refer to the actual price and the specific currency,

respectively. We introduce concepts *Lessthan400*, *Lessthan300*, *Lessthan200*, and *Lessthan100* that refer to prices that are less than 400 units, 300 units, 200 units, and 100 units, respectively.

We assume that the *serviceProfile* of the Web services has *hasPrice* as input, *hasProduct* as output. *hasPrice* is the input parameter of the Web services and described by concepts in the Price ontology (figure 7.6). *hasProduct* is the output parameter of the Web services and described by concepts in the computer component ontology (figure 7.5). The operation uses concepts in *eBusiness* ontology (figure 7.4).

Figure 7.6: Ontology for price domain

Figure 7.7 describes a requested service and advertised services. IBM wishes to buy CPU with price less than 200. It defines a requested service as shown in figure 7.7a. Intel is a provider which sells CPUs. It defines the service for their trading partners (figure 7.7b).

Similarly, AMD is a provider which sells computer devices. The company develops a service for its partners who want to buy devices as described in figure 7.7c.

Figure 7.7: Requested and advertised services

7.3.2 Examples

This section first introduces two cases of matching Web services which illustrate how MOD operates. Firstly, the requesters and providers use the same ontology. Secondly, the requesters and providers use different ontologies, followed by the experiments.

7.3.2.1 The Requester and Providers Share the Same Ontology

In this situation, the requesters and providers share the same ontology to describe their services. Thus, when IBM looks for business partners, there are two advertised Web

services from Intel and AMD that are potentially able to satisfy its requirement. They are matched as follows:

Matching with Intel:

- Input matching.** This matching component measures how the input of the advertised service is satisfied by input of the requested service. In this case, it measures how the input of Intel's Web service is satisfied by the input of IBM's Web service. IBM's Web service and Intel's Web service are represented by the two concepts *Lessthan200* and *Lessthan300* in the price ontology, respectively. The satisfaction is actually the similarity between two concepts *Lessthan200* and *Lessthan300*. As described in chapter 5 (section 5.3.2), this similarity is "Inverse-Subsumes" since *Lessthan200* is a direct sub-concept of *Lessthan300*. MOD uses CS-MOD resulting in a value 0.5 while MOD-O uses CS-O resulting in a value 0.936. The value 0.5 is more reasonable than 0.936 since it expresses that the provider would like to sell the product with the price is 300 units of currency while the requester wishes to buy the product with the price is only 200 units.
- Output matching.** The output in Intel's Web service and IBM's Web service are represented by the two concepts *CPU Intel* and *CPU* in the *Computer Component* ontology, respectively. The satisfaction in this case is the similarity between the two concepts *CPU Intel* and *CPU*. The case is "Inverse-Subsumes" since *CPU Intel* is a direct sub-concept of *CPU*. MOD uses CS-MOD resulting in a value 0.5 while MOD-O uses CS-O resulting in a value 0.564. The value 0.5 is more reasonable than 0.564 since the requested service would like to buy a

general CPU which can be any kind of CPU while the advertised service only offers *CPU Intel*.

- **Operation matching.** The operation of Intel's Web service and IBM's Web service are represented by the same concept *BuyCPU* in the *eBusiness* ontology. The satisfaction in this case is an "Exact". Therefore, both MOD and MOD-O result in the same value 1.0.
- **User-defined.** The user can define more rules or constraints to restrict the results. In this example, we assume that the user-defined matching is satisfied. Consequently, the user-defined matching results in a value 1.

The final similarity of the two Web services is calculated by the equation (2) introduced in chapter 6 (section 6.2). In this case, each weight w_i is set as 0.5 as we assume that each of the three matching components is equally important. The final similarity is computed as the following equation:

$$S = \frac{0.5 * inputMatch + 0.5 * outputMatch + 0.5 * operationMatch}{1.5}$$

Therefore, the final similarities of the two Web services which are returned by MOD and MOD-O are 0.667 and 0.834, respectively. Since the input, output, and operation matching occur in the same ontology, they result in values which are the same as the expected values, namely, "Inverse-Subsumes" and "Exact". In other words, the "ground truth" in this case should be 0.666 and it is the same with MOD's result.

Matching with AMD:

The matching with AMD is carried out similarly to the matching with Intel:

- **Input matching.** The case is “Subsumes” since the input of IBM’s Web service which is represented by the concept *Lessthan200* is more general than the input of AMD’s Web service which is represented by the concept *Lessthan100* in the *price* ontology. Consequently, MOD results in a value 0.85 while MOD-O results in a value 0.763.
- **Output matching.** The case is “Subsumes” because the output of AMD’s Web service which is represented by the concept *Hardware* is more general than the output of IBM’s Web service which is represented by the concept *CPU* in the *Computer Component* ontology. Consequently, MOD results in a value 0.85 while MOD-O results in a value 0.758.
- **Operation matching.** The case is “Subsumes” because AMD service operation is more general than IBM service operation. MOD results in a value 0.85 while MOD-O results in a value 0.94, respectively.
- **User-defined.** The user can define more rules or constraints to refine the results. In this example, we assume that the user-defined matching is satisfied. Therefore, the user-defined matching results in a value 1.

Therefore, the final similarities of the two Web services which are returned by MOD and MOD-O are 0.85 and 0.821, respectively. Similar to the Web service pair IBM and Intel, since the input, output, and operation matching occur in the same ontology, they result in

values which are the same with expected values, namely, Subsumes. In other words, the “ground truth” in this case should be 0.85 and it is the same with MOD’s result.

Since the similarity between Intel’s Web service and IBM’s Web services is 0.666 and the similarity between AMD’s Web service and IBM’s Web services is 0.85, we can conclude that AMD is the provider offering a service which satisfies IBM’s requirement more than Intel. Moreover, the two examples also demonstrate that MOD is more accurate than MOD-O in the cases Web service using the same ontology.

7.3.2.2 The Requester and Providers use Different Ontologies

In this situation, the requester and providers use different ontologies to describe their services. For simplicity, we assume that the *Price* and the *eBusiness* ontologies remain shared by both requester and provider. However, the requester is unaware of the computer device ontology (figure 7.5). Instead, they develop a “Hardware device” ontology as shown in figure 7.8.

We assume that Toshiba is another computer manufacturing company which has developed the requested service based on the ontology shown in figure 7.8. Toshiba’s request service is described in figure 7.9.

Figure 7.8: Ontology for computer hardware device domain

Figure 7.9: Toshiba request service

There are two advertised Web services from Intel and AMD that are potentially able to satisfy its requirements. Matching is carried out as follows:

Matching with Intel

- **Input matching.** The case is “Inverse-Subsumes” because the input of Intel’s service is more general than Toshiba’s service. In this case, MOD results in a value 0.5 while MOD-O results in a value 0.984.
- **Output matching:** Since the two concepts used to describe the inputs of the two Web services are from different ontologies, the semantic similarity of the two concepts is computed using the algorithm in section 5.4. MOD results in a value 0.68 while MOD-O results in a value 0.53.
- **Operation matching:** The result is an “Exact” match since both operations point to the same concept “BuyCPU” in the eBusiness ontology. In this case, both MOD and MOD-O result in the same value 1.0.
- **User-defined:** The user can define more rules or constraints to restrict the results. In this example, we assume that the user-defined matching is satisfied. So, the user-defined matching results in a value 1.

Finally, MOD results in a value 0.726 while MOD-O results in a value 0.838. Based on input, output, and operation of the two Web services, the “ground truth” in this case should be around 0.75. Therefore, MOD is slightly more accurate than MOD-O.

7.3.3 More Tests

As there is no standard test data or benchmark available, we developed 100 Web service profiles based on the scenario. The ontologies and 300 concept pairs which were used for testing in chapter 5 were employed in creating the Web service profiles. The profiles were created in the same manner as in creating the examples in section 7.3.2. The ontologies and Web service profiles are available at: <http://www.ntu.edu.sg/home5/LEDU0001/WSTestdata/>. Using these 100 profiles, we ran the test data for five times and each time chose a profile as a requester, the remaining Web service profiles were treated as providers. In this way, we obtained 500 Web service profile pairs.

Assume that MOD and $MOD-O$ measure the difference between the ground truth and MOD's results and MOD-O's results, respectively. Hence, they are quantities representing the accuracy of MOD's results and MOD-O's results. Similar to the examples, the MOD's results are accurate and almost the same with the ground truth for pairs of Web services based on common ontologies. The results are less accurate for pairs of Web services based on different ontologies and do not have common parts. The average accuracy in the 500 pairs of MOD and MOD-O are $MOD = 0.030435$ and $MOD-O = 0.094495$, respectively. This demonstrates that MOD is more accurate than MOD-O.

MOD allows users to restrict the matching results by placing thresholds on the matching stages or at the final result stage. For example, users wishing only to have an “average” for the output matching stage may set a value 0.5 for the output threshold. Consequently, only 82 Web service profile pairs among the 500 pairs satisfy the threshold. In another example, users set the final similarity to be “high” in order to eliminate the low similarity matching profile pairs. Thus, they set a threshold of 0.8. In this case, 21 profile pairs satisfy the threshold.

7.4 Summary

The chapter has presented a scenario, examples, and an experiment for the MOD algorithm and engine introduced in chapter 6. The scenario involved an e-supply chain in the computer components domain. MOD is employed to locate companies that supply computers and components. The examples have been presented to illustrate how the algorithm works. The experiment which has been performed with 500 Web service profile pairs confirms the validity of the MOD system.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Web services provide a foundation to e-business systems. They enable business operations to function more efficiently via the Web and enhance business opportunities to companies. Moreover, they allow companies to reduce cost by fast, effective, and reliable services to customers, suppliers, and partners over the Internet. As a result, a large number of Web services have been developed recently. The rapid development has led to a demand for a discovery mechanism. The thesis has presented a survey of Web service discovery systems, concluding that current discovery systems have the following shortcomings. Firstly, they do not support matching Semantic Web services that use different ontologies. Secondly, they do not support matching a mix of Semantic Web services and non-Semantic Web services. Finally, they do not support matching Semantic Web services using different Web service description languages.

Amongst the above problems, matching Web services using different ontologies is the most urgent. In the real world, a Web service provider can provide an exact service to the requester even though both the services use different ontologies. Therefore, a discovery system that supports Web services using different ontologies is important. Hence, in the thesis, our work focused on the matching of Web services using different ontologies. To

solve this problem, we have developed MOD, a Multi-Ontology Discovery system. The contents of the thesis as well as the major contributions achieved are summarized as follows:

- *Ontology Comparison.* Since ontologies which are used by different organizations may be related, there is a need to check these relationships. Chapter 4 introduced an algorithm that compares two ontologies to determine their relationship, namely, the same ontology, sub ontology, common ontology, or different ontologies. The relationship is used in computing concept similarity. It should be noted that the ontology comparison algorithm can be applied in many ontology applications related to *ontology mining, mapping, merging, integration, and alignment.*
- *Concept Similarity Computation.* In order to promote ontology use, there are many operations that must be supported. Computing the semantic similarity between concepts is the core of these operations. Chapter 5 presented an algorithm for measuring the similarity. It improves previous work by supporting multi-ontology concept measurement and includes the domain similarity dimension into the measurement. Moreover, by employing the ontology comparison algorithm in chapter 4, the semantic similarity algorithm is more accurate. Web services composition has received much interest in supporting business-to-business or enterprise application integration. As a result, researchers have addressed the issue of Web service composition [8,71,120]. Measuring concept similarity is a core of the composition activities. Hence, the proposed concept similarity measurement can be applied to enhance the composition.

- *Multi-Ontology Web Service Discovery System.* Chapter 6 introduced a Multi-Ontology Discovery system for semantic Web services (termed MOD). The system supports Web services which are based on the same ontology as well as on different ontologies. It is an improvement on current discovery systems which are adequate only when the Web service requester and provider use the same ontology. The core of the system is the Web service matching process which is divided into four stages, namely, *input*, *output*, *operation*, and *user-defined* matching. By defining the process into four stages, users can intervene and fine-tune the matching process. The experiments that have been carried out show the viability of the system.
- *A Survey of Web Service Discovery Systems.* Since many Web service discovery systems have been developed, a survey of Web service discovery systems is needed to explore existing techniques and to highlight the advantages and disadvantages of each system. Chapter 3 presented a survey of Web service discovery systems. A taxonomy of approaches has been proposed to classify the systems. The classification was based on the usage of semantics. The advantages and disadvantages of each system were highlighted. Three shortcomings of current Web service discovery systems were pointed out. We also elaborated on open issues relating to such discovery systems.
- *A Benchmark for Web Service Discovery systems.* There is a lack of benchmarks that has led to a difficulty in comparing Web service discovery systems. An e-supply chain scenario involving the purchase of equipment was introduced and Web service profile pairs based on the scenario have been developed.

8.2 Future Work

Even though the system has been developed successfully and contributions have been achieved as mentioned above, the work can be extended in three dimensions. They are: 1) Supporting Web services using different description languages as well as non-semantic Web services. 2) Applying fuzzy logic to the system to overcome the mismatches arising from different ontology designs. 3) Applying the concept similarity measurement approach to improve the ontology operations.

8.2.1 Supporting Non-Semantic Web Services and Different Description Languages

We have explored three issues of current Web service discovery systems. We have developed MOD to discover Web services using different ontologies since this issue is important and has the highest research potential. The second and the third issues, namely, matching Semantic Web service and non-Semantic Web service, and matching Semantic Web services that use different description languages, should be considered. The details of these issues have been discussed in chapter 3.

8.2.2 Applying Fuzzy Logic

The following terms *Exact*, *Subsumes*, *Inverse-Subsumes*, and *Fail* are commonly used to denote the degree of matching when a concept pair has a subsumption relation. The use of these terms is crucial. The most challenging aspect is that they are open to different

interpretations. Moreover, matching of Web services can only be carried out if the above terms are defined numerically.

The Web service discovery algorithm proposed by Paolucci *et al.* [76] attempts to classify the Web service similarity into one of the following terms: *Exact*, *Plug-in*, *Subsumes*, or *Fail*. This system cannot return a rank match since these terms are not defined numerically. To overcome it, Said *et al.* [92] proposed similar terms namely, *Exact*, *Subsumes*, *Inverse-Subsumes*, *Fail* and mapped these terms into integers. For example, *Exact* = 4, *Subsumes* = 3, *Inverse-Subsumes* = 2, and *Fail* = 1. However, crisp values present several problems including their derivation. For example, when the similarity is 3.3, is it exact or is it subsume? TUB [62] also defined these terms using crisp values. However, crisp values cannot capture vague and uncertain meaning associated with the terms.

Fuzzy logic theory provides a means of representing uncertainties and vagueness. It is an appropriate approach for modeling the kind of uncertainty associated with vagueness and imprecision, and with a lack of information of a problem. Therefore, fuzzy logic may be applied to address this problem. As mentioned in chapter 3 (section 3.1.1), attempts have been made to apply fuzzy logic to Semantic Web and Web service discovery. However, their focus was on applying fuzzy logic to enhance the expressive ability of the ontology language. In contrast, we propose applying fuzzy logic to match Web services. *Exact*, *Subsumes*, *Inverse-Subsumes*, and *Fail* which are represented in figure 8.1 are fuzzy values instead of crisp values.

Figure 8.1: Fuzzy numbers

After transformation, Web service similarity measured by equation (2) in chapter 6 is transformed to the following fuzzy function equation:

$$fWSMatch = (x)*Exact + (y)*Subsume + (z)*Invert-Subsumes + (t)*Fail$$

where x,y,z, and t are obtained during the transformation and range between 0 and 1.

Web service similarity is obtained by defuzzifying the above function.

8.2.3 Applying the Concept Similarity Measurement to Ontology Operations

As discussed in chapter 5, the approach of concept similarity can be further employed in ontology operations. Ontology integration and evolution are operations that can employ the approach. Ontology integration is the process of building an ontology in a domain by using one or more ontologies. Ontology evolution is a process that traces the changes of ontology with time. In both the operations, measuring concept similarity is a fundamental activity. Future work that applies the concept similarity measurement approach to the mentioned operations is likely to result in improving existing technologies.

In conclusion, the thesis has presented the achievements of a new Web service discovery system. The proposed approach is general and therefore can be used for Web services using different Web service description languages such as OWL-S, DAML-S, and RDFS. Each of these languages uses different reasoners. The future work described above shows that there is potential work in the Web service discovery area which is both interesting and challenging.

Bibliography

- [1] Abolhassani, H.; Hariri, B. B.; and Haeri, S. H., *On Ontology Alignment Experiments*, (2006), [Electric version] Vol. 3, 1735-188X. Available at: <http://www.webology.ir/2006/v3n3/a28.html>.
- [2] Angell, R. C.; Freund, G. W.; and Willett, P., Automatic spelling correction using a trigram similarity measure, *Information Processing and Management*, **19**(4), pp 255, 1983.
- [3] Bayardo, R.; Bohrer, W.; Brice, R.; Cichocki, A.; Fowler, G., *et al.*, InfoSleuth: agent-based semantic integration of information in open and dynamic environments, *Proceedings of the 1997 ACM International Conference on the Management of Data (SIGMOD)*, Tucson, Arizona, United States, ACM Press, pp 195, 0-89791-911-4, 1997.
- [4] Benatallah, B.; Hacid, M. S.; Leger, A.; Rey, C.; and Toumani, F., On automating Web services discovery, *The VLDB Journal — The International Journal on Very Large Data Bases*, **14**(1), pp 84, 1066-8888, Springer-Verlag New York, 2005.
- [5] Berners-Lee, T.; Hendler, J.; and Lassila, O., The Semantic Web, *Scientific American*, **284**(5), pp 34, 2001.
- [6] Box, D.; Ehnebuske, D.; Kakivaya, G.; Layman, A.; Mendelsohn, N., *et al.*, SOAP - Simple Object Access Protocol, W3C: (2000). Available at: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.
- [7] Bray, T.; Paoli, J.; Sperberg, M.; Maler, E.; and Yergeau, F., Extensible Markup Language (XML) 1.0 (Third Edition), W3C: (2006). Available at: <http://www.w3.org/TR/REC-xml/>.

- [8] Cardoso, J.; and Sheth, A., Semantic e-Workflow Composition, *Journal of Intelligent Information Systems*, **21**(3), pp 191, 0925-9902, Kluwer Academic Publishers, 2003.
- [9] Carroll, J. J., Matching RDF graphs, *First International Semantic Web Conference on The Semantic Web*, Sardinia, Italy, Springer-Verlag, pp 5, 3-540-43760-6, 2002.
- [10] Castillo, J. G.; Trastour, D.; and Bartolini, C., Description Logics for Matchmaking of Services, *Paper presented at the Workshop on Application of Description Logic*, CEUR, 2001.
- [11] Chao, K. M.; Younas, M.; Lo, C. C.; and Tan, T. H., Fuzzy Matchmaking for Web Services, *WAMIS Workshop*, Tamkang University, Taiwan, IEEE CS, **2**, pp 721, 2005.
- [12] Christensen, E.; Curbera, F.; Meredith, G.; and Weerawarana, S., WSDL - Web Services Description Language, W3C: (2001). Available at: <http://www.w3.org/TR/wsdl>.
- [13] Cohen, W. W., Data integration using similarity joins and a word-based information representation language, *ACM Transactions on Information Systems*, **18**(3), pp 288–321, 2000.
- [14] Cohen, W. W.; Ravikumar, P.; and Fienberg, S. E., A Comparison of String Distance Metrics for Name-Matching Tasks, *Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, Acapulco, Mexico, pp 73, 2003.

- [15] DAML Organization, The DARPA Agent Markup Language. Available at: www.daml.org.
- [16] DAML Organization, OWL-S markup of services. Available at: www.daml.org/services/owl-s/.
- [17] DAML Organization, Profile of OWL-S. Available at: <http://www.daml.org/services/owl-s/1.0/Profile.owl>.
- [18] David, W. F., IDL Programming Techniques, Fanning Software Consulting; p 445, 0-9662383-2-X, 2nd Edition, November, 2000.
- [19] Doan, A. H.; Madhavan, J.; Dahamankar, R.; Domingos, P.; and Halevy, A., Learning to match Ontologies on the semantic web, *The VLDB Journal*, **12**(4), pp 303, 1066-8888, Springer-Verlag New York, 2003.
- [20] Doan, A. H.; Madhavan, J.; Domingos, P.; and Halevy, A., Ontologies Matching: A Machine Learning Approach, In *Handbook on Ontologies in Information Systems*, S.Staab; Studer, R. Eds.; Springer-Velag, 2003.
- [21] Ehrig, M.; and Sure, Y., Ontology Mapping - an integrated approach, *1st European Semantic Web Symposium*, Heraklion, Greece, 2004.
- [22] Euzenat, J.; Bach, T. L.; Barrasa, J.; Bouquet, P.; Jan De Bo, R., *et al.*, State of the art on ontology alignment, *Knowledge web NoE*, 2004.
- [23] Euzenat, J.; and Valtchev, P., An integrative proximity measure for ontology alignment, *ISWC-2003 workshop on semantic information integration*, Sanibel Island (FL US), pp 33–38, 2003.

- [24] Farquhar, A.; Fikes, R.; and Rice, J., The Ontolingua Server: A Tool for Collaborative Ontology Construction, *International Journal of Human-Computer Studies*, **46**(6), pp 707, 1071-5819, Academic Press, Duluth, MN, USA, 1997.
- [25] Fensel, D.; Decker, S.; Erdmann, M.; and Studer, R., Ontobroker in a Nutshell, *European Conference on Digital Libraries*, Crete, Greece, pp 663, 1998.
- [26] Fowler, J.; Perry, B.; Nodine, M.; and Bargmeyer, B., Agent-based semantic interoperability in Infosleuth, *Sigmod Record*, p 28, 1999.
- [27] Frolund, S.; and Koistinen, J., QML: A Language for Quality of Service Specification, *Technical report HPL on HP Laboratories*, 1998.
- [28] Gagnes, T.; Plagemann, T.; and Munthe-Kaas, E., A Conceptual Service Discovery Architecture for Semantic Web Services in Dynamic Environments, *The 22nd International Conference on Data Engineering*, Georgia, USA, p 74, 2006.
- [29] Garofalakis, J.; Panagis, Y.; Sakkopoulos, E.; and Tsakalidis, A., Web Service Discovery Mechanisms: Looking for a Needle in a Haystack?, *International Workshop on Web Engineering, in conjunction with ACM Hypertext 2004*, Santa Cruz, California, USA, 2004.
- [30] Genesereth, M. R.; and Fikes, R. E., Knowledge Interchange Format - Version 3 - Reference Manual, *Stanford University, Logic Group, Logic-92-1, Stanford, CA*, 1992, 2002.
- [31] GICL, OWLJessKB: A Semantic Web Reasoning Tool, (2005). Available at: <http://edge.cs.drexel.edu/assemblies/software/owljesskb/>.

- [32] Giunchiglia, F.; and Shvaiko, P., Semantic matching, *The Knowledge Engineering Review*, **18**(3), pp 265, 0269-8889, Cambridge University Press, New York, NY, USA, 2003.
- [33] Gruber, T. R., A Translation Approach to Portable Ontology Specifications, Academic Press; 1993.
- [34] Haarslev, V.; and Möller, R., RACER System Description, *the First International Joint Conference on Automated Reasoning*, Springer-Verlag: London, UK, **2083**, pp 701, 3-540-42254-4, 2001.
- [35] Heflin, J.; Hendler, J.; and Luke, S., Coping with Changing Ontologies in a Distributed Environment, *AAAI Conference Ontology Management Workshop*, AAAI Press, pp 74, 1999.
- [36] Heflin, J.; and Pan, Z., A Model Theoretic Semantics for Ontology Versioning, *Semantic Web - ISWC 2004: Third International Semantic Web Conference*, Hiroshima, Japan, Springer, pp 62, 3-540-23798-4, 2004.
- [37] Helal, A., InfoSleuth: Agent-based Semantic Integration of Information in Open and Dynamic Environments, In *Readings in Agents*, Huhns, M.; Singh, M. Eds.; Morgan Kaufman, 1997.
- [38] Hondo, M.; and Kaler, C., Web Services Policy Framework (WSPolicy) Version 1.0, Availabel at <http://www.verisign.com/wss/WS-Policy.pdf>, 2002.
- [39] Hsu, P.-Y.; and Zaniolo, C., A new User's Impressions on LDL++ and CORAL, *Workshop on Programming with Logic Databases (Informal Proceedings)*, ILPS, pp 134, 1993.

- [40] Huang, C.-L.; Chao, K.-M.; and Lo, C.-C., A Moderated Fuzzy Matchmaking for Web Services, *The 5th International Conference on Computer and Information Technology (CIT2005)*, Shanghai, China, pp 1116, 2005.
- [41] Hull, R., Managing semantic heterogeneity in databases: A theoretical perspective, *ACM Symposium on Principles of Database Systems*, pp 51–61, Tuscon, Arizona, USA, 1997.
- [42] Jilovec, N., A to Z of EDI, Second Edition, 29th Street Press; p 221, 1882419162, 2nd edition, April 1, 1998.
- [43] Kalfoglou, Y.; and Schorlemmer, M., Ontology mapping: the state of the art, *The Knowledge Engineering Review*, **18**(1), pp 1 0269-8889, Cambridge University Press New York, NY, USA, 2003.
- [44] Kalyanpur, A.; Parsia, B.; and Hendler, J., A Tool for Working with Web Ontologies, *International Journal on Semantic Web and Information Systems*, **I**(1), pp 36, 2005.
- [45] Karp, P. D.; Chaudhri, V. K.; and Thomere, J. F., XOL: An XML-Based Ontology Exchange Language, *Technical Note 559, AI Center, SRI International, Menlo Park, Canada*, 1999.
- [46] Karp, P. D.; Meyers, K.; and Gruber, T., The Generic Frame Protocol, *Joint Conference on Artificial Intelligence*, Montreal, Quebec, Canada, Morgan Kaufmann, pp 768, 1995.
- [47] Kawamura, T.; Blasio, J. D.; Hasegawa, T.; Paolucci, M.; and Sycara, K., Preliminary Report of Public Experiment of Semantic Service Matchmaker with UDDI Business Register, *1st International Conference on Service Oriented Computing (ICSOC 2003)*, Trento, Italy, pp 208, 2003.

- [48] Kawamura, T.; Blasio, J. D.; Hasegawa, T.; Paolucci, M.; and Sycara, K., Public Deployment of Semantic Service Matchmaker with UDDI Business Registry, *3rd International Semantic Web Conference (ISWC 2004)*, Hiroshima, Japan, Springer, **3298**, pp 752, 978-3-540-23798-3, 2004.
- [49] Kent, R., Conceptual knowledge modelling language, (2002). Available at: <http://www.ontologos.org/CKML/>.
- [50] Kifer, M.; Lausen, G.; and Wu, J., Logical foundations of object-oriented and frame-based languages, *Journal of the ACM (JACM)*, **42**(4), pp 741, 0004-5411, ACM Press, New York, NY, USA, 1995.
- [51] Klein, M.; Fensel, D.; Kiryakov, A.; and Ognyanov, D., Ontology versioning and change detection on the web, *13th International Conference on Knowledge Engineering and Knowledge Management*, Spain, Springer Verlag, **2473**, pp 197, 2002.
- [52] Klein, M.; Kiryakov, A.; Ognyanoff, A.; and Fensel, D., Finding and specifying relations between ontology versions, *ECAI-02 Workshop on Ontologies and Semantic Interoperability*, Lyon, France, 2002.
- [53] Lara, R.; Roman, D.; Polleres, A.; and Fensel, D., A Conceptual Comparison of WSMO and OWL-S, In *Web Services*, Springer Berlin / Heidelberg: pp 254, 2004.
- [54] Lausen, H.; Bruijn, J. d.; Polleres, A.; and Fensel, D., WSML - a Language Framework for Semantic Web Services, *Position Paper for the W3C rules workshop*, Washington DC, USA, April 2005.

- [55] Lenat, D. B.; and Guha, R. V., Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project, Addison-Wesley Pub, Boston, Massachusetts, USA; p 372, 0201517523, 1990.
- [56] Li, L.; and Horrocks, I., A software framework for matchmaking based on semantic web technology, *12th International World Wide Web Conference*, Budapest, Hungary, ACM Press, pp 331, 1-58113-680-3, 2003.
- [57] Li, Y.; Su, S.; and Yang, F., A Peer-to-Peer Approach to Semantic Web Services Discovery, *The 6th International Conference on Computational Science – ICCS 2006*, University of Reading, UK, Springer Berlin, pp 73, 3-540-34385-7, 2006.
- [58] Liam, E. G., Practical IDL Programming, Morgan Kaufmann; p 528, 1558607005, 1st edition, August 7, 2001.
- [59] Liang, Q.; Chakarapani, L. N.; Su, S. Y. W.; Chikkamagalur, R. N.; and Lamr, H., A Semi-automatic approach to composite web services discovery, description and invocation,, *International Journal of Web Services Research*, *1*(4), pp 64–89, Idea Group Publishing, 2004.
- [60] Lopes, J. G. R. C.; Mel, A. C. M. A.; Dantas, M. A. R.; and Ralha, C. G., A Proposal and Evaluation of a Mechanism for Grid Ontology Merge, *20th International Symposium on High-Performance Computing in an Advanced Collaborative Environment (HPCS'06)*, Newfoundland, Canada IEEE Computer Society, p 2, 1550-5243, 2006.
- [61] Ludwig, H., Web Service Level Agreement (WSLA) Language Specification Version 1.0, *Techincal report in IBM*, 2003.
- [62] M.C.Jaeger; Tang, S.; and Liebetrueth, C., The TUB OWL-S Matcher. Available at: Available at: <http://ivs.tu-berlin.de/Projekte/owlsmatcher/index.html>.

- [63] Maedche, A.; and Staab, S., Measuring Similarity between Ontologies, *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, Sigüenza, Spain, Springer, **2473**, pp 251, 3-540-44268-5, 2002.
- [64] Mazzieri, M.; and Dragoni, A. F., *A Fuzzy Semantics for Semantic Web Languages*, Proceedings of the ISWC Workshop on Uncertainty Reasoning for the Semantic Web, (2005), [Electric version] Vol. Vol-173. Available at: <http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-173/>.
- [65] Miles, M. B.; and Huberman, M., Qualitative data analysis: an expanded sourcebook, Thousand Oaks, California, Sage Publications; 1994.
- [66] Miller, G., WordNet: An Electronic Lexical Database, The MIT Press; 026206197X, May 15, 1998.
- [67] Miller, G. A., WordNet: a lexical database for English, *Communications of the ACM*, **38**(11), pp 39, 0001-0782, ACM Press, 1995.
- [68] Monge, A.; and Elkan, C., An efficient domain-independent algorithm for detecting approximately duplicate database records, *The proceedings of the SIGMOD 1997 workshop on data mining and knowledge discovery*, 1997.
- [69] Mowbray, T. J.; and Malveau, R. C., CORBA Design Patterns, John Wiley & Sons; p 333, 0471158828, 1997.
- [70] Mowbray, T. J.; and Zahavi, R., The Essential CORBA: Systems Integration Using Distributed Objects, John Wiley & Sons, New York; 1995.

- [71] Narayanan, S.; and McIlraith, S. A., Simulation, verification and automated composition of web services, *International World Wide Web Conference*, Honolulu, Hawaii, USA, ACM Press, New York, NY, USA, pp 77, 1-58113-449-5, 2002.
- [72] Natalya, F. N.; and Mark, A. M., Prompt: Algorithm and tool for automated ontology merging and alignment, *17th Natl. Conf. On Artificial Intelligence (AAAI2000)*, Austin, Texas, USA, 2000.
- [73] Noy, N. F.; and Musen, M. A., Using PROMPT Ontology-Comparison Tools in the EON Ontology Alignment Contest, *Workshop on Evaluation of Ontology-based Tools (EON2004) at the Third International Conference on the Semantic Web (ISWC-2004)*, Hisroshima, Japan, 2004.
- [74] OASIS Organization, UDDI - Universal Description, Discovery and Integration. Available at: <http://uddi.org/>.
- [75] Oundhankar, S.; Verma, K.; Sivashanugam, K.; Sheth, A.; and Miller, J., Discovery of web services in a Multi-Ontologies and Federated Registry Environment, *International Journal of Web Services Research*, **1**(3), 2005.
- [76] Paolucci, M.; Kawamura, T.; Payne, T. R.; and Sycara, K., Semantic Matching of Web services Capabilities, *1st International Semantic Web Conference (ISWC 2002)*, Sardinia, Italy, Springer, **2342**, pp 333, 3-540-43760-6, 2002.
- [77] Patil, A.; Oundhakar, S.; Sheth, A.; and Verma, K., METEOR-S Web service Annotation Framework, *Proceedings of the 13th international conference on World Wide Web*, New York, USA, ACM Press, pp 553, 1-58113-844-X, 2004.
- [78] Phyllis, K. S., EDI: The competitive edge, McGraw-Hill; p 346, 0070595119, 1989.

- [79] Pinto, H. S.; and Martins, J. P., A methodology for ontology integration, *Proceedings of the 1st international conference on Knowledge capture* Victoria, British Columbia, Canada, ACM Press, pp 131, 1-58113-380-4, 2001.
- [80] Porter, E. H.; and Winkler, W. E., Approximate String Comparison and its Effect on an Advanced Record Linkage System, *Record Linkage Techniques - 1997*, pp 190, National Research Council, 1999.
- [81] Porter, M. F., An algorithm for Suffix Stripping, In *Morgan Kaufmann Multimedia Information And Systems Series*, Morgan Kaufmann Publishers: pp 313 1997.
- [82] Preece, A. D.; Hui, K. K.; Gray, W. A.; Marti, P.; BenchCapon, T. J. M., *et al.*, KRAFT: An agent architecture for knowledge fusion, *International Journal of Cooperative Information Systems*, **10**(1-2), pp 171–195, 2001.
- [83] Recuerda, F. M.; and Robertson, D., Discovery and Uncertainty in Semantic Web Services, *Proceedings of the International Semantic Web Conference (ISWC 2005), Workshop 3: Uncertainty Reasoning for the Semantic Web*, Galway, Ireland, Availabe at: <http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-173/>, 2005.
- [84] Redmond, F., COM/DCOM: Developing Distributed Enterprise Applications With the MS Common Object Model, HUNGRY MINDS publisher; 0764580442, 1997.
- [85] Rijsbergen, K. V., Information Retrieval, Butterworths; Second Edition, 1979.

- [86] Ristad, E. S.; and Yianilos, P. N., Learning string edit distance, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20**(5), pp 522–532, 1998.
- [87] Rodriguez, M. A.; and Egenhofer, M. J., Determining Semantic similarity among entity classes from different ontologies, *IEEE Transactions on Knowledge and Data Engineering*, **15**(2), pp 442, 1041-4347, 2003.
- [88] Roman, D.; Lausen, H.; and Keller, U., Web Service Modeling Ontology - Standard (WSMO - Standard), WSMO deliverable D2 version 1.1, (2004). Available at: <http://www.wsmo.org/TR/d5/d5.2/v0.1/>.
- [89] Roman, D.; Lausen, H.; and Keller, U., Web Service Modeling Ontology - Standard (WSMO - Standard), WSMO deliverable D2 version 1.1. Available at: <http://www.wsmo.org/TR/d5/d5.2/v0.1/>.
- [90] Roy, M.; and Edward, A., Inside DCOM: Microsoft's distributed object architecture extends the capabilities of COM to work across the network, *DBMS*, **10**(4), pp 26, 1041-5173, 1997.
- [91] Sahai, A.; Durante, A.; and Machiraju, V., Towards Automated SLA Management for Web Services, *Technical report HPL-2001-310 (R.1) on HP Laboratories*, 2002.
- [92] Said, M. P.; Matono, A.; and Kojima, I., SPARQL-based OWL-S Service Matchmaking, *SMR 2006 - 1st International Workshop on Semantic Matchmaking and Resource Retrieval: Issues and Perspectives*, Seoul, Korea, on-line proceedings of CEUR-WS, **178**, 2006.

- [93] Salton, G., Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer, Addison-Wesley Ed. Massachusetts; 1988.
- [94] Sapkota, B.; Kilgarriif, E.; Moyano, F. J. M.-R.; Toma, I.; and Krummenacher, R., D21.v0.1 WSMX Triple-Space Computing, WSMO Working Draft, (2005). Available at: <http://www.wsmo.org/TR/d21/v0.1/20050411/>.
- [95] Shaw, S.; Foggiano-Bish, L.; Garcia, I.; Tillman, G.; Tryon, D., *et al.*, Improving Data Quality Via LDL++, *Workshop on Programming with Logic Databases (Informal Proceedings)*, ILPS, pp 60, 1993.
- [96] Simpson, T.; and Dao, T., WordNet-based semantic similarity measurement. Available at: <http://www.codeproject.com/csharp/semanticssimilaritywordnet.asp>.
- [97] Sivashanmugam, K.; Verma, K.; Sheth, A.; and Miller, J., Adding Semantics to Web Services Standards, *The 2003 International Conference on Web Services (ICWS'03)*, Erfurt, Germany, 2003.
- [98] Srinivasan, N.; Paolucci, M.; and Sycara, K., Adding OWL-S to UDDI, implementation and throughput, *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, California, USA, 2004.
- [99] Stanford University, Protégé Project. Available at: <http://protege.stanford.edu/>.
- [100] Stoilos, G.; Stamou, G.; Tzouvaras, V.; Pan, J.; and Horrocks, I., The Fuzzy Description Logic f-SHIN, *Proceedings of the ISWC Workshop on Uncertainty Reasoning for the Semantic Web*, Galway, Ireland, Available at: <http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-173/>, **173**, 2005.

- [101] Sycara, K.; J. Lu, M. K.; and Widoff, S., Dynamic service matchmaking among agents in open information environments, *ACM SIGMOD Record (Special Issue on Semantic Interoperability in Global Information Systems)*, **28**(No.1), pp 47, 1999.
- [102] Sycara, K.; J. Lu, M. K.; and Widoff, S., Matchmaking among heterogeneous agents on the Internet, *AAAI Spring Symposium on Intelligent Agents in Cyberspace*, 1999.
- [103] Sycara, K.; Widoff, S.; and M. Klusch, J. L., LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace, *Autonomous Agents and Multi- Agent Systems*, Vol.5, pp 173, 2002.
- [104] Tejada, S.; Knoblock, C. A.; and Minton, S., Learning object identification rules for information integration, *Information Systems*, **26**(8), pp 607–633, 2001.
- [105] Trastour, D.; Bartolini, C.; and Gonzalez-Castillo, J., A Semantic Web Approach to Service Description for Matchmaking of Service, *1st Semantic Web Working Symposium*, CA, 2001.
- [106] Tversky, A.; and Kahneman, D., Judgement under uncertainty: Heuristics and biases, *Science*, **185**(4157), pp 1124, 185.4157.1124, 1974.
- [107] U, T.; W, S.; and W, N., A Semantic Web based Peer-to-Peer Service Registry Network, Technical Report, Learning Lab Lower Saxony, 2003.
- [108] UCLA University, Logical deduction language, (1998). Available at: <http://pike.cs.ucla.edu/ldl/tutorial/ldl++tutorial.pdf>.
- [109] W3C Organization, HTML - Hypertext Markup Language, (1995). Available at: <http://www.w3.org/MarkUp/>.

- [110] W3C Organization, OWL - Web Ontology Language Overview. Available at: <http://www.w3.org/TR/owl-features/>.
- [111] W3C Organization, RDF Vocabulary Description Language 1.0: RDF Schema, (2004). Available at: <http://www.w3.org/TR/rdf-schema/>.
- [112] W3C Organization, The World Wide Web Consortium. Available at: <http://www.w3.org/>.
- [113] Walsh, A. E., UDDI, SOAP, and WSDL: The Web Services Specification, Pearson Education; p 336, 0130857262, 1st edition, 2002.
- [114] Wang, J. Z.; and Ali, F., An Efficient Ontology Comparison Tool for Semantic Web Applications, *2005 IEEE / WIC / ACM International Conference on Web Intelligence (WI 2005)*, Compiegne, France, IEEE Computer Society, pp 372, 0-7695-2415-X, 2005.
- [115] Wang, J. Z.; Ali, F.; and Appaneravanda:, R., A Web Service for Efficient Ontology Comparison, *2005 IEEE International Conference on Web Services (ICWS 2005)*, Orlando, FL, USA, IEEE Computer Society, pp 843, 0-7695-2409-5, 2005.
- [116] Wu, H.; Jin, H.; Li, Y.; and Chen, H., An Approach for Service Discovery Based on Semantic Peer-to-Peer, *Advances in Computer Science – ASIAN 2005: 10th Asian Computing Science Conference*, Kunming, China, Springer Berlin, **3818**, pp 259, 3-540-30767-2, 2005.
- [117] Wu, Z.; and Palmer, M., Verbs semantics and lexical selection, *32nd annual meeting on Association for Computational Linguistics*, Las Cruces, New Mexico, Association for Computational Linguistics, pp 133, 1994.

- [118] Yu, S.; Liu, J.; and Le, J., Decentralized Web Service Organization Combining Semantic Web and Peer to Peer Computing, *European Conference on Web Services ECOWS 2004*, Erfurt, Germany, Springer Berlin, 3250, pp 116, 3-540-23202-8, 2004.
- [119] Yu, S.; Liu, J.; and Le, J., Intelligent Web Service Discovery in Large Distributed System, *Intelligent Data Engineering and Automated Learning – IDEAL 2004*, Exeter, UK, Springer Berlin, **3177**, pp 166, 3-540-22881-0, 2004.
- [120] Yu, X.; Zhang, L.; Li, Y.; and Chen, Y., WSCE: a flexible Web Service Composition Environment, *IEEE International Conference on Web Services (ICWS'04)*, San Diego, California, USA, IEEE Computer Society, pp 428, 0-7695-2167-3, 2004.
- [121] Zamora, E. M.; Pollock, J. J.; and Zamora, A., The Use of Trigram Analysis for Spelling Error Detection, *Information Processing and Management*, **17**(6), pp 305, 1981.
- [122] Zhou, C.; Chia, L. T.; and Lee, B. S., Semantics in Service Discovery and QoS Measurement, *IEEE IT Professional Magazine*, **7**(2), pp 29, 1520-9202, IEEE Educational Activities Department, 2005.
- [123] Zhou, C.; Chia, L. T.; and Lee, B. S., Web Services Discovery with DAML-QoS Ontology, *International Journal of Web Services Research (JWSR)*, **2**(2), pp 44, 1545-7362, Idea Group Publishing, 2005.

Publications Derived from this Research

1. Le, D. N. and Goh, A. E. S., Matching Semantic Web Services Using Different Ontologies, *The Fifth International Conference on Web Engineering (ICWE2005)*, LNCS 3579, pp 302, Sydney, Australia, 2005.
2. Le, D. N.; Goh, A. E. S.; and Cao, H.T., Multi-Ontology Matchmaker, *The 7th International Conference on Information Integration and Web-Based Applications & Services (iiWAS'2005)*, Kuala Lumpur, Malaysia, 2005.
3. Le, D. N. and Goh, A. E. S., Semantic Similarity between Concepts from Different OWL Ontologies, *The 4th International IEEE Conference on Industrial Informatics (INDIN'06)*, Singapore, 2006.
4. Le, D. N.; Tran, M.H.; and Goh, A. E. S., MOD - A Multi-Ontology Discovery System, *The International Workshop on Semantic Matchmaking and Resource Retrieval (co-located with VLDB'06)*, Seoul, Korean, 2006.
5. Le, D. N.; Goh, A. E. S.; and Cao, H.T., Multi-Ontology Matchmaker, *International Journal of Metadata, Semantics and Ontologies (IJMSO)*, Inderscience, **1**(4), pp 250, 2007.

6. Le, D. N. and Goh, A. E. S., Identifying Related Ontologies, *The First International Conference on Ubiquitous Information Management and Communication*, Suwon, Korea, 2007.
7. Le, D. N.; Goh, A. E. S.; and Cao, H.T., A Survey and a Taxonomy of Web Service Discovery Systems, *Int. J. of International Journal of Information Technology and Web Engineering (IJITWE)*, IGI Global, 2(2),pp 65, 2007.
8. Le, D. N. and Goh, A. E. S., A Survey of Computing Ontological Concept Similarity, *The 3rd International Conference on Semantics, Knowledge and Grid (SKG)*, Xi'An, China, 2007.
9. Le, D. N. and Goh, A. E. S., Applying Fuzzy Logic To Multi-Ontology Web Service Discovery System, *The 2007 IEEE Asia-Pacific Services Computing Conference (IEEE APSCC 2007)*, Tsukuba Science City, Japan, 2007.
10. Le, D. N.; Goh, A. E. S.; and Le, Q.H., Comparing Two Ontologies, *to appear in the Int. J. of Web Engineering and Technology (IJWET)*, Inderscience.

Appendix A

Fig. A.1: e-Business and Price ontologies

Fig. A.2: Animal, Biology, and New Equipment ontologies

Appendix B

Table B.1: Completed results of comparing 100 ontology pairs. Each ontology is represented by its name and root. Two ontologies have “sub ontology” relationship has one common part.

Index	Ontology 1	Ontology 2	Relationship	Percentage Similarity
1	Equipment (Equipment)	EquipSame (Equipment)	The Same	100%
2	Equipment (Equipment)	Equipsub01 (Equipment)	Sub Ontology	96.15%
3	Equipment (Equipment)	Equipsub02 (Electronic Equip)	Sub Ontology	94.23%
4	Equipment (Equipment)	Equipsub03 (Electronic Equip)	Sub Ontology	25%
5	Equipment (Equipment)	Equipsub04 (Electronic Equip)	Sub Ontology	71.15%
6	Equipment (Equipment)	Equipsub05 (Electronic Equip)	Sub Ontology	36.53%
7	Equipment (Equipment)	Equipsub06 (Computer)	Sub Ontology	19.23%
8	Equipment (Equipment)	Equipsub07 (Hardware)	Sub Ontology	38.46%
9	Equipment (Equipment)	Equipsub08 (Software)	Sub Ontology	30.77%
10	Equipment (Equipment)	Equipsub09 (Software)	Sub Ontology	26.92%
11	Equipsub01 (Equipment)	Equipsub02 (Electronic Equip)	Sub Ontology	98%
12	Equipsub01 (Equipment)	Equipsub03 (Electronic Equip)	Sub Ontology	26%
13	Equipsub01 (Equipment)	Equipsub04 (Electronic Equip)	Sub Ontology	74%
14	Equipsub01 (Equipment)	Equipsub05 (Electronic Equip)	Sub Ontology	38%
15	Equipsub01 (Equipment)	Equipsub06 (Computer)	Sub Ontology	20%

16	Equipsub09 (Software)	Equipsub03 (Electronic Equip)	Different Onts	0%
17	Equipsub09 (Software)	Equipsub04 (Electronic Equip)	1 Common Ont	15.0%
18	Equipsub09 (Software)	Equipsub05 (Electronic Equip)	Sub Ontology	73.68%
19	Equipsub09 (Software)	Equipsub06 (Computer)	Different Onts	0%
20	Equipsub09 (Software)	Equipsub07 (Hardware)	Different Onts	0%
21	Equipsub08 (Software)	Equipsub03 (Electronic Equip)	Different Onts	0%
22	Equipsub08 (Software)	Equipsub04 (Electronic Equip)	1 Common Ont	17.78
23	Equipsub08 (Software)	Equipsub05 (Electronic Equip)	Sub Ontology	84.21
24	Equipsub08 (Software)	Equipsub06 (Computer)	Different Onts	0%
25	Equipsub08 (Software)	Equipsub07 (Hardware)	Different Onts	0%
26	Equipsub07 (Hardware)	Equipsub03 (Electronic Equip)	Different Onts	0%
27	Equipsub07 (Hardware)	Equipsub04 (Electronic Equip)	Sub Ontology	54.05
28	Equipsub07 (Hardware)	Equipsub05 (Electronic Equip)	Different Onts	0%
29	Equipsub07 (Hardware)	Equipsub06 (Computer)	Different Onts	0%
30	Equipsub07 (Hardware)	Equipsub07 (Hardware)	The Same	100%
31	TechSub01 (Hardware)	TechSub07 (Software)	Different Onts	0%
32	TechSub01 (Hardware)	TechSub08 (Output)	Different Onts	0%
33	TechSub01 (Hardware)	TechSub09 (CPU)	Different Onts	0%
34	TechSub01 (Hardware)	TechSub10 (Software)	Different Onts	0%
35	TechSub01 (Hardware)	TechSub11 (Output)	Different Onts	0%
36	TechSub01 (Hardware)	TechSub12 (CPU)	Different Onts	0%
37	TechSub01 (Hardware)	TechSub13 (Software)	Different Onts	0%
38	TechSub01 (Hardware)	TechSub14 (Output)	Different Onts	0%

39	Technology (Technology)	TechSub03 (CPU)	Sub Ontology	10.3%
40	Technology (Technology)	TechSub04 (Software)	Sub Ontology	31.0%
41	TechSub04 (Software)	TechSame (Technology)	Sub Ontology	31%
42	TechSub04 (Software)	TechSub01 (Hardware)	Different Onts	0%
43	TechSub04 (Software)	TechSub02 (Output)	Different Onts	0%
44	TechSub04 (Software)	TechSub03 (CPU)	Different Onts	0%
45	TechSub04 (Software)	TechSub04 (Software)	The Same	100%
46	TechSub04 (Software)	TechSub05 (Office)	Sub Ontology	33%
47	TechSub04 (Software)	TechSub06 (Software)	Sub Ontology	72.2%
48	TechSub04 (Software)	TechSub07 (Software)	Sub Ontology	61.1%
49	TechSub04 (Software)	TechSub08 (Software)	Sub Ontology	38.8%
50	TechSub04 (Software)	TechSub09 (Technology)	Different Onts	0%
51	TechSub04 (Software)	TechSub10 (Technology)	Different Onts	0%
52	TechSub04 (Software)	TechSub11 (Computer)	Different Onts	0%
53	TechSub04 (Software)	TechSub12 (Computer)	Different Onts	0%
54	TechSub04 (Software)	TechSub13 (Computer and Personal Computer)	Different Onts	0%
55	TechSub04 (Software)	TechSub14 (Personal Computer)	Different Onts	0%
56	TechSub09 (Technology)	TechSub07 (Software)	Different Onts	0%
57	TechSub09 (Technology)	TechSub08 (Software)	Different Onts	0%
58	TechSub09 (Technology)	TechSub09 (Technology)	The same	100%
59	TechSub09 (Technology)	TechSub10 (Technology)	Sub Ontology	57%

60	TechSub09 (Technology)	TechSub11 (Computer)	Sub Ontology	52.6%
61	Equipment (Equipment)	Technology (Technology)	3 Commons	68.75%
62	Equipment (Equipment)	TechSub01 (Hardware)	1 Common	32.73%
63	Equipment (Equipment)	TechSub02 (Output)	Sub Ontology	11.54%
64	Equipsub01 (Equipment)	Technology (Technology)	3 Commons	70.96%
65	Equipsub02 (Electronic Equip)	Technology (Technology)	3 Commons	72.13%
66	Equipsub03 (Electronic Equip)	Technology (Technology)	1 Common	16.95%
67	Equipsub04 (Electronic Equip)	Technology (Technology)	1 Common	55%
68	Equipsub01 (Equipment)	TechSub04 (Software)	1 Common	30.77%
69	Equipsub02 (Electronic Equip)	TechSub04 (Software)	1 Common	31.37%
70	Equipsub03 (Electronic Equip)	TechSub04 (Software)	Different Onts	0%
71	Agent	Bookroom	Sub Ontology	99.78%
72	Agent	Broker-admin	Different Onts	0%
73	Agent	Broker-comm	Sub Ontology	86.17
74	Agent	calendarclock	Sub Ontology	2.65%
75	Agent	contextbroker	Sub Ontology	96.43
76	Agent	meeting	The Same	100%
77	Agent	foaf-basic	1 Common	8.64
78	role	rcc-basic	Different Onts	0%
79	role	location	Sub Ontology	44.44%
80	role	Agent	The Same	100%
81	foaf-basic	rcc-basic	Different Onts	0%
82	Place	Location	1 Common	37.5%
83	Action	role	The Same	100%
84	action	academia	Sub Ontology	88.04%
85	context	contextbroker	1 Common	90%

86	broker-comm	broker-admin	Different Onts	0%
87	context	calendarclock	Sub Ontology	2.30%
88	space-basic	place	Sub Ontology	84.21%
89	powerpoint	document	Sub Ontology	96.30%
90	image-doc	document	Sub Ontology	83.87
91	eBusiness	BuyTechTemplate	Different Onts	0%
92	BuyTechSub05	BuyTechTemplate	Sub Ontology	25.80%
93	BuyTechSub04	BuyTechTemplate	Sub Ontology	56.46%
94	BuyTechSub03	BuyTechTemplate	Sub Ontology	17.74
95	BuyTechSub04	BuyTechSub03	Sub Ontology	31.42
96	eBusiness	BuyTechSub04	Different Onts	0%
97	eBusiness	PriceTemplate	Different Onts	0%
98	PriceTemplate	PriceSame	The Same	100%
99	PriceSame	PriceSub01	Sub Ontology	54.54%
100	PriceSub01	BuyTechTemplate	Different Onts	0%