Ontology Based Web Services Personalisation

Ojitha W. H. Kumanayaka
Registration Number: MSC/AC/2004/013

Supervisor: Dr. D. N. Ranasinghe

December, 2007

This dissertation is submitted in partial fulfilment of the requirements
of the Degree of Msc in Advance Computing
of
The University of Colombo.

Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any materials previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, to be made available for photocopying and for inter library loans, and for the title and summary to be made available to outside organisations.

Signature of Candidate   Date

Abstract

With the maturing of the semantic web, Web services community has been looking to find the solutions for the problems which were unresolved for a long time. Recent advent of the semantic web and semantic web services languages are still in their childhood, although they are rich languages compared to other web based languages. Converting conventional web services to semantic web is more than an enhancement to the conventional service, which is a paradigm shift. Conventional web services are based on the flow control and data, but semantic web services are based on the knowledge. Therefore, in this thesis, we have address the theoretical problems of creating semantic web services from the conventional web services.

One of the very important problems we have addressed is autonomous web services selection and composition. Due to the lack of semantic support in the conventional web services, this has been a dream for many years for them. Although semantic web services infrastructure provide an elegant solution to enabling semantic to conventional services, still service matching for selection remains problem for research.

In this thesis, we have presented a complete end-to-end framework to select web service which is the best fit for the service consumer. In the occasion where there are number of best services available, finding the best fit among them is the challenge we have addressed. Basically, the framework responsible to accomplish two task, which are service matching and selection of the best fit on consumer preferences.

Because semantic web services approach is knowledge based, incorrect ontologies can be caused to malformed results which are not common in the conventional web services. Semantic defects are logical contradictions that manifest as either inconsistent ontologies or as concepts that can not be satisfied. Incorrect ontologies are possible due to the lack of ontology debugging, tracing and testing difficulties. Therefore, a new ontology validation formalism has been introduced to check the correctness of the ontology before use. This validation formalism will open the door to check the accuracy of any ontology which was equally applicable to personalisation ontology created as a part of the framework.

Contents

Contents **i**

List of Figures **v**

List of Tables **vi**

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Research Objectives . . . 2
- 1.3 Preliminaries . . . 3

2 Background **4**
- 2.1 Current status of the Web . . . 4
- 2.2 Description Logic . . . 6
- 2.3 Ontology . . . 6
- 2.4 OWL and OWL-S . . . 6
- 2.5 Semantic Web Services . . . 7

3 DL for OWL **10**
- 3.1 Ontology Engineering . . . 10
- 3.2 Knowledge Discovery . . . 11
- 3.3 ontology construction . . . 11
- 3.4 Analysis of Family Ontology . . . 12
  - 3.4.1 Infer the Family Ontology . . . 16
  - 3.4.2 $\mathcal{SHIQ}$ features . . . 17

4 Ontology Validation **20**
- 4.1 Logical formalism . . . 21
- 4.2 Sequent calculus based validation . . . 21
- 4.3 General validation . . . 23
  - 4.3.1 Class Inferences . . . 23

4.3.2 Instance Inference . . . 28

5 Ontology for Personalisation 32
5.1 Introduction . . . 32
5.2 Ontology for Personalisation . . . 32
5.2.1 Whole-part issues . . . 33
5.2.2 Temporal Model . . . 33
5.2.3 Locative Model . . . 38
5.2.4 Service Availability . . . 39
5.3 Matchmaking Example . . . 40
5.3.1 Preference Engineering . . . 43

6 Matchmaker 46
6.1 Architectural Significances . . . 46
6.2 Survey on selection and composition . . . 49
6.3 Solution Architecture . . . 50
6.3.1 Grounding related issues . . . 53
6.3.2 Capability Matching . . . 53
6.4 Approach to Dynamic Composition . . . 53
6.4.1 Advertise Services . . . 53
6.4.2 Semantic Service Discovery . . . 54
6.4.3 Operate Services . . . 54
6.4.4 UDDI Integration . . . 55
6.4.5 Matchmaker . . . 55
6.5 Matchmaking Algorithm . . . 61
6.5.1 Exact Level . . . 61
6.5.2 Other Levels . . . 61
6.5.3 Matching . . . 65

7 Conclusion 67
7.1 This work . . . 67
7.2 Future Work . . . 68
7.2.1 Personalisation Ontology . . . 68
7.2.2 Matchmaker . . . 68

A Abbreviations 73

B Preliminaries 75
B.1 Semantic Web . . . 75
B.2 Description Logic . . . 75
B.2.1 Description Languages . . . 77
B.2.2 Terminological axioms . . . 78
B.2.3 Semantics . . . 79

B.2.4 Open World Assertions 80
B.2.5 Additional constructors 80
B.3 Ontology for semantic web 81
B.3.1 Introduction to Knowledge Discovery 81
B.3.2 DL as ontology language 82

C Ontologies 83
C.1 Family Ontology 83
C.2 Common Ontology 87
C.3 Temporal Ontology 88
C.4 Calendar Ontology 92
C.5 Locative Ontology 95

D Sequent Calculus for DL 97

Bibliography 99

Index 106

Acknowledgement

First, I would like to thank my advisor, Dr. D. N. Ranasinghe, for his continuous support, encouragement and guidance over the years and for providing me the freedom to pursue my interests and goals.
I would also like to thank Mr. Arjuna Samaraweera and Mrs. Marietta Lovendhal. Their continuing support and ongoing review and non-geek analysis of the manuscript has been invaluable. I would like to thank Mr. Ruwan Wijesinghe, who has done technical review. We are grateful to the online discussion forums, papers, research, and suggestions of the OWL-S Coalition.

On the personal front, I am eternally grateful to my family, my wife Waruni and daughter Dhisini Osani, for their endless love and support in every step of life, and for providing me with all the opportunities in the world to excel.

List of Figures

2.1 Semantic pyramid for Semantic Web 7
2.2 Semantic Web services infrastructure 9

3.1 CRISP-DM process Model 13
3.2 Protégé and Pellet Integrated Environment 14
3.3 Abstract Syntax for axiom 3.1 14
3.4 OWL for axiom 3.1 15
3.5 Abstract Syntax for axiom 3.2 15
3.6 OWL for axiom 3.2 15
3.7 Abstract Semantic for axiom 3.3 16
3.8 OWL code for axiom 3.3 16
3.9 Abstract Semantic for axiom 3.4 16
3.10 OWL code for axiom 3.4 17
3.11 Instance checking will result `GrandMother` entailment 18

4.1 SG for the Daily Mirror is a Tabloid 30
4.2 SG for the Pet is Person, Spike is an Animal 30
4.3 SG for the Walt is animal lover 31

5.1 Definition for the TemporalEntity 34

6.1 Service discovery general architecture 48
6.2 OWL technology stack for Matchmaker Architecture 52
6.3 The Exact graph 62
6.4 The Subsume graph 63
6.5 The Matching graph 65

7.1 Quality Rating of Book Selling profile 70
7.2 Personalisation Ontology 70

List of Tables

5.1 Specifications For Advertisement And Query For Exact 42
5.2 Specifications For Advertisement And Query For Plug-in 42
5.3 Specifications For Advertisement And Query For Subsume 42
5.4 Specifications For Advertisement And Query For Intersection 43

B.1 The basic description languages. 77
B.2 Syntax and semantics of the $\mathcal{SI}$ family of DLs. 78

D.1 First order classical sequent calculus 98

Chapter 1

Introduction

1.1 Motivation

In 1940's at the time of the industrial era to be precise in 1947; the mathematician called Alan Turing for first time explored the concept of machine intelligence. According to him, computing machine can be called intelligent if it could deceive a human into believing that it is human. Still, in the current day information age, we are researching to achieve this goal. The based of the machine intelligence is machine learning. In computing, computational intelligences are the important topics. soft-computing and artificial intelligences are the two distinct branches of computational intelligences. For example, Artificial Intelligence involves symbolic computation (i.e.,the mathematical transformation of symbolic expressions, using computer algorithms), while Soft-Computing (i.e., techniques in computer science and in artificial intelligence, such as Fuzzy logic, Neural networks, and Probabilistic reasoning, that resemble human reasoning more closely than traditional techniques) involves intensive numerical computation. The following subbranches of machine intelligence have particular relevance to the Semantic Web and the idea Web intelligence: computational complexity, descriptive logic,ontology, inference, and software agents.

Semantic Web[67] offers users the ability to work on shared knowledge by constructing new meaningful representations on the web [4]. The current state-of-the art, of web service deployment and applications do not fully exploit the potential of web services and even less so of semantic web services which is a counterpart of the next generation web services based on semantic web.

The current emphasis of web services design has been on the execution of individual services rather than on the more important problem of how to select the best suitable web service for the context and how selected web services could collaborate dynamically within the context to provide higher a degree of functionality. OWL-S [24] services extend this paradigm by providing a more flexible approach to dynamically locate and assemble distributed web services [57] in a global environment. However, the problem of how to select trustworthy web service remains.

The main characteristic of semantic web is, its expressive power. Success of the Semantic Web depends on it expressiveness against computational complexity. Other hand, most notable success so far is the adaptation of the DL-based language OWL[49] as ontology language [7]. Therefore, usually Description Logic(DL) has been used to describe expressiveness of the Semantic Web. DL includes family of knowledge representation formalisms. Therefore, DL has been used as a primary language throughout this thesis.

Ontology is specification of a conceptualisation [31]. It is the theory of concepts and their relationships as well as rules within that domain. The Semantic Web requires the construction of ontologies for its various representations because ontology defines a common vocabulary which is shared by computers and humans. In this work, we need to compare two or more concepts, for that ontology is ideal because it combine taxonomies(set of classes and their relationships) with set of inference rules. Therefore, in this work we determine ontology as taxonomies plus inferences.

1.2 Research Objectives

As stated above, We have addressed the problem of how to select the trustworthy web services among semantically suitable web services . In this work, we present a novel approach which is driven on the above motivations. To sum up, our direct solution for this problem is personalisation. But we have deviated from the conventional personalisation which is based on statistical calculations mainly. Instead in this work we propose semantic based personalisation.

So far, most of the matching has been done using synthetic similarities to determine compatibilities. In our approach, we enhance the matching with semantically created assertions based on taxonomies encapsulated in the personalisation ontology which is explained in the chapter 5. Generally, to determine if a web service is suitable for particular use, both the functional and nonfunctional capabilities of the web service should be matched with the request. The Matchmaker serves as a liaison between a requester and a service provider.

Functional requirements are different form web service to web service. First step of the matchmaking is selecting the service which is functionally a match. The second step of the matchmaking is made on nonfunctional properties. In our approach, we address the second step of the matching which is on nonfunctional requirements. Matchmaking on nonfunctional requirements is only critical when there are one or more selected services which are semantically suitable on functional requirements. In this circumstance, only two choices are available, either human intervention to select best web service or select the best suitable service by evaluating against the predefine policies.

Our approach is based on the latter solution which will avoid the human intervention and automate the entire process. Both the requester and the service provider should share the same ontology; this is the main constraint focused in this research. Requester can define his or her policies based on the personalisation ontology. As well as service provider can advertise nonfunctional capabilities by refereeing to the same ontology.

The primary objectives of this research can be listed as follows

- Casual ontology development with DL,
- Define Personalisation ontology with nonfunctional requirements,
- Formal ontology validation independent of DL reasoners and
- Matchmaking of services based on personalisation ontology.

1.3 Preliminaries

All the important preliminaries are listed and described briefly in the appendix B. The goal of the this section is to sketch technologies which are indispensable for this research project. DL is the main ingredient which used to defined ontologies because DL has been used to define abstract specifications. Reader should have good knowledge of DL to read this thesis. However, Web Ontology Language (OWL)[49] is the implementation language[11] of the current day Semantic Web according to the W3C. Therefore, we have explained DL to OWL conversion in the 3^{rd} Chapter. To simplify the process we have used number of tools, one of the primary tools are Protégé and SWOOP editors [38, 37].

Chapter 2

Background

Over the past several decades, the Web has changed from proprietary sophisticated and distributed applications to a Web portals, such as Yahoo, Google and MSN, which records highest number of access hits comparing the entire internet. This tremendous improvement happened due to the W3C developed open Web standards. One of the notable standards is the eXtensible Markup Language (XML) which was developed as a markup language based on the principles and rules of Standard Generalized Markup Language (SGML) and uses tags that are not predefined. This gives XML great flexibility, and extensibility. The XML remains the interoperable bridge for exchanging data between reference architectures such a as J2EE and .NET, and as a result XML is an essential support for both Web Services' frameworks.

2.1 Current status of the Web

Nevertheless, the problem with performing intelligent tasks, such as automated Web Services, is that they first require human assistance, and second that they must rely on the inter-operation and inefficient exchange of the two competing proprietary server frameworks to successfully communicate complex business logic. However, current Web is based on HTML, which describes how information is to be displayed and laid out on a Web page for humans to read because the World Wide Web has its origin in the idea of hypertext, the Web is focused on textual data enriched by illustrative insertions of audiovisual materials. As a result, the Web has developed as a medium to display of information directly to humans. Hence, information is largely restricted to manual keywords searches such as Google is doing. There has been no real emphasis on establishing the capability for machine to understanding and learn ability and processing of web-based information. Although web is not developed in human sense, it is far developed with technological capabilities. For example, interoperability has been achieved by web services, so that web can operate in heterogeneous environments. The life cycle of WS includes the following main steps.

- *Advertisement:* The process of publishing the description and endpoints of WS in a service registry such as UDDI.
- *Discovery:* The process of locating all WSs that matches the requester's functional requirements.
- *Selection:* The process of selecting the most suitable WS out of the discovered ones, usually based on quality of service (QoS)
- *Composition:* The process of integrating selected WSs into complex process.
- *Invocation:* The process of invoking a single WS or complex process, by providing necessary inputs.

Out of five steps, WS *discovery* is considered to be the most important. It has attracted the major interest of the semantic web research community because this is a prerequisite for the next step, service *selection* which is the main focus of this thesis. WS descriptions are usually expressed as simple key-value pairs. Hence, current WS matching is based on the syntax description. Although underlining protocols of WSs are using SOAP or REST like protocols to communicate but not for the service matching. This kind of service discovery is not intelligent enough to find the best suitable service.

Due to the lack of machine understanding capabilities, current web services have suffered and devalued the technical capabilities. An emerging Web architecture called the Semantic Web (SW) offers users the ability to work on shared knowledge by constructing new meaningful representations on the Web that enable advance reasoning. Semantic Web research has developed from the traditions of Artificial Intelligence (AI) and ontology languages and offers automated processing through machine-understandable metadata. Semantic Web agents could utilise metadata, ontologies, and logic to carry out its tasks. As expect SW has effect on WSs and such a WS is called Semantic Web Service(SWS). In this case, no more used of key-value pair based syntax matching but rather more sophisticated ontology based mechanism is used. These set of rich technologies are capable of describe service capabilities, execution flows, personalisation and so on. Moreover, these technologies have given a new era of service *discovery* and *composition*.

2.2 Description Logic

Semantic Web architecture is designed to add some intelligence to the Web through machine processing capabilities. For the Semantic Web to succeed, the expressive power of the logic added to its markup languages must be balanced against the resulting computational complexity. The First-Order Logic (FOL) and its subset Description Logic offers attractive characteristics to represent semantics in semantic web applications. Description Logics (DLs)[27] allow specifying a terminological hierarchy using a restricted set of FOL formulas. The DLs have nice computational properties (they are often decidable and tractable), but the inference services are restricted to classification and subsumption. Given an instance description, the classifier will determine the most specific classes to which the instance belongs. DL has been mostly used in ontology development for semantic web. The reason is, DLs are the possible language for inference engines which are used in semantic web architecture.

2.3 Ontology

The semantic web's most important element is a set of ontologies that provide conceptual models for interpreting the information provided. Practical ontology is defined for particular domain of interest. In other words, Ontology is an explicit and formal specification of the domain of interest. In semantic web, ontologies are defined in Ontology web languages. Next section, we introduce these markup languages.

In the new field of semantic discovery, researchers have proposed service discovery methods for SWS. In this thesis we are consider new techniques which can implement the matchmaker in the Chapter 6.

2.4 OWL and OWL-S

Consider using logic such as DL within markup languages on the Semantic Web. Let us see how semantic we solve the main problem that machines are unable to automatically process the meaning of Web content without human intervention. With respect to the current web capabilities, the main lacking factor is that machines to perform useful automatic reasoning (inference) tasks, the language machines used must go beyond the basic semantics of XML Schema. Therefore semantic web is actually new addition of an ontology language, logic connectives, and rule systems to current web.

The Semantic Web will be constructed over the Resource Description Framework (RDF)[50] and Web Ontology Language (OWL). In addition, it will implement logic inference and rule systems. These languages are being developed by the W3C. Data can be defined and linked using RDF and OWL so that there

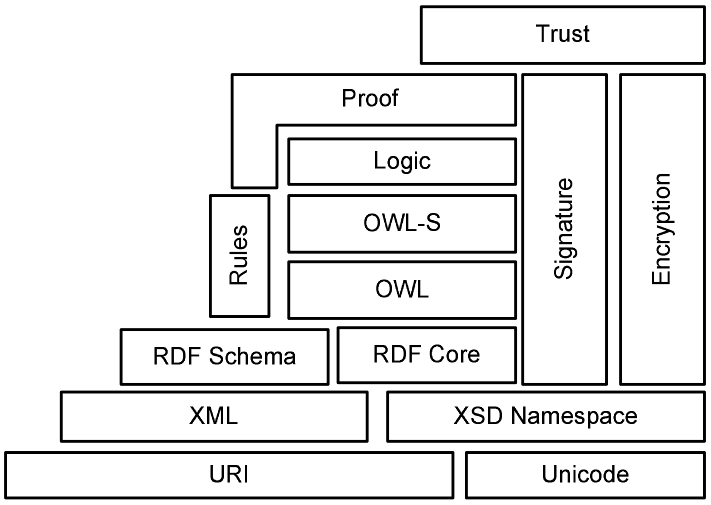


Figure 2.1: Semantic pyramid for Semantic Web

is more effective discovery, automation, integration, and reuse across different applications.

Figure 2.1 illustrates how Semantic Web languages are built upon XML and climbs up the Markup Language Pyramid. Ontologies can be developed on RDF and RDF schemas, but limited to a subclass hierarchy and a property hierarchy with domain and range definitions. For example, RDF can not express equivalence between properties and expressiveness of concept disjointness, union and so on. W3C has defined a more expressive language which is on top of the RDF and RDF schema. OWL is available in three flavours OWL-Light, OWL-DL and OWL-full [49] which are specialization of predicate logic. OWL-DL provide a syntax that fits well with Web languages and also define reasonable subset of DL that offer a trade-off between expressive power and computational complexity. The semantic OWL for web services is being developed by W3C which is OWL-S [24] based on OWL.

The purpose of logic, proof and trust layer is to provide similar features to the ones that can be found in FOL. The idea is to state any logical principle and allow the computer to reason by inference using DL reasoner.

2.5 Semantic Web Services

One overarching characteristic of the conventional web services infrastructure is its lack of semantic representation. Because current XML based infrastructure support only for interoperability but lack of understanding each other is the most important problem even in the homogenous platform. Even, according to new proposals for web services infrastructure, developers require to reach explicit agreement on both the way their web service interact and the format of the messages they exchange. Web service interactions are hard-coded. Developers are responsible to modify the web services when the interaction patterns have been changed. Simply, growing number of specifications for web services infrastructure facilitate to tighten the agreement between interoperable services which not lead to more open automatic interaction. Therefore current web service infrastructure is inflexible, limited and expensive to maintain. The lack of explicit semantics prevents web services from acting autonomously by understanding each other messages and what tasks each service perform. Current web service proposals are fail to enable semantic representations. Enriching the web services infrastructure with semantics will let web services [57]

- explicitly express and reason about business relations and rules
- represent and reason about the task a web service performs (for example, selling books or verifying credit card), thus enabling automated service discovery based on explicit advertisements and descriptions of service functionality
- represent and reason about message ordering
- understand the meaning of exchanged messages
- represent and reason about preconditions for using services and the effects of invoking them
- combine Web services to achieve more complex services

The semantic web has the potential to provide the Web services infrastructure with the information it needs through ontologies.

Let us see the infrastructure which takes the semantic capabilities to the current Web services infrastructure. The resulting infrastructure is called semantic web services infrastructure as shown in the figure 2.2.

Ontologies are defined in OWL and gap between semantics and conventional web is filled by the OWL-S. OWL-S defines an upper ontology for describing the properties and capabilities of Web services in OWL. It is intended to enable users and software agents to automatically discover, invoke, compose, and monitor Web resources offering services, under specified constraints. It defines high level constructs such as a service profile: to represent the interfaces of services including inputs, outputs, preconditions and effects, a service (process) model to represent the details of inner working of a service, and a service grounding to provide information about how to use a service. Important to note that all these

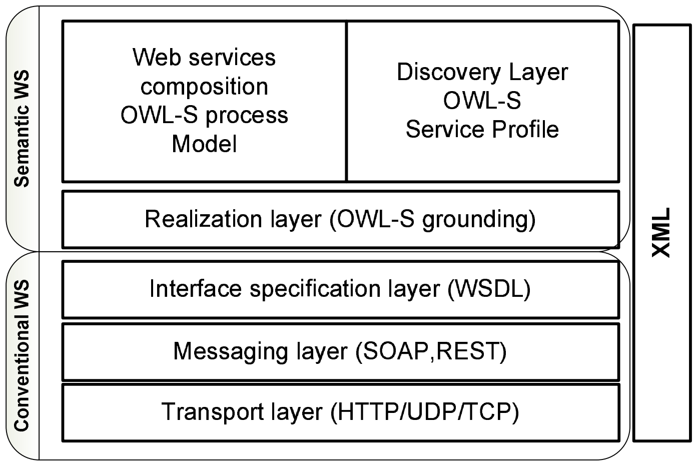

Semantic WS
Web services composition OWL-S process Model
Discovery Layer OWL-S Service Profile
Realization layer (OWL-S grounding)
Conventional WS
Interface specification layer (WSDL)
Messaging layer (SOAP,REST)
Transport layer (HTTP/UDP/TCP)
XML

Figure 2.2: Semantic Web services infrastructure

technologies are markup based. The OWL-S profile relies on ontologies to specify what type of information the web serve advertise and what will be the effects after the execution. At discovery time, on a request discovery process select a web service which match the request. Rest of this thesis discusses this matching problem and propose more elegant solution.

Chapter 3

DL for OWL

People and the technology are the two main drivers of knowledge construction. Ontologies are used for organising knowledge in a structured way in many areas from philosophy to Knowledge Management(knowledge!management) and the semantic web. The name *description logic* is motivated by the fact that the important notion of the domain are described by concept *description*. the key characteristic features of DLs resides in the construction for establishing relationships between concepts.

In this chapter, we do the survey on how DL can be used as Ontology language for Semantic Web [25] and sketch the process we have used to define Personalisation ontology, best practices [54] and limitations [62].

3.1 Ontology Engineering

In any engineering discipline, main success factor is the use of methodology. This is equally applicable to ontology construction. In this section, we will look at the general criteria for and specific properties of methodologies for the *ontology life cycle*. An ontology engineering methodology needs to consider the following three types of disciplines:

- Ontology management disciplines
- Ontology construction disciplines
- Ontology supporting disciplines

Ontology management includes scheduling ontology engineering tasks. As usual, there should be control mechanism and quality assurance procedures. After the decision to build an ontology, the *ontology engineer* needs procedures for ontology construction which specify, conceptualise, formalise, and implement the ontology. Supporting activities include knowledge acquisition, evaluation, integration, merging and alignment, and configuration management.

3.2 Knowledge Discovery

Knowledge Discovery(KD) is the most important for ontology development. It is a process which aims at the extraction of interesting (nontrivial, implicit, previously unknown and potentially useful) information from data in large databases [26] may be the best definition for KD. Knowledge Discovering playing very important role in semi-automated ontology construction. In this section, our main objective is to consider different approaches of KD. The main goal of KD is to find useful pieces of knowledge within from raw data with little or in best case without human interventions. This idea is catalyst for dynamic web service composition. The main question is how to perform automated learning on different kinds of data and especially with different representation languages for representing learnt concepts [26].

Today almost all ontology construction methodologies support for *semi-automatic ontology construction*[5]. Dynamic web services composition of semantic web is far away which should need fully automated ontology construction, according to the literacy survey did for this research. Next section address the semi-automated ontology construction which may be the first step of fully-automated ontology construction of dynamic web services composition.

3.3 ontology construction

Ontology construction is a process which has to be accomplished by the *ontology engineer, domain expert* and ontology user. Ontology construction processes are well defined and enrich form preliminary data mining methodologies. In this section we explain the one of the well known ontology construction process which will use the similar fashion of the ontology engineering described in the above section. This methodology is CRISP-DM [15]. CRISP-DM has six interrelated phases:

1. **Business understanding:** This initial phase focuses on understanding the project objectives and requirements from a business perspective.

2. **Data understanding:** The data understanding phase starts with an initial data collection and proceeds with activities in order to get familiar with the data, to identify data quality problems, to discover first insights into the data, or to detect interesting subsets to form hypotheses for hidden information.

3. **Data preparation**: The data preparation phase covers all activities to construct the final dataset (data that will be fed into the modeling tool(s)) from the initial raw data. Data preparation tasks are likely to be performed multiple times, and not in any prescribed order. Tasks include table, record, and attribute selection as well as transformation and cleaning of data for modeling tools.

4. **Modeling:** In this phase, various modeling techniques are selected and applied, and their parameters are calibrated to optimal values. Typically, there are several techniques for the same data mining problem type. Some techniques have specific requirements on the form of data. Therefore, stepping back to the data preparation phase is often needed.

5. **Evaluation:** At this stage in the project you have built a model (or models) that appears to have high quality, from a data analysis perspective. Before proceeding to final deployment of the model, it is important to more thoroughly evaluate the model, and review the steps executed to construct the model, to be certain it properly achieves the business objectives. A key objective is to determine if there is some important business issue that has not been sufficiently considered.

6. **Deployment:** Creation of the model is generally not the end of the project. Even if the purpose of the model is to increase knowledge of the data, the knowledge gained will need to be organised and presented in a way that the customer can use it.

As shown in the figure 3.1, web services can be define as atomic and composite web services according to the OWL-S [24]. Human involvement is necessary for the fist three phases. We have used integrated environment where Protégé [39, 40, 33] editor has used to develop Personalisation ontology and Pellet [63] DL reasoner for check for consistency as shown in the Fig. 3.2.

3.4 Analysis of Family Ontology

The semantics of axioms is defined as one would expect. In this section, we consider the *Family ontology* explained in [27] as a start. If no symbolic name is defined more than once, this finite set of $\mathcal{T}$ (a terminology of TBox) definitions are unequivocal. In this ontology, fist suppose `Person` and `Female` are primitive concepts (base symbols).

$$\texttt{Woman} \equiv \texttt{Person} \sqcap \texttt{Female} \tag{3.1}$$

This declaration usually interpreted as logical equivalence. This amounts to providing both *sufficient and necessary* condition for classifying an individual

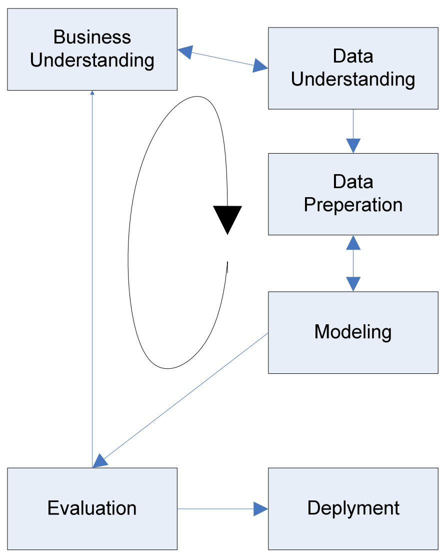


Figure 3.1: CRISP-DM process Model

as a `Woman`. Important to note that DL declaration (logical equivalence) is the strongest DL representation unlike other representations which are capable to provide only *necessary* conditions. The axiom 3.1 introduce a symbolic name (definition): `Woman` for complex atomic concept which is defined as the subclass of `Person` and `Female`. The abstract syntax of 3.1 shown in the figure 3.3 and the OWL code for this axiom is shown in the figure 3.4.

The `intersectionOf` constructor combines two classes, forming their intersection. Individuals who are members of `Woman` are precisely those individuals who are members of both the class `Person` and `Female`.

Let us define `Man` name symbol for following axiom 3.2,

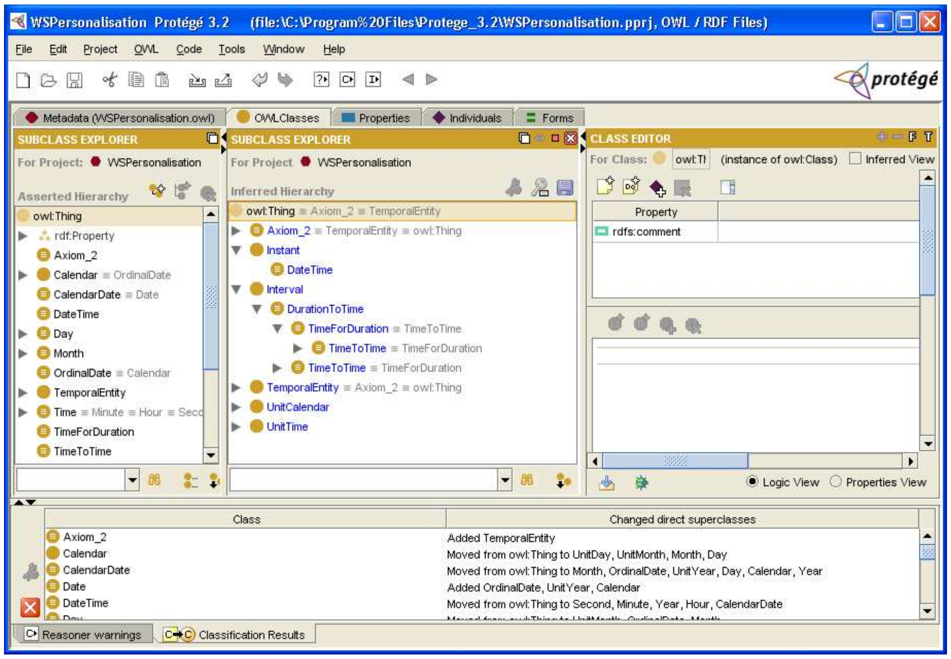
$$\texttt{Man} \equiv \texttt{Person} \sqcap \neg\texttt{Woman} \tag{3.2}$$

The atomic concept `Man` can be defined as subclasses of `Person` and complement of `Woman`. Abstract Syntax is given in the figure 3.5 and OWL code is given in the figure 3.6.

The `complementOf` constructor is applied to a single class and described the collection of individuals who will never be an instance of that class. For example, instance of `Man` never becomes an instance of `Woman` and vice-versa.

$$\texttt{Mother} \equiv \texttt{Woman} \sqcap \exists\texttt{hasChild.Person} \tag{3.3}$$

The axiom 3.3 define `Mother` concept. Mother should be a `Woman` and she should have some children they are type of `Person`. In the figure 3.7 shows the


WSPersonalisation Protégé 3.2 (file:\C:\Program%20Files\Protege_3.2\WSPersonalisation.pprj, OWL / RDF Files)
File Edit Project OWL Code Tools Window Help
protégé
Metadata (WSPersonalisation.owl)
OWLClasses
Properties
Individuals
Forms
SUBCLASS EXPLORER
For Project: WSPersonalisation
Asserted Hierarchy
owl:Thing
rdf:Property
Axiom_2
Calendar ≡ OrdinalDate
CalendarDate ≡ Date
DateTime
Day
Month
OrdinalDate ≡ Calendar
TemporalEntity
Time ≡ Minute ≡ Hour ≡ Seco
TimeForDuration
TimeToTime
SUBCLASS EXPLORER
For Project: WSPersonalisation
Inferred Hierarchy
owl:Thing ≡ Axiom_2 ≡ TemporalEntity
Axiom_2 ≡ TemporalEntity ≡ owl:Thing
Instant
DateTime
Interval
DurationToTime
TimeForDuration ≡ TimeToTime
TimeToTime ≡ TimeForDuration
TimeToTime ≡ TimeForDuration
TemporalEntity ≡ Axiom_2 ≡ owl:Thing
UnitCalendar
UnitTime
CLASS EDITOR
For Class: owl:Tl (instance of owl:Class)
Inferred View
Property
rdfs:comment
Logic View
Properties View
Reasoner warnings
Classification Results

Class	Changed direct superclasses
Axiom_2	Added TemporalEntity
Calendar	Moved from owl:Thing to UnitDay, UnitMonth, Month, Day
CalendarDate	Moved from owl:Thing to Month, OrdinalDate, UnitYear, Day, Calendar, Year
Date	Added OrdinalDate, UnitYear, Calendar
DateTime	Moved from owl:Thing to Second, Minute, Year, Hour, CalendarDate

Figure 3.2: Protégé and Pellet Integrated Environment

```
Class(family:Woman complete
   intersectionOf(
      family:Female
      family:Person
   )
)
```

Figure 3.3: Abstract Syntax for axiom 3.1

abstract syntax for `Mother` and figure 3.8 shows OWL code for axiom 3.3.

The `someValuesFrom` constructor describes those individuals that have a relationship to other individuals of particular class. For instance, the individuals of `Mother` are related via the `hasChild` property to at least one instance of the `Person` class.

$$\texttt{Parent} \equiv \texttt{Father} \sqcup \texttt{Mother} \tag{3.4}$$

The `unionOf` (refer to the figure 3.9) combines two classes, for example individuals form from the axiom 3.4 are either `Father` or `Mother`. This is precisely `Parent` should be either `Father` or `Mother`. As shown in figure 3.10, same

```
 <owl:Class rdf:ID="Woman">
      <owl:equivalentClass>
         <owl:Class>
             <owl:intersectionOf rdf:parseType="Collection">
                 <owl:Class rdf:about="#Female"/>
                 <owl:Class rdf:about="#Person"/>
             </owl:intersectionOf>
        </owl:Class>
   </owl:equivalentClass>
 </owl:Class>
```

Figure 3.4: OWL for axiom 3.1

```
Class(family:Man complete
   intersectionOf(
      family:Person
      complementOf(
         family:Woman
      )
   )
)
```

Figure 3.5: Abstract Syntax for axiom 3.2

```
<owl:Class rdf:ID="Man">
  <owl:equivalentClass>
      <owl:Class>
          <owl:intersectionOf rdf:parseType="Collection">
              <owl:Class>
                  <owl:complementOf rdf:resource="#Woman"/>
              </owl:Class>
              <owl:Class rdf:about="#Person"/>
          </owl:intersectionOf>
      </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Figure 3.6: OWL for axiom 3.2

```
Class(family:Mother complete
   intersectionOf(
      restriction(family:hasChild someValuesFrom(family:Person))
      family:Woman
   )
)
```

Figure 3.7: Abstract Semantic for axiom 3.3

```
<owl:Class rdf:ID="Mother">
  <owl:equivalentClass>
      <owl:Class>
          <owl:intersectionOf rdf:parseType="Collection">
              <owl:Restriction>
                  <owl:onProperty rdf:resource="#hasChild"/>
                  <owl:someValuesFrom rdf:resource="#Person"/>
              </owl:Restriction>
              <owl:Class rdf:about="#Woman"/>
          </owl:intersectionOf>
      </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Figure 3.8: OWL code for axiom 3.3

```
Class(family:Parent complete
   unionOf(
      family:Father
      family:Mother
   )
)
```

Figure 3.9: Abstract Semantic for axiom 3.4

`unionOf` operator used by the OWL.

The effect of equation 3.4 will be described in next section.

3.4.1 Infer the Family Ontology

Let us create individuals `HARRY`, `PETER` and `MARRY` respectively `Person`, `Man` and `Woman` instances. Following are the concept assertions,

$$\texttt{Person(HARRY)},\quad \texttt{Man(PETER)},\quad \texttt{Woman(MARRY)} \tag{3.5}$$

```
<owl:Class rdf:ID="Parent">
  <owl:equivalentClass>
      <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
              <owl:Class rdf:about="#Father"/>
              <owl:Class rdf:about="#Mother"/>
          </owl:unionOf>
      </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Figure 3.10: OWL code for axiom 3.4

Let us define the family relationships, PETER's mother is MARRY, as well HARRY's father is PETER. Then role assertions are,

$$\texttt{hasChild}(\texttt{MARRY}, \texttt{PETER}), \quad \texttt{hasChild}(\texttt{PETER}, \texttt{HARRY}) \tag{3.6}$$

In terms of model theoretic semantics , an ABox $\mathcal{A}$ is consistent with respect to the TBox $\mathcal{T}$. Inference is key to the implicit knowledge. Family TBox $\mathcal{T}$ will be checked for *satisfiability* , then for *subsumption*. Concept Father and Mother subsumed Parent is entailment. In addition to that, Parent subsumed Person is another entailment.

Suppose we have add the following constraints to $\mathcal{T}$

$$\texttt{GrandMother} \equiv \texttt{Woman} \sqcap \exists \texttt{hasChild}.\texttt{Parent} \tag{3.7}$$

Let us consider the Family ABox $\mathcal{A}$, which will *realise* that MARRY is instance of GrandMother.

The *realization problem* shown in the figure 3.11. Due to the *subsumption*, MARRY and PETER will be classified as instance of Parent concept on axiom 3.4. Although both are parents, MARRY is the Mother of PETER according to the 3.6. To satisfy the role, according to the axiom 3.7, instance MARRY should be classified as GrandMother in instance checking of $\mathcal{T}$.

3.4.2 $\mathcal{SHIQ}$ features

Let us point out some of the features of $\mathcal{SHIQ}$ (see appendix B, table B.1) that make this DL expressive enough to be used as an ontology language. For example, to say that Person has at most two children without mentoring the properties of these children, we can use *qualified number restrictions*

$$(\leq 2\texttt{hasChild})$$

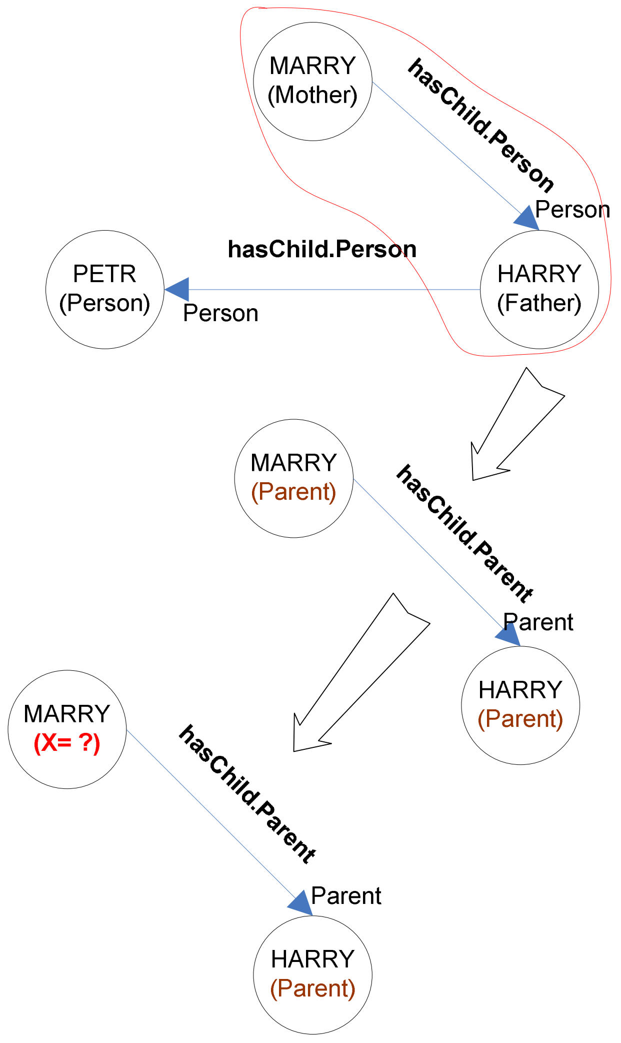


Figure 3.11: Instance checking will result `GrandMother` entailment

To specify that someone has at most one son and at most one daughter

$$(\leq 1\texttt{hasChild}.\neg\texttt{Female}) \sqcap (\leq 1\texttt{hasChild}.\texttt{Female})$$

In $\mathcal{SHIQ}$, we can define *sub role*

$$\texttt{hasParent} \sqsubseteq \texttt{hasAncestor}$$

We can define *inverse role* in $\mathcal{SHIQ}$

$$\texttt{hasParent} \equiv \texttt{hasChild}^{-}$$

Example for complex terminological axiom in $\mathcal{SHIQ}$ is "Persons have Person parents":

$$\texttt{Person} \sqsubseteq \exists\texttt{hasParent}.\texttt{Human}$$

and `hasAncestor` can be defined as *transitive role*

$$\texttt{hasAncestor}^{+} \sqsubseteq \texttt{hasAncestor}$$

How to create `Person` called `Child` who has `Father` and `Mother`. The following axiom implies that

$$\texttt{Child} \equiv \texttt{Person} \sqcap \exists \texttt{hasParent.Mother} \sqcap \exists \texttt{hasParent.Father} \tag{3.8}$$

According to the above `Child` definition (*necessary and sufficient* condition), instance `HARRY` should have at least one `Mother` and one `Father` to classified as `Child`.

Now we are ready to define `Son` and `Daughter`. As in the following axioms

$$\texttt{Son} \equiv \texttt{Child} \sqcap \neg\texttt{Female}$$

$$\texttt{Daughter} \equiv \texttt{Child} \sqcap \texttt{Female}$$

However, in OWL, classifiers are overlapping until disjointness axioms are entered. But in this case is not need because `Daughter` is disjoint from `Son` based on the `Female`. As well as, in OWL, important to model with language such as OWL to be aware of the difference between *necessary* and *sufficient and necessary* conditions. For example, to further elaborate equation 3.8 that is `Child` can have only one `Mother` and one `Father` which is exactly of two. Contrast to equation 3.8 this is only *necessary* condition as shown in equation 3.9

$$\texttt{Child} \sqsubseteq 2 = \texttt{hasChild} \tag{3.9}$$

In OWL, these necessary conditions are represented using the `subClassOf` axiom. For complete Family Ontology, refer to the Appendix C.

Chapter 4

Ontology Validation

All the systems need to be debugged while being developed. DL reasoners performs a considerable variety of inference types, most of the results are unpredictable. For example, someone may intended to conclude equation 3.9 and 3.8 and rewrite as in the following definition.

$$\texttt{Child} \equiv \texttt{Person} \sqcap \exists\texttt{hasParent.Mother} \sqcap \\ \exists\texttt{hasParent.Father} \sqcap 2 = \texttt{hasChild}$$

as a result you will get nothing after the inference. Conventional tracking mechanisms for debugging will not be feasible because the declarative knowledge is compiled into data structures and imperative code in order to achieve efficiency in DL. Therefore it is necessary to investigate Description Logic under different perspectives where debugging is possible. For example, above problem can be solved by normalisation-Compare approach [20], where implicit knowledge is fist explicated into a normal form, after which subsumption can be checked. In this chapter we are going to consider deductive framework where to cast the various problems of the DL. Our framework is in two steps

- Model the domain in terminological definitions.
- The concepts and roles defined in the terminology are used to enter descriptions of domain objects and to retrieve objects matching given descriptions.

Because ABox $\mathcal{A}$ represents infinite interpretations query answering is complex. Over an aBox $\mathcal{A}$, one can pose queries about the relationships between concepts, roles and individuals. The ABox inference on which such queries are based is the *instance checking*, or the check whether an assertion is entailed by an ABox. An assertion α is entailed by $\mathcal{A}$ and we write $\mathcal{A} \vdash \alpha$. For example,

$$\mathtt{Parent(PETER), Man(PETER, HaRRY)} \vdash \mathtt{Father(PETER)} \tag{4.1}$$

4.1 Logical formalism

Although DL is given in a Tarski-style model theoretic way because DL can be translated into *predicate logic* (although DL is variable free, it is a subset of *First Order* formulae). The translation of concepts yields formulae in one free variable because DL concept denotes a set of individuals and it is in variable free syntax. The DL is said to be less expressive than a logic $\mathcal{L}$ if there is a translation that translates all DL-concepts into equivalent $\mathcal{L}$ formulae. Such a translation is called *preserving* [28]. The translation π that translates $\mathcal{ALC}$ into predicate logic.

$$\begin{array}{ll}
\pi_x(a) = a(x) & ,\pi_y(a) = a(y) \\
\pi_x(C \sqcap D) = \pi_x(C) \wedge \pi_x(D) & ,\pi_y(C \sqcap D) = \pi_y(C) \wedge \pi_y(D) \\
\pi_x(C \sqcup D) = \pi_x(C) \vee \pi_x(D) & ,\pi_y(C \sqcup D) = \pi_y(C) \vee \pi_y(D) \\
\pi_x(\exists R.C) = \exists_y.R(x,y) \wedge \pi_y(C) & ,\pi_y(\exists R.C) = \exists_y.R(y,x) \wedge \pi_x(C) \\
\pi_x(\forall R.C) = \forall_y.R(x,y) \rightarrow \pi_y(C) & ,\pi_y(\forall R.C) = \forall_y.R(y,x) \rightarrow \pi_x(C)
\end{array}$$

The translation is given by two mappings π_x and π_y from $\mathcal{ALC}$-concepts into $\mathcal{L}^2$ formulae in one free variable[66]. For further information refer to [28].

4.2 Sequent calculus based validation

The sequent is an expression of the form $\Gamma \vdash \Delta$, where Γ and Δ are multisets of formulae. The symbol $\vdash$ is called "entailment". Therefore "Γ entails Δ means that from all formula in Γ some of the formulae in Δ follows. More information refer to the Appendix D.

The theorems based on subsumption [13] is important to analyse ontologies.

Theorem 4.1. *If* $\mathtt{C_1, C_2}$ *are two concepts and* $\mathtt{R_1, R_2}$ *are two roles,*

$$\frac{\mathtt{C_1 \sqsubseteq C_2}, \quad \mathtt{R_1 \sqsubseteq R_2}}{\mathtt{R_1.C_1 \sqsubseteq R_2.C_2}}$$

Proof: Because DL is subset of First Order Logic (FOL), first translate DL subsume to FOL.

$$\mathtt{C_1 \sqsubseteq C_2} \quad \overset{\mathtt{def}}{=} \quad \forall_{\mathtt{x}} \pi_{\mathtt{x}}(\mathtt{C_1}) \rightarrow \pi_{\mathtt{x}}(\mathtt{C_2})$$

Therefore we can write $\exists \mathtt{R_1.C_1} \sqsubseteq \exists \mathtt{R_2.C_2}$ DL axiom as

$$\vdash \forall_{\mathtt{x}}(\exists_{\mathtt{y}} \mathtt{R_1(x,y)} \wedge \pi_{\mathtt{y}}(\mathtt{C_1})) \rightarrow \exists_{\mathtt{y}} \mathtt{R_2(x,y)} \wedge \pi_{\mathtt{y}}(\mathtt{C_2})$$

in FOL. The derivation tree search from bottom to top manner.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{R_1(z,b) \vdash R_2(z,b)}}{R_1(z,b) \vdash \exists_y R_2(z,y)}\,\exists_r}{\exists_y R_1(z,y) \vdash \exists_y R_2(z,y)}\,\exists_l}{\exists_y R_1(z,y), \pi_y(C_1) \vdash \exists_y R_2(z,y)}\,W_l}{\exists_y R_1(z,y) \wedge \pi_y(C_1) \vdash \exists_y R_2(z,y)}\,\wedge_l
\qquad
\dfrac{\dfrac{\dfrac{}{\pi_y(C_1) \vdash \pi_y(C_2)}}{\exists_y R_1(z,y), \pi_y(C_1) \vdash \pi_y(C_2)}\,W_l}{\exists_y R_1(z,y) \wedge \pi_y(C_1) \vdash \pi_y(C_2)}\,\wedge_l
$$

$$
\dfrac{\dfrac{\dfrac{}{\exists_y R_1(z,y) \wedge \pi_y(C_1) \vdash \exists_y R_2(z,y) \wedge \pi_y(C_2)}\,\wedge_r}{\vdash \exists_y R_1(z,y) \wedge \pi_y(C_1) \rightarrow \exists_y R_2(z,y) \wedge \pi_y(C_2)}\,\rightarrow_r}{\vdash \forall_x(\exists_y R_1(x,y) \wedge \pi_y(C_1)) \rightarrow \exists_y R_2(x,y) \wedge \pi_y(C_2)}\,\forall_r
$$

The above derivation tree, search bottom to top. all the sequents are listed in appendix D.

For example, Using sequent calculus we can do derivation for the following axiom

$$\exists \mathtt{hasParent.Parent} \sqsubseteq \exists \mathtt{hasAncestor.Person}$$

Let us assume, `Parent` and `Person` abbreviate as C_1 and C_2, as well `hasParent` and `hasancestor` abbreviate as R_1 and R_2.

$$\begin{array}{c} \mathtt{Parent} \sqsubseteq \mathtt{Person}, \quad \mathtt{hasParent} \sqsubseteq \mathtt{hasAncestor} \\ \mathtt{iff} \quad \exists \mathtt{hasParent.Parent} \sqsubseteq \exists \mathtt{hasAncestor.Person} \end{array}$$

Corollary 4.1.1. *If* C_1, C_2 *are two concepts and* R *is role,*
$C_1 \sqsubseteq C_2, \quad \mathtt{iff} \quad r.C_1 \sqsubseteq r.C_2$.

Proof: If C_1, C_2 are two concepts and R_1, R_2 are two roles, assume $C_1 \sqsubseteq C_2$ and $R_1 \sqsubseteq R_2$ then $R_1.C_1 \sqsubseteq R_2.C_2$ was proved in the theorem 4.1. Instead of roles subsume, if roles are same as $R_1 \equiv R_2$ then decidability entirely depend on the concept subsume. To satisfy theorem 4.1, $C_1 \sqsubseteq C_2$ assumption should be valid.

Corollary 4.1.2. *If* C *is DL concept and* R_1, R_2 *are roles,*
$R_1 \sqsubseteq R_2, \quad \mathtt{iff} \quad R_1.C \sqsubseteq R_2.C$.

Proof: If C_1, C_2 are two concepts and R_1, R_2 are two roles, assume $C_1 \sqsubseteq C_2$ and $r_1 \sqsubseteq r_2$ then $R_1.C_1 \sqsubseteq R_2.C_2$ was proved in the theorem 4.1. Instead of concept subsume, if concepts are same as $C_1 \equiv C_2$ then decidability entirely depend on the role subsume. To satisfy theorem 4.1, $R_1 \sqsubseteq R_2$ assumption should be valid.

Theorem 4.2. *if* C_1, C_2 *and* C_3 *are DL concepts and* $C_1 \sqsubseteq C_2$*,* $C_1 \sqsubseteq C_3$ *then* $C_1 \sqsubseteq C_2 \sqcap C_3$ *must be true.*

Prof: assume $C_1 \sqsubseteq C_2 \sqcap C_3$ is true to use deductive reasoning to prove this theorem. First, we will define this assumption in FOL as follows

$$\pi_x(C_1) \rightarrow \pi_x(C_2 \sqcap C_3)$$

Derivation tree start with the above assumption

$$\dfrac{\dfrac{\dfrac{}{\pi_{\mathtt{x}}(\mathtt{C_1}) \vdash \pi_{\mathtt{x}}(\mathtt{C_2})} \qquad \dfrac{}{\pi_{\mathtt{x}}(\mathtt{C_1}) \vdash \pi_{\mathtt{x}}(\mathtt{C_3})}}{\pi_{\mathtt{x}}(\mathtt{A}) \vdash \pi_{\mathtt{x}}(\mathtt{C_2}) \wedge \pi_{\mathtt{x}}(\mathtt{C_3})}\ \wedge_r}{\vdash \pi_{\mathtt{x}}(\mathtt{A}) \rightarrow \pi_{\mathtt{x}}(\mathtt{C_2} \sqcap \mathtt{C_3})}\ \rightarrow_r$$

Theorem 4.3. *if* $\mathtt{C_1}, \mathtt{C_2}$ *are DL concepts and* $\mathtt{R_1}, \mathtt{R_2}$ *are DL roles,* $\mathtt{C_1} \sqsubseteq \mathtt{C_2}$ *and* $\mathtt{R_1} \sqsubseteq \mathtt{R_2}$*, then* $\mathtt{C_1} \sqsubseteq \mathtt{C_3}$ *then* $\mathtt{C_1} \sqsubseteq \mathtt{C_2} \sqcap \mathtt{C_3}$ *must be true.*

We can use sequent calculus for query analysis. Given a set of assertions Σ, a query concept Q and individual a, the decision problem in DLs which is relevant to the present context is called instance checking, i.e this check whether every interpretation satisfy all the assertions in Σ also satisfy $Q(a)$. This is just another way of saying that $Q(a)$ is a logical consequence of Σ, written in $\Sigma \vdash Q(a)$. The query 4.1 can be written as

$$(\mathtt{Father} \sqcup \mathtt{Mother})(\mathtt{PETER}), \mathtt{Man}(\mathtt{PETER}) \vdash (\mathtt{Father})(\mathtt{PETER})$$

To save space, abbreviate the `Father, Mother` and `Man` respectively `C, D` and `E`.

$$\dfrac{\dfrac{\dfrac{}{\mathtt{C(x)} \vdash \mathtt{C(x)}}}{\mathtt{C(x)}, \mathtt{E(x)} \vdash \mathtt{C(x)}}\ W_l\ ; \quad \dfrac{}{\mathtt{D(x)}, \mathtt{E(x)} \vdash \mathtt{C(x)}}}{(\pi_{\mathtt{x}}(\mathtt{C}) \vee \pi_{\mathtt{x}}(\mathtt{D})), \pi_{\mathtt{x}}(\mathtt{E}) \vdash \pi_{\mathtt{x}}(\mathtt{C})}\ \vee_l$$

as shown in the above deduction tree, query should be successful. However, above proof is very simple, because we can generally prove that.

4.3 General validation

In this section, we are going to validate entailments explained in the well known ”OWL reasoning examples” article [10]. This article explains the number of entailment examples based on DL reasoners, which are denoted in OWL. Therefore, in this section, all the examples are represented in DL to formally rationalise the entailments. Readers who are interesting to the OWL representation, should refer to the original article [10] for complete information.

4.3.1 Class Inferences

Class Inferences are considered to be one of the most important.

Eg:1 Bus Drivers are Drivers

Following axioms explain that, bus_driver is a person who drives a bus. a bus is a vehicle.

$$\begin{aligned}&\texttt{bud\_driver} \equiv \texttt{person} \sqcap \exists \texttt{drives.bus}\\&\texttt{bus} \sqsubseteq \texttt{vehicle}\\&\texttt{driver} \equiv \texttt{person} \sqcap \exists \texttt{drives.vehicle}\end{aligned}$$

We need to prove the entailment that bus_driver is a driver. Assume that bus_driver drives a vehicle is true. Then bus_driver becomes driver true.

$$\begin{aligned}&(\texttt{bus\_driver})(\texttt{b}) \vdash \texttt{driver}(\texttt{b})\\&(\texttt{person} \sqcap \exists \texttt{drives.bus})(\texttt{b}) \vdash (\texttt{person} \sqcap \exists \texttt{drives.vehicle})(\texttt{b})\end{aligned}$$

from theorem 4.2,

$$(\exists \texttt{drives.bus})(\texttt{b}) \vdash (\exists \texttt{drives.vehicle})(\texttt{b})$$

from the theorem 4.1,

$$\texttt{bus} \sqsubseteq \texttt{vehicle}$$

above axiom is true premise, therefore assumption should be true

$$\texttt{bus\_driver} \sqsubseteq \texttt{driver}$$

The subclass is inferred due to subclasses being used in existential quantification.

Eg:2 Cat owners like Cats

In this example, cat owners have cats as pets because they like pet. The property has_pet is a subproperty of likes, so anyone who has a pet must like that pet.

$$\begin{aligned}&\texttt{cat\_owner} \equiv \texttt{person} \sqcap \exists \texttt{has\_pet.cat}\\&\texttt{has\_pet} \sqsubseteq \texttt{likes}\\&\texttt{cat\_liker} \equiv \texttt{person} \sqcap \exists \texttt{like.cat}\end{aligned}$$

The entailment to prove is cat_owner likes cat;

$$\begin{aligned}&\texttt{cat\_owner}(\texttt{a}) \vdash \texttt{cat\_liker}(\texttt{a})\\&(\texttt{person} \sqcap \exists \texttt{has\_pet.cat})(\texttt{a}) \vdash (\texttt{person} \sqcap \exists \texttt{like.cat})(\texttt{a})\end{aligned}$$

from corollary 4.2, we can deduce

$$(\exists \texttt{has\_pet.cat}) \vdash (\exists \texttt{like.cat})$$

With reference to theorem 4.1 this entailment is true because the

$$\texttt{has\_pet} \sqsubseteq \texttt{likes}$$

, therefore we can conclude that

$$\texttt{cat\_owner} \sqsubseteq \texttt{cat\_liker}.$$

The subclass is inferred due to a subproperty assertion.

Eg:3 Drivers are Grown Ups

Drivers are defined as persons that drive cars. Drivers are adults, so all drivers must be adults.

$$\begin{aligned}&\texttt{driver} \equiv \texttt{person} \sqcap \exists \texttt{drives.vehicle}\\ &\texttt{driver} \sqsubseteq \texttt{adult}\\ &\texttt{grownup} \equiv \texttt{adult} \sqcap \texttt{person}\end{aligned}$$

from theorem 4.2, $\texttt{driver} \sqsubseteq \texttt{person}$,

we have two subsumptions which are	$\texttt{driver} \sqsubseteq \texttt{person}$ and
	$\texttt{driver} \sqsubseteq \texttt{adult}$
therefore from theorem 4.2	$\texttt{driver} \sqsubseteq \texttt{adult} \sqcap \texttt{person}$
therefor,	$\texttt{driver} \sqsubseteq \texttt{grownup}$

an example of axioms being used to assert additional necessary information about a class. We do not need to know that a driver is an adult in order to recognise one, but once we have recognised a driver, we know they must be adult.

Eg:4 Sheep are herbivores

Sheep only eats grass. Grass is a plant. Plants and part of plants are disjoint from animal and parts of animals. Herbivores only eat things which are not animals or parts of animals.

$$\begin{aligned}&\texttt{sheep} \sqsubseteq \forall \texttt{eats.grass} \sqcap \texttt{animal}\\ &\texttt{grass} \sqsubseteq \texttt{plant}\\ &\exists \texttt{part\_of.animal} \sqcup \texttt{animal} \equiv \neg(\texttt{plant} \sqcup \exists \texttt{part\_of.plant})\\ &\texttt{herbivore} \equiv \forall \texttt{eats.}\neg(\exists \texttt{part\_of.animal}) \sqcap \forall \texttt{eats.}\neg(\texttt{animal}) \sqcap \texttt{animal}\end{aligned}$$

The disjointness is a very important concept in the DL because it avoids the instances overlap. according to the above axioms, who eats things which are not animals or parts of animals never be instance of who eats plants and parts of the plants. Simply herbivore never be a carnivore. We can rewrite `herbivore` as follows

$\texttt{herbivore} \equiv \forall\texttt{eats}.\neg(\exists\texttt{part\_of.animal} \sqcup \texttt{animal}) \sqcap \texttt{animal}$

let us substitute disjoint concept,

$\texttt{herbivore} \equiv \forall\texttt{eats}.\neg(\neg(\texttt{plant} \sqcup \exists\texttt{part\_of.plant})) \sqcap \texttt{animal}$

$\texttt{herbivore} \equiv \forall\texttt{eats}.(\texttt{plant} \sqcup \exists\texttt{part\_of.plant}) \sqcap \texttt{animal}$

$\texttt{herbivore} \equiv \forall\texttt{eats.plant} \sqcap \texttt{animal} \sqcup \forall\texttt{eats}.\exists\texttt{part\_of.plant}$

Therefore, $\texttt{herbivore} \sqsubseteq \forall\texttt{eats.plant} \sqcap \texttt{animal}$

let us consider `sheep`,

$\texttt{sheep} \sqsubseteq \forall\texttt{eats.grass} \sqcap \texttt{animal}$

rewrite as $\texttt{sheep} \sqsubseteq \forall\texttt{eats.plant} \sqcap \texttt{animal}$,because $\texttt{grass} \sqsubseteq \texttt{plant}$

therefore, $\texttt{sheep} \sqsubseteq \texttt{herbivore}$

Note the complete definition, which means that we can recognise when things are herbivores.

Eg:5 Giraffes are herbivores

Giraffes only eat leaves. Leaves are parts of trees, which are plants. Plants and parts of plants are disjoint from animals and parts of animals. Herbivores only eats things which are not animals or parts of animals.

$\texttt{giraffe} \sqsubseteq \texttt{animal} \sqcap \forall\texttt{eats.leaf}$

$\texttt{leaf} \sqsubseteq \exists\texttt{part\_of.tree}$

$\texttt{tree} \sqsubseteq \texttt{plant}$

$\exists\texttt{part\_of.animal} \sqcup \texttt{animal} \equiv \neg(\texttt{plant} \sqcup \exists\texttt{part\_of.plant})$

$\texttt{herbivore} \equiv \forall\texttt{eats}.\neg(\exists\texttt{part\_of.animal}) \sqcap \forall\texttt{eats}.\neg(\texttt{animal}) \sqcap \texttt{animal}$

as in the above example, we can write `herbivore` as follows,

$\texttt{herbivore} \equiv \forall\texttt{eats.plant} \sqcup \forall\texttt{eats}.(\exists\texttt{part\_of.plant}) \sqcap \texttt{animal}$

therefore, $\texttt{herbivore} \sqsubseteq \forall\texttt{eats}.(\exists\texttt{part\_of.plant}) \sqcap \texttt{animal}$

as well, we can write `leaf` as

$\texttt{leaf} \sqsubseteq \exists\texttt{part\_of.plant}$ because $\texttt{tree} \sqsubseteq \texttt{plant}$

This subsume can be substituted in `giraffe`

$\texttt{giraffe} \sqsubseteq \texttt{animal} \sqcap \forall\texttt{eats}.(\exists\texttt{part\_of.plant})$

$\therefore, \texttt{giraffe} \sqsubseteq \forall\texttt{eats}.(\exists\texttt{part\_of.plant})$

therefore, $\texttt{giraffe} \sqsubseteq \texttt{herbivore}$

Similar to the previous example with additional inference provided by the existential restriction in the definition of leaf.

Eg:6 Old ladies own Cats

Old ladies must have a pet. All pets that old ladies have must be cats. Cat owners have pets all of them are cats.

$$\texttt{old\_lady} \equiv \texttt{person} \sqcap \texttt{female} \sqcap \texttt{elderly}$$
$$\texttt{old\_lady} \sqsubseteq \forall\texttt{has\_pet.cat} \sqcap \exists\texttt{has\_pet.Animal}$$
$$\texttt{cat\_owner} \equiv \texttt{person} \sqcap \exists\texttt{has\_pet.cat}$$

We have to prove that an old lady is a cat owner who must have a pet that is a cat.

$\texttt{old\_lady} \sqsubseteq \texttt{person}$ and $\texttt{old\_lady} \sqsubset \texttt{forallhas\_pet.cat}$
therfore we can write, $\texttt{old\_lady} \sqsubseteq \texttt{person} \sqcap \forall\texttt{has\_pet.cat}$
more general form is
$\texttt{person} \sqcap \forall\texttt{has\_pet.cat} \sqsubseteq \texttt{person} \sqcap \exists\texttt{has\_pet.cat}$
Therefore, $\texttt{old\_lady} \sqsubseteq \texttt{cat\_owner}$

This is an example of the interaction between an existential quantification (asserting the existence of a pet) and a universal quantification (constraining the types of pet allowed).

Eg:7 Mad Cows are inconsistent

Cows are naturally herbivores. A cow suffering from ”mad cow” disease is a one that has been eating sheep’s brain which is an animal part.

$$\texttt{cow} \sqsubseteq \texttt{herbivore}$$
$$\exists\texttt{part\_of.animal} \sqcup \texttt{animal} \equiv \neg(\texttt{plant} \sqcup \exists\texttt{part\_of.plant})$$
$$\texttt{herbivore} \equiv \forall\texttt{eats}.\neg(\exists\texttt{part\_of.animal}) \sqcap \forall\texttt{eats}.\neg(\texttt{animal}) \sqcap \texttt{animal}$$
$$\texttt{mad\_cow} \equiv \texttt{cow} \sqcap \exists\texttt{eats}.(\exists\texttt{part\_of.sheep} \sqcap \texttt{brain})$$
$$\texttt{sheep} \sqsubseteq \texttt{animal} \sqcap \forall\texttt{eats.grass}$$

We have to show that the cow suffering from the ”Mad Cow” disease is not a herbivore.

from theorem 4.2, $\texttt{mad\_cow} \sqsubseteq \exists\texttt{eats}.(\exists\texttt{part\_of}.\texttt{sheep})$
but $\texttt{sheep} \sqsubseteq \texttt{animal}$ from theorem 4.2
therefore general form is, $\texttt{mad\_cow} \sqsubseteq \exists\texttt{eats}.(\exists\texttt{part\_of}.\texttt{animal})$
we can rewrite $\texttt{mad\_cow} \sqsubseteq \neg\neg\exists\texttt{eats}.(\exists\texttt{part\_of}.\texttt{animal})$
$\texttt{mad\_cow} \sqsubseteq \neg(\forall\texttt{eats}.\neg(\exists\texttt{part\_of}.\texttt{animal}))$
because, $\texttt{herbivore} \sqsubseteq (\forall\texttt{eats}.\neg(\exists\texttt{part_of}.\texttt{animal}))$from theorem 4.2
$\therefore\ \texttt{mad\_cow} \sqsubseteq \neg\texttt{herbivore}$

but from the theorem 4.2, $\texttt{mad\_cow} \sqsubseteq \texttt{cow}$
in turn $\texttt{mad\_cow} \sqsubseteq \texttt{herbivore}$ because, $\texttt{cow} \sqsubseteq \texttt{herbivore}$
This is inconsistent.

Thus a mad cow has been eating part of an animal, which is inconsistent with the definition of a herbivore.

4.3.2 Instance Inference

Conceptual Graphs (CG) are an expressive formalism for representing knowledge about an application domain in a graphical way. To simplify inference of individuals, *Simple Conceptual Graphs*(SGs) are used which is most prominent decidable fragment of the CGs. An SG over the support $\mathcal{S}$ is a labelled bipartite graph of the form $g = (C, R, E, l)$, where C is a set of *concept nodes*, R is a set of *relation nodes*, and $E \subseteq C \times R$ is the edge relation. Each concept node is labelled with a concept type and *referent*, i.e, an individual marker or the generic marker $*$. A concept node is called *generic* if its referent is the generic marker ; otherwise, it is called *individual concept node* . Each relation node is labelled with a indices according to the arity or r. For more information on Simple Conceptual Graphs, refers to the [53].

Eg:8 The Daily Mirror is a Tabloid

He is an adult person therefore he is a man. Mike drives a white van so he is called white van man. A white van man only reads tabloids, and Mick read the Daily Mirror, thus the Daily Mirror must be a tabloid.

$$\texttt{white\_van\_man} \equiv \texttt{man} \sqcap \exists\texttt{drives}(\texttt{van} \sqcap \texttt{white\_thing})$$
$$\texttt{white\_van\_man} \sqsubseteq \forall\texttt{reads}.\texttt{tabloid}$$

Let $\mathcal{N}_C$ be a set of primitive concepts and $\mathcal{N}_R$ is set of primitive roles

$$\mathcal{N}_C := \{\texttt{man}, \texttt{white\_van\_man}, \texttt{van}, \texttt{white\_thing}, \texttt{tabloid}\}$$
$$\mathcal{N}_R := \{\texttt{drives}, \texttt{reads}\}$$

The SG is shown in the figure 4.1(b), which is derived from the above axioms. Let us define $\mathcal{N}_I$ which denotes the set of constants.

$$\mathcal{N}_I := \{\texttt{Daily\_Mirror}, \texttt{Mick}, \texttt{Q123\_ABC}\}$$

The figure 4.1(a) is drawn based on the information provided.

$$\begin{array}{l}\texttt{Daily\_Mirror} : \top \\ \texttt{Mick} : \texttt{male} \\ \texttt{drives}(\texttt{Mike}, \texttt{Q123\_ABC}) \\ \texttt{reads}(\texttt{Mike}, \texttt{Daily\_Mirror}) \\ \texttt{Q123\_ABC} : \texttt{van} \\ \texttt{Q123\_ABC} : \texttt{white\_thing}\end{array}$$

The expanded SG h depicted on the 4.1(b) subsumes g depicted in 4.1(a) because mapping w_0 onto v_0, and w_1 onto v_1, and w_2 onto v_2 yields a homomorphism from h to g. The concept description g can we written from the 4.1(b) as

$$\begin{array}{l} g := \texttt{man} \sqcap \{\texttt{Mike}\} \sqcap \exists\texttt{drives}(\texttt{van} \sqcap \texttt{white\_thing} \sqcap \{\texttt{Q123\_ABC}\})\sqcap \\ \quad \exists\texttt{reads}.(\texttt{tabloid} \sqcap \{\texttt{Daily\_Mirror}\})\end{array}$$

According to that `Daily_Mirror` should be classified as the `tabloid`.

Here we saw interaction between complete and partial definitions
plus a universal quantification allowing an inference about a role filler.

Eg:9 Pete is a Person, Spike is an Animal

In this example, Spike is the pet of Pete, Then Pete must be a Person and Spike must be an Animal. Let us define, $\mathcal{N}_C$ is set of primitive concepts, $\mathcal{N}_R$ is set of primitive roles and $\mathcal{N}_I$ is set of constants.

$$\begin{array}{l}\mathcal{N}_C := \{\texttt{animal}, \texttt{person}\} \\ \mathcal{N}_R := \{\texttt{has\_pet}, \texttt{is\_pet\_of}\} \\ \mathcal{N}_I := \{\texttt{Pete}, \texttt{Spike}\}\end{array}$$

The SG shown in the figure 4.2(a) which is based on the following information.

$$\begin{array}{l}\texttt{Spike} : \top \\ \texttt{is\_pet\_of}(\texttt{Spike}, \texttt{Pete}) \\ \texttt{Pete} : \top\end{array}$$

As shown in the figure 4.2(b), The expanded SG can be drawn from the following information which are already provided.

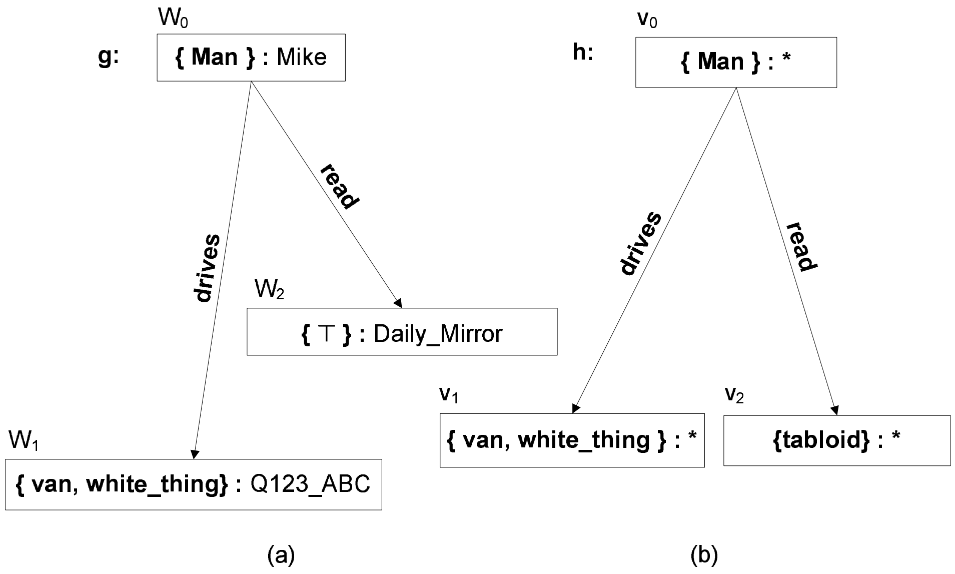


Figure 4.1: SG for the Daily Mirror is a Tabloid

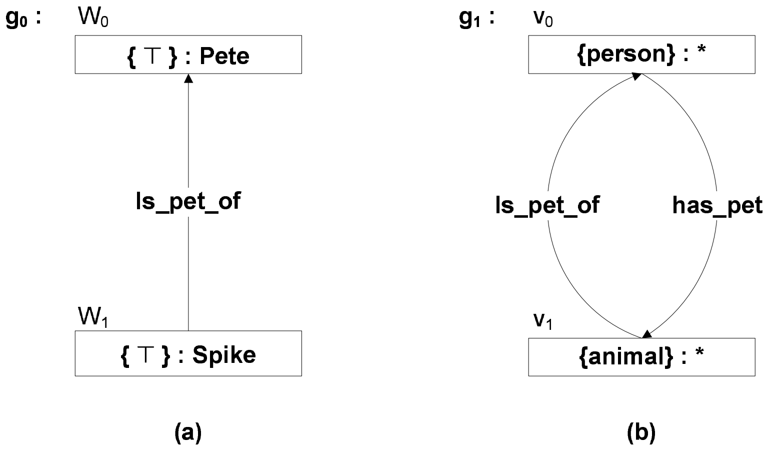


Figure 4.2: SG for the Pet is Person, Spike is an Animal

$$\exists \texttt{has\_pet}.\top \sqsubseteq \texttt{person}$$
$$\top \sqsubseteq \forall \texttt{has\_pet}.\texttt{animal}$$

$$\texttt{is\_pet\_of} \equiv \texttt{has\_pet}^{-}$$

The expanded SG g_1 depicted at right hand side subsumes g_0 because mapping w_0 onto v_0, and w_1 onto v_1 yields a homomorphism from g_1 to g_0. Therefore `Pete` should be a `person` and `Spike` is an `animal` as shown in the figure 4.2.

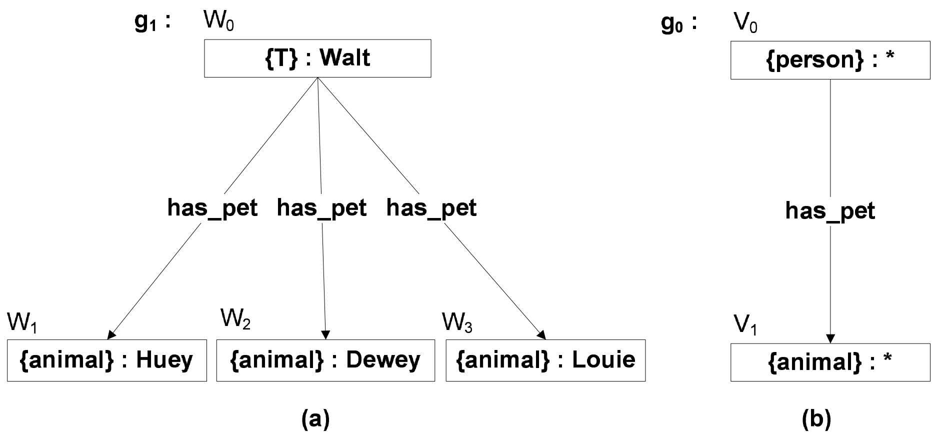


Figure 4.3: SG for the Walt is animal lover

.

Here we saw an interaction between an inverse relationship and domain and range constraints on a property.

Eg:10 Walt loves animals

In this example, Walt is an animal lover. He has three pets named Huey, Dewely and Louie. They are all distinct individuals. Walt has at least three pets and is thus an animal lover. In this example,

$$\begin{array}{l}\texttt{has\_pet}(\texttt{Walt},\texttt{Huey})\\ \texttt{has\_pet}(\texttt{Walt},\texttt{Dewely})\\ \texttt{has\_pet}(\texttt{Walt},\texttt{Louie})\\ \texttt{animal\_lover} \equiv \texttt{person} \sqcap 3 \leq \texttt{has\_pet}\end{array}$$

The expanded SG can be drawn from the above information and are depicted in the figure 4.3(a).

In the figure 4.3, the expanded SG g_1 depicted at right hand side subsumes g_0 because mapping w_0 onto v_0 and so on yields a homomorphism from g_1 to g_0. Therefore, Walt is person who is animal lover.

Chapter 5

Ontology for Personalisation

5.1 Introduction

A web service must be well defined. The description of the service should be well described means semantically rich and well structured. We want that structure to exploit the services based on *functional* and *nonfunctional* properties. For example a nonfunctional property of a service is its temporal availability. One dimension of *temporal availability* for services can be considered constraining when a service can be accessed by a service requester.

Use of semantic techniques liaison to the selection of services motivate to select the best suite service among semantically suitable services otherwise hinder the support for sophisticated service discovery, service selection, automated service negotiation and dynamic service substitution. The primary motivation behind our research is to utilise this semantic richness of service descriptions to personalise the web service.

5.2 Ontology for Personalisation

The lack of description of non-functional service properties; hinder the support of sophisticated, personalised service selection, automated service negotiation and dynamic service substitution. Issues that arise for service requesters include the inability to compare services objectively, insufficient details about either or both the functional and the non-functional properties, no common language for the description of services, and a lack of mechanisms to assist with effective filtering of relevant services. We believe that web services need to be well described by means of semantic rich. It is necessary to capture common concepts such as time (i.e. temporal), place (i.e. locative), price, payment, discounts, obligations, penalties, rights, security, trust and quality which are regularly encounter independently of the services[44].

5.2.1 Whole-part issues

Whole and part relation is very important for the personalisation ontology. However, RDF Schema and OWL does not contain specific primitives for whole-part relations. *Transitiveness* (refer to B.2.1) is one of the solution. The Expressive power of the transitiveness is important. For example, in a temporal model, a second is part of a minute, a minute is a part of an hour, and then a second is a part of an hour due to the transitivity. Part-whole relation in DL is still not clear. However, OWL provides a general construct for declaring properties to be transitive with `partOf` property [60]. We have create *common* Ontology where `partOf` and `hasPart` defined. In common ontology, `partOf` is inverse of the `hasPart` as denoted in the axiom 5.1 and transitivity as in the axiom 5.2.

$$\texttt{partOf} \equiv \texttt{hasPart}- \tag{5.1}$$
$$\texttt{partOf} \equiv (\texttt{hasPart}-)^{+} \tag{5.2}$$

In this case we have a hidden assumption that *hasPart* is a kind of part attribute [3]. However, $\texttt{hasPart} \equiv \texttt{partOf}-$ is the only inference you will find in this ontology. For complete common ontology, refer to the appendix C.2. Let us create an ontology which will be useful to describe the personalisation ontology.

5.2.2 Temporal Model

The temporal model [55] is the foundation for service availability. A More descriptive time ontology for semantic web is explained in [32], here we'll explain new simple time ontology which is not similar to the ones mentioned above but neither will deviate from them considerably.

Time may be the most important among the temporal information. Temporal ontology has been developed in response to this need in conjunction with OWL-S. This sub-ontology of time covers topological relations among instants and intervals. There are two subclasses for `TemporalEntity`, those are `Instant` and `Interval` as in the axiom 5.3 and 5.4. According to the concept definition (necessary and sufficient) in the axiom 5.5, `TemporalEntity` is either `Instance` or `Interval`.

$$\texttt{Instant} \sqsubseteq \texttt{TemporalEntity} \tag{5.3}$$
$$\texttt{Interval} \sqsubseteq \texttt{TemporalEntity} \tag{5.4}$$
$$\texttt{TemporalEntity} \equiv \texttt{Instant} \sqcup \texttt{Interval} \tag{5.5}$$

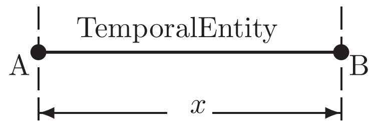


Figure 5.1: Definition for the TemporalEntity

Temporal entities have *begins* and *ends*, for the beginning point A in the Figure 5.1 and the end point B in the Figure 5.1 of the temporal entity if they exist and unique (where $x \neq 0$). A and B can be lie on each other when $x = 0$. Hence, we define `begins` and `ends` object properties which are equals to each other by the axiom 5.8 . These properties are partial. The property `begins` is a functional property, the domain is `TemporalEntity` and its range is `DateTime`.

$$\exists\texttt{begins}.\top \sqsubseteq \texttt{TemporalEntity} \tag{5.6}$$
$$\top \sqsubseteq \forall\texttt{begins.DateTime} \tag{5.7}$$
$$\texttt{ends} \equiv \texttt{begins} \tag{5.8}$$
$$\texttt{DateTime} \equiv \texttt{Date} \sqcap \texttt{Time} \tag{5.9}$$

The concept `DateTime` is defined as intersection of `Date` and `Time` in the axiom 5.9. Although `DateTime` define in the Temporal ontology, `Date` is defined in the separate *Calendar ontology* (see the section 5.2.2) which is imported to *Temporal Ontology*. `DateTime` is range of `begins`. From inference, `DateTime` is range of the `ends` from the axiom 5.8.

The beginning and end are not identical for proper interval. Proper interval is a subclass of `Interval` and is disjoint with `Instant`

$$\texttt{ProperInterval} \sqsubseteq \texttt{Interval} \tag{5.10}$$
$$\texttt{ProperInterval} \sqsubseteq \neg\texttt{Instance} \tag{5.11}$$

Proper interval is true `Interval` where $x \neq 0$ forever. We can define `ProperInterval` as a subclass of the `Interval` as in the axiom 5.10 and disjoint with the `Instance` as in the axiom 5.11.

When used to describe a temporal interval, points in time act as boundary positions (i.e. the start or end position) of an interval. We consider at all times, to have at least the following properties: hours (a unit of 60 minutes), minutes (a unit of 60 seconds), seconds (the smallest unit of time that we choose to represent), and a time zone (expressed as a positive or negative offset from -12 to + 12). This offset is expressed according to Coordinated Universal Time (UTC), and includes hours and minutes. In addition to the offset we capture a region that the time zone relates to. This is via a type of locative entity called 'Region' which will be discussed in-depth in the next section. Here, `unitTime` is an atomic property,

$$
\begin{aligned}
\texttt{Second} &\equiv \geq 0\,\texttt{unitTime} \sqcap \leq 59\,\texttt{unitTime} \sqcap \forall \texttt{partOf}.\texttt{Minute} && (5.12)\\
\texttt{Minute} &\equiv \geq 0\,\texttt{unitTime} \sqcap \leq 59\,\texttt{unitTime} \sqcap \forall \texttt{partOf}.\texttt{Hour} && (5.13)\\
\texttt{Hour} &\equiv \exists \texttt{hasPart}.\texttt{Minute} \sqcap \\
&\quad \geq 0\,\texttt{unitTime} \sqcap \leq 23\,\texttt{unitTime} && (5.14)\\
\texttt{Time} &\equiv \texttt{Hour} \sqcap \texttt{Minute} \sqcap \texttt{Second} && (5.15)\\
\texttt{Time} &\sqsubseteq \texttt{Instance} && (5.16)
\end{aligned}
$$

The `Second` has been defined in the axiom 5.12, where the `unitTime` varying between 0 to 59. Second is part of the `Minute` which in turn part of the `Hour` due to the transitiveness of the `partOf` object property defined in the common ontology. In the axiom 5.13, `Minute` defined as a parts of the `Hour` and can be varied between 0 and 59. The necessary and sufficient condition for `Hour` has been defined using `hasPart` which is converse of the `partOf` object property. As defined in the axiom 5.14, `Hour` is composition of the `Minute`. The `Time` is defined using above concepts in the axiom 5.15 which is intersection of `Hour`, `Minute` and `Second`.

We are in the position to discuss the inference of these concepts. First of all it is necessary to justify axiom 5.14 where `Second` is not mentioned. Not necessary to define `Second` explicitly because transitiveness of the `hasPart` implicitly defined the composition of the `Second` through the `Minute`. Therefore in the axiom 5.14, existential quantification has been used instead of value restriction to aggregate `Minutes` to `Hour`, this may allow implicitly define $\texttt{Hour} \sqsubseteq \exists\texttt{hasPart}.\texttt{Second}$. Rest of the inferences are shown in the following inclusion axioms,

$$
\begin{aligned}
\texttt{Minute} &\sqsubseteq \texttt{Second} && (5.17)\\
\texttt{Hour} &\sqsubseteq \texttt{Minute} && (5.18)\\
\texttt{Time} &\equiv \texttt{Hour} && (5.19)
\end{aligned}
$$

All the above inferences are occurred due to the transitiveness of the `partOf`. Last inference yield that n^{th} (here, $0 \leq n \leq 23$) hour of the day. This may be important to denote particular recurring time for every day. Another inference is `Duration` is subsumed by the `Time` ($\texttt{Duration} \sqsubseteq \texttt{Time}$) due to the axiom 5.22.

Let us define the Time-To-Time type which has a specific *begins* and an *ends* time. Suppose T_S and T_E are respectively the start and end time, then to qualify the following, $T_S < T_E$.

$$
\begin{aligned}
\texttt{TimeToTime} &\equiv\ = 1\,\texttt{begins} \sqcap = 1\,\texttt{ends} && (5.20)\\
\texttt{TimeToTime} &\sqsubseteq \texttt{Interval} && (5.21)
\end{aligned}
$$

By default due to the axiom 5.21, `TimeToTime` is subsumed by the `Interval`. It has one `begins` time and one `ends` time as defined in the axiom 5.20. Time for duration define the start time and the duration (see axiom 5.22) which is type of `Interval` (see axiom 5.23). There are two definitions which are defined same in the axiom 5.15 and the axiom 5.22, but there is a difference, that is `Time` is subsumed by the `Instance` and the `Duration` is subsumed by the `Interval` as necessary conditions. Hence, no contraction because the `Time` and the `Duration` will not overlapped each other.

$$\begin{aligned}
\texttt{Duration} &\equiv \exists\texttt{hasPart.Hour} \sqcap \\
&\quad \exists\texttt{hasPart.Minute} \sqcap \exists\texttt{hasPart.Second} && (5.22)\\
\texttt{Duration} &\sqsubseteq \texttt{Interval} && (5.23)\\
\texttt{TimeForDuration} &\sqsubseteq\ = 1\,\texttt{begins} \sqcap \texttt{Duration} && (5.24)\\
\texttt{DurationToTime} &\equiv\ = 1\,\texttt{ends} \sqcap \texttt{Duration} && (5.25)
\end{aligned}$$

The `Duration` is subsumed by the concept `Time`, this is very basic subsume inference.

The end time can be derived by adding the `Duration` to the start time `begins` as in the axiom 5.24. Duration to time defines the duration and the end time as in the axiom 5.25.

The most important inference is `TimeForDuration` $\equiv$ `DurationToTime`. This is what we have expected, because both the concepts define the same thing using `begins` and `ends`. The concept `TimeToTime` subsumes the `TimeForDuration` and the `DurationToTime` is the next important inference you can find.

$$\begin{aligned}
\texttt{TimeForDuration} &\sqsubseteq \texttt{TimeToTime}\\
\texttt{DurationToTime} &\sqsubseteq \texttt{TimeToTime}\\
\\
\texttt{TimeForDuration} &\sqsubseteq \texttt{Duration}\\
\texttt{DurationToTime} &\sqsubseteq \texttt{Duration}\\
\\
\texttt{TimeForDuration} &\equiv \texttt{DurationToTime}
\end{aligned}$$

The concept `Time` is subsumed by the `Interval` as well as `Instance`. This is an interesting result because `Time` can be either the duration or the instant time. For example, 12.30 can be either 12.30pm or 12 and half hours. To avoid conflicts, `Duration` has been defined which is subsumed by the `Time` only in inference. The concept `ProperInterval` has been entirely ignored disjunction of `Interval` and `Instance` which allowed circumstance where `A` and the `B` points to be fallen on each other ($x = 0$) regarding to the figure 5.1.

Temporal Dates

Our discussion of dates is limited to the Gregorian calendar. Date can be classified in to three subtypes such as calendar dates, ordinal dates and week. The concept of Calendar date called *CalendarDate* can not be defined completely with inclusion only. Therefore, Calendar dates are described using a year(see the axiom 5.28) number, a month (see the axiom 5.27) number between 1 and 12 (representing the months of January through December) and a day (see 5.26) of the month by a number between 1 and 31. Concepts such as `Year`, `Month` and `Day` can be defined using the atomic type `UnitDay`.

$$\begin{aligned}
\texttt{Day} &\equiv \geq 1\,\texttt{UnitDay} \sqcap \leq 31\,\texttt{UnitDay} \sqcap \texttt{partOf.Month} && (5.26)\\
\texttt{Month} &\equiv \geq 1\,\texttt{UnitDay} \sqcap \leq 12\,\texttt{UnitDay} \sqcap \texttt{partOf.Year} && (5.27)\\
\texttt{Year} &\equiv \texttt{hasPart.Day} \sqcap \texttt{hasPart.Month} && (5.28)
\end{aligned}$$

Inference of these three basic concepts are as follows

$$\begin{aligned}
\texttt{Year} &\sqsubseteq \texttt{Month}\\
\texttt{Month} &\sqsubseteq \texttt{Day}
\end{aligned}$$

The concept `Date` defined in the axiom 5.29 is important for other ontologies. For example, Temporal ontology is using `Date` to define `DateTime` concept in the axiom 5.9.

$$\begin{aligned}
\texttt{Date} &\equiv \texttt{hasPart.Day} \sqcap \texttt{hasPart.Month} \sqcap\\
&\quad\ \texttt{hasPart.Year} && (5.29)\\
\texttt{CalendarDate} &\equiv \texttt{Day} \sqcap \texttt{Month} \sqcap \texttt{Year} && (5.30)
\end{aligned}$$

The `CalendarDate` can be defined by intersecting `Year`, `Month` and `Day` concepts using equality defined in the axiom 5.30. Then the inference result is as follows

$$\begin{aligned}
\texttt{CalendarDate} &\sqsubseteq \texttt{Year}\\
\texttt{Date} &\sqsubseteq \texttt{CalendarDate}
\end{aligned}$$

Ordinal dates are a combination of a day of the year a number between 1 and 366 (catering for leap years) and a number for the year.

$$\texttt{OrdinalDate} \equiv \ \geq 1\,\texttt{UnitDate} \sqcap \ \leq 366\,\texttt{UnitDate} \sqcap \texttt{partOf.Year} \qquad (5.31)$$

In the inference, `Month` is subsumed by the `OrdinalDate`. This means `OrdinalDate` can be converted to the `Month` then `Year` and last even to the `Date` or `CalendarDate`.

$$\begin{aligned} \texttt{Month} &\sqsubseteq \texttt{OrdinalDate} \\ \texttt{WeekOfYear} &\sqsubseteq \texttt{OrdinalDate} \end{aligned}$$

Week dates are defined using a number for the day of the week (where 1 - 7 identifies the days from Sunday to Saturday respectively), a number for the week per year (indicated by a number between 1 and 52), and a number for the year. You can define a new concept for the Week of a year as in the axiom 5.32.

$$\texttt{WeekOfYear} \equiv \quad \geq 1\,\texttt{UnitDate} \sqcap \leq 52\,\texttt{UnitDate} \sqcap \texttt{partOf.Year} \tag{5.32}$$

In the inference, `CalendarDate` is subsumed by the `WeekOfYear`.

We might want to say that Date must be either Calendar date, Ordinal date or Week date.

The length of time between the start time and the end time of an interval we refer to as the temporal duration.

5.2.3 Locative Model

The locative model that we have chosen to present the non-functional property of location. The Locative Model acts as a foundation along with the temporal model for availability of the service. In addition to describing where the service is available to provide the service, it is also useful to present the distance between requester and the provider. A Particular position on the earth can be identified by, latitude and longitude coordinates which are described in degrees.

$$\texttt{hasMinute} \sqsubseteq \texttt{hasPart} \tag{5.33}$$

$$\texttt{Degree} \equiv \quad \geq 1\,\texttt{typeOf.Minute} \sqcap \leq 60\,\texttt{typeOf.Minute} \tag{5.34}$$

$$\texttt{Latitude} \equiv \texttt{Degree} \tag{5.35}$$

$$\texttt{Longitude} \equiv \texttt{Degree} \tag{5.36}$$

$$\texttt{Point} \equiv \texttt{Latitude} \sqcap \texttt{Longitude} \tag{5.37}$$

Region is the abstraction which we use to represent countries, republics, states, and territories. Therefore Region has a type and a name, in addition to that region to be a bounded collection of points which are used to describe an area. Define the Area which should have at least three points to define an area surface. This can be shown in a relationship-role in DL,

$$\texttt{Area} \equiv \geq 3\,\texttt{hasPoint.Point}$$

Now you can define a City, Sate, Country, Province, Territory, Republic and Continent separately. These are disjoint classes. However, due to the limitation of the article only the definition of the Continent is given here, the rest can be defined the same way using an attribute name.

$$\texttt{Asia} \equiv \texttt{Continent} \sqcap \forall \texttt{regionOf}.\{\texttt{ASIA}\}$$

These continents are disjoint from each other. For example,

$$\texttt{Asia} \equiv \neg \texttt{America}$$

The continent should also be defined in the ontology using nominals [22] which are the individuals existing in the same ontology;

$$\begin{aligned}\texttt{Continent} = \{&\texttt{EUROPE}, \texttt{ASIA}, \texttt{AMERICA},\\ &\texttt{ANTARTICA}, \texttt{AFRICA}, \texttt{OCEANIA}\}\end{aligned}$$

$$\texttt{Continent} \sqsubseteq \texttt{Area} \sqcap \texttt{name.String}$$

Define Region as follows in this ontology

$$\begin{aligned}\texttt{Continent} \sqsubseteq\ &\texttt{City} \sqcup \texttt{State} \sqcup \texttt{Province} \sqcup\\ &\texttt{Country} \sqcup \texttt{Republic} \sqcup \texttt{Continent}\end{aligned}$$

5.2.4 Service Availability

Service availability refers to the combined use of temporal and locative aspects of web services. Requests to providers to get their services as well as inquiry service capabilities which include service availability. Locative availability can be one of the primary concerns which are bias to a preferred set of locations. Temporal availability may be the other concern. In addition, the capability request can be one of which contains issue resolution, feedback and etc.

$$\begin{aligned}&\texttt{Request} \equiv\\ &\quad\{\texttt{Capability}, \texttt{IssueResolution}, \texttt{Feedback}, \cdots\}\\ &\texttt{Request} \sqsubseteq \texttt{Duration} \sqcap \texttt{Region}\end{aligned}$$

Sender (called Consumer in the web service arena) who sends the Request and the Receiver (called Provider in the web service arena) who has received the service are two sides of the Request that need to be considered. Sender inquires about the available time while the receiver advertises the available time as a responsibility of the Provider. For example, suppose QUERY (from sender) and ADVERTISEMENT (from receiver) are two individuals referring to the same ontology. It can be expressed in DL as follows;

$$\begin{aligned}&\texttt{TimeToTime} \sqcap \texttt{Region}(\texttt{QUERY})\\ &\texttt{DurationToTime} \sqcap \texttt{Region}(\texttt{ADVERTISEMENT})\end{aligned}$$

Although *TimeToTime* (Instant) is different from *DurationToTime* (Interval), both are a type of Duration. The Inference engine should be capable of converting these individuals to either *TimeToTime* or *DurationToTime* before assertion in ABox.

5.3 Matchmaking Example

This ontology has been extended to incorporate to the OWL-S as explained in [45]. According to [45] the *ServiceProfile* class becomes the common super class for concept Advertisement, Query and so on .

$$\begin{aligned}&\texttt{ServiceProfile} \sqsubseteq \top\\&\texttt{Advertisement} \sqsubseteq \texttt{ServiceProfile}\\&\texttt{Query} \sqsubseteq \texttt{ServiceProfile}\end{aligned}$$

One of the primary services that can be found in many domains is Delivery within the available duration. Delivery describes the structure of delivery information by specifying one delivery location, delivery date and time. Suppose Actor is an atomic concept. Then we can define ServiceProfile and Delivery as follows;

$$\begin{aligned}&\texttt{Consumer} \sqsubseteq \texttt{Actor}\\&\texttt{Provider} \sqsubseteq \texttt{Actor}\\&\texttt{ServiceProfile} \sqsubseteq\\&\quad (= 1\,\texttt{providedBy.Provider}) \sqcap\\&\quad (= 1\,\texttt{requestedBy.Consumer}) \sqcap\\&\quad \cdots \&/ * \,\texttt{about service}\, * /\& \cdots\\&\quad (= 1\,\texttt{canDelivery.Delivery})\end{aligned}$$

Let us define Delivery using the availability, delivery date, and delivery location using the personalized ontology we have developed.

$$\begin{aligned}&\texttt{Delivery} \sqsubseteq\\&\quad (= 1\,\texttt{deliveryLocation.Region}) \sqcap\\&\quad (= 1\,\texttt{deliveryDate.Date}) \sqcap\\&\quad (= 1\,\texttt{deliveryAvailable.Duration})\end{aligned}$$

The service is represented by the input and output properties of the profile. The input property specifies the information that the service requires to proceed with the computation.

$$\texttt{canDelivery} \sqsubseteq \texttt{input}$$

So far we have created TBox [27] axioms for ontology, let us look at ABox [27] assertions which is for assert by DL reasoner.

In advertisement, there are some restrictions on the delivery. For example, next delivery is possible after the date 20th of June, 2005 within the time of that day 5pm to next day 2am to Asia. According to DL notations, the *ADV* is the individual of concept *Advertisement*.

$$
\begin{array}{l}
\texttt{Advertisement}\,(\texttt{ADV})\\
\texttt{Delivery}\,(\texttt{DELI})\\
\texttt{canDelivery}\,(\texttt{ADV}, \texttt{DELI})\\
\texttt{deliveryLocation}\,(\texttt{DELI}, \texttt{ASIA})\\
\texttt{deliveryDate}\,(\texttt{DELI},\\
\quad \texttt{CalendarDate}\,(\,\texttt{Day}\,(20) \wedge \texttt{Month}\,(6) \wedge\\
\qquad\qquad \texttt{Year}\,(2005)\,)\,)\\
\texttt{TimeToTime}\,(\texttt{TIME}-\texttt{TO}-\texttt{TIME})\\
\texttt{start}\,(\texttt{TIME}-\texttt{TO}-\texttt{TIME},\\
\qquad\qquad \texttt{START}-\texttt{DATE}\,)\\
\texttt{end}\,(\texttt{TIME}-\texttt{TO}-\texttt{TIME},\\
\qquad\qquad \texttt{END}-\texttt{DATE}\,)\\
\texttt{DateTime}\,(\texttt{START}-\texttt{DATE})\\
\texttt{DateTime}\,(\texttt{END}-\texttt{DATE})\\
\texttt{START}-\texttt{DATE} = (\\
\quad \texttt{CalendarDate}\,(\,\texttt{Day}\,(20) \wedge \texttt{Month}\,(6) \wedge\\
\qquad \texttt{Year}\,(2005) \wedge \texttt{Time}\,(\texttt{Hour}\,(17)) \wedge\\
\qquad\quad \texttt{Minute}\,(00) \wedge \texttt{Second}\,(00)\,)\\
\texttt{END}-\texttt{DATE} = (\\
\quad \texttt{CalendarDate}\,(\,\texttt{Day}\,(21) \wedge \texttt{Month}\,(6) \wedge\\
\qquad \texttt{Year}\,(2005) \wedge \texttt{Time}\,(\texttt{Hour}\,(2)) \wedge\\
\qquad\quad \texttt{Minute}\,(00) \wedge \texttt{Second}\,(00)\\
)\\
\texttt{deliveryAvailable}\,(\texttt{DELI},\\
\qquad\qquad \texttt{TIME}-\texttt{TO}-\texttt{TIME}\,)
\end{array}
$$

The above advertisement is created according to reasoning in ABox. However, ABox reasoning is less effective than TBox reasoning.

Similar to the advertisement, we can define a Query. The query and the advertisements are almost identical because both of them are subsumed by the ServiceProfile. For simplicity and clarity, we will present the advertisement and query specifications tabular format here on wards. For example, ADV specification can be stated as shown in Table 5.1. In this example the advertisement is exactly equal to the Query. Therefore ratings are in the highest level that is called *Exact* and the given value is 4. In this case, the provider can guarantee the service completion as expected by the consumer. The Advertisement will not always satisfy all the attributes of a Query. For example consider the query "product needs to be received at least on or before 20th of June, 2005 within

Table 5.1: Specifications For Advertisement And Query For Exact

Attribute or Role	**Value**	
	Advertisement	**Query**
Location	Asia	Asia
Due Date	20/06/2005	20/06/2005
Availability	20/06/2005 - 17:00:00	20/06/2005 - 17:00:00
	21/06/2005 - 02:00:00	21/06/2005 - 02:00:00

Table 5.2: Specifications For Advertisement And Query For Plug-in

Attribute or Role	**Value**	
	Advertisement	**Query**
Location	Asia	Asia
Due Date	20/06/2005	20/06/2005
Availability		20/06/2005 - 01:00:00
		21/06/2005 - 18:00:00

Table 5.3: Specifications For Advertisement And Query For Subsume

Attribute or Role	**Value**	
	Advertisement	**Query**
Location	Asia	Asia
Due Date	20/06/2005	20/06/2005
Availability	20/06/2005 - 17:00:00	
	21/06/2005 - 02:00:00	

one working hour at our Asia office" against the advertisement "next delivery is possible after the date 20th of June, 2005 to Asia". According to Table 5.2, the provider may get late to deliver the product within the working hour. Most probably the delivery may get delayed until the next day, that is 21st June 2005. In this example, there is a possibility that the service happen out of the consumer preferences. However, the service will be completed because the advertisement is more general than the query. This circumstance is called *plug-in* and rated as value 3. The success of the service used depends on the query. Most of the time,

Table 5.4: Specifications For Advertisement And Query For Intersection

Attribute or Role	**Value**	
	Advertisement	**Query**
Location	Asia	Asia
Due Date	20/06/2005	20/06/2005
Availability	20/06/2005 - 17:00:00	20/06/2005 - 09:00:00
	21/06/2005 - 02:00:00	21/06/2005 - 18:00:00

the provider advertises constraints to the consumers. As a result, the consumer query becomes more general than the advertisement. For example, consider the advertisement "next delivery is possible after the date 20th of June, 2005 within 5pm to 2pm the next day to Asia". The query against the advertisement is "product needs to be delivered to our Asia office." Although provider demands more than the consumer, still the consumer can use the services if he can fulfil the provider's requirements. This kind of circumstance is called *subsume* and rated as 2. The success of the service use depends on the advertisement (see Table 5.3).

Let us consider the query against the first advertisement (ADV) that is the "product needs to be received at least on or before the date 20th of June, 2005 during working hour to our Asia office". There is an overlap between the query and the advertisement. According to Table 5.4, the delivery time is limited to around 6 hours. In this case, concept *Duration* is plays an important role. The Advertisement and query intersect due to the duration overlap. Therefore this situation is called *intersection* and rated with a value 2.
The next situation is where the advertisement and the query are completely incompatible. For example, according to the advertisement, the delivery available only to America and the query demands the delivery to Asia. This circumstance is called *disjoint* and rated with a value 0.
However, this example does not completely describes the entire ontology explained.

5.3.1 Preference Engineering

Personalisation of web services requires a powerful preference model. In this section we summaries the suitable preference model for semantic matching explained in the section 6.4.5. This preference model is brought from the [46]. Although this preference model has been originally developed for database systems, it is applicable to web services and semantics [47].

Intuitively people express their wishes in the form "I like A better than B".

Since this can be understood by everybody, it is a natural way of modeling user preferences. Mathematically such preferences can be characterized by *strict partial orders*.

Preferences can be engineered inductively as follows. Depending on the application domain, a set of pre-defined so-called *base preferences* are assumed to be defined. Given single preference, more complex ones can be gained by means of three fundamental operators.

1. *Pareto preference*:
$$P := P_1 \otimes P_2 \otimes \cdots \otimes P_n$$
   P is a combination of equally important preferences, following the well-known Pareto principle.

2. *Prioritise preference*:
$$P := P_1 \& P_2 \& \ldots \& P_n$$
   P evaluates more important preferences earlier, similar to a lexicographical ordering. P_1 is most important, P_2 next etc.

3. *Ranked* preference:
$$P := rank_F(P_1, P_2, \ldots, P_n)$$
   P combines individual numerical scores $SCORE(p_i)$, $1 \leq i \leq n$, by means of some ranking function F.

For example, from the section 5.3 we can define the following preference query. Contemplating about her or his personal customer preferences, comes up with this list first

$$P_1 = (Location, < P_1) := \\ POS/POS(Location, POS_1set\{Asia\}; POS_2set\{Europe\}) \tag{5.38}$$
$$P_2 = (DueDate, < P_2) := \\ AROUND(DueDate, 20/06/2006) \tag{5.39}$$
$$P_3 = (Availability, < P_3) := BETWEEN(Availability, \\ [20/06/2005 - 17:00:00, \quad 21/06/2005 - 02:00:00]) \tag{5.40}$$
$$P_4 = (Price, < P_4) := LOWEST(Price) \tag{5.41}$$

In the equation 5.38 defines the location as POS/POS preference which says a desired value "Asia" should be from a set of favourites that is set POS_1set. Otherwise, it should be from a set of positive alternates that is set POS_2set. The POS/POS is non-numerical based preference constructor. In this case preferred location is "Asia" but if favourite is not available, then possible value is "Europe".

In the equation 5.39 defines the due date as AROUND numerical based preference. In this preference operator, a desired value should be an explicitly stated value such as due date 20/06/2006. If this is not feasible, values with shortest distance apart from 20/06/2006 will be acceptable.

In the equation 5.40 defines the availability using numerical base BETWEEN operator. According to this operator, a desired value should be between the bounds of an explicitly sated interval. If this is not feasible values with shortest distance apart from the interval boundaries will be acceptable.

In the equation 5.41 defines the price of the product using numerical based LOWEST preference operator. A desired value should be as low as possible.

Let us decides about the relative importance of these single preferences:

$$Q_1 = (\{Location, DueDate, Availability, Price\}, < Q_1) := (P_4 \& (P_2 \otimes P_3) \& P_1)$$

In this query Q_1, highest priority has been given to P_4 which is price. Second highest priority for P_2(due date) and P_3 (availability), which are equally important. Least priority has been given to the P_1 that is Location.

Chapter 6

Matchmaker

For the above mentioned matchmaking example explained in the section 5.3, matchmaker plays a key role. In this chapter we are going to consider common matchmaking algorithms and optimization mechanisms under the logical framework which combines preferences and semantics to deliver personalised service. In general, semantic matchmaker should fulfil the following requirements [36];

- High degree of flexibility and expressiveness : As in the above advertisement the flexibility and the expressiveness entirely depends upon the scope of the ontology and the abstract characterisation (expressiveness supports by the DL language selected to create the ontology) of that ontology.
- Support for Data types: Attributes such as quantities or price depends upon primitive data types such a *+integer* and *+double*. General XML support these data types.
- Semantic level of agreement: Both the service requester and the providers should refer to the same ontology to understand each other in a semantic agreement.

In this chapter we survey the existing approaches to discover Semantic Web Service (SWS) discovery. First, the architectural components are discussed. Subsequently, a wide range of algorithms that has been proposed are presented which is the key objective of this chapter.

6.1 Architectural Significances

Automation of the composition of existing web services to form an execution flow is a challenging problem [17]. A fundamental problem in Web service composition (WSC) is to identify the most appropriate component services in order to generate an optimal composition plan. The best possible composition plan can be created dynamically composition which is an automated process taking place at run time rather than static composition at design time as predefined. Such a plug-and-play service composition scheme significantly reduces the effort involved in the development and management of complex services, increases the Quality of Service (QoS) of the composition, and is a critical step towards interactive, flexible, and collaborative E-business [6]. In many situations, service selection requires the consideration of service consumer's preferences for a particular requirement but the standard web services are not designed nor implemented for such personalised service. Service profiles can be a key artifact for personalisation. A Service profile starts with a static profile and gets further complicated by automatic customisation while gathering potential changes of user needs and preferences in subsequent user-system interactions. Information personalisation has been recognised as one of the best solution to cope with information overload. The overall goal of the personalised service provisioning has to be the fulfilment of individual user needs expressed as complex task which are typically further divided into simpler sub-goals and subsequently matched to different services [9].

Let us consider the general process of service discovery and selection . While Fig. 6.1 depicts the components of a typical service discovery and selection, personalisation has been added and extend the process. The *Service Provider* offers some services to *service consumer/requester* upon request. A *service request* declares the intention of the service requester regarding the desired service functionality. It should be formal functional definition which share the common ontology shared by the service advertisement. Service consumer may be enterprise application agent, mobile agent , human user or any other system that adapt the web service client paradigm. Service implementation located at the service provider is called *service instance*, but interchangeably we use word service for service instance in this thesis. Each provider promote service via service description which is called *service advertisement* in this case. A service advertisement is analogous to a service request but from the provider's perspective. These advertisements are published in *service registry* which has global access of point. Service registry is an implementation of a directory service. It indexes the service advertisements. In addition to that, advertisements are classified according to their respective domains and the service provider. Although service registry don't have actual service instance, but only binding to service instance is available. Service request match against the advertisements, for that matchmaker(see section 6.4.5) do the semantic matching instead of conventional syntax based matching. Matching algorithm tightly coupled with the service registry. Therefore, this algorithm significantly affect the discovery quality of discovery and overall registry performance. Higher the quality of the discovery; overall registry performance is drastically reduced. There can be a number of services

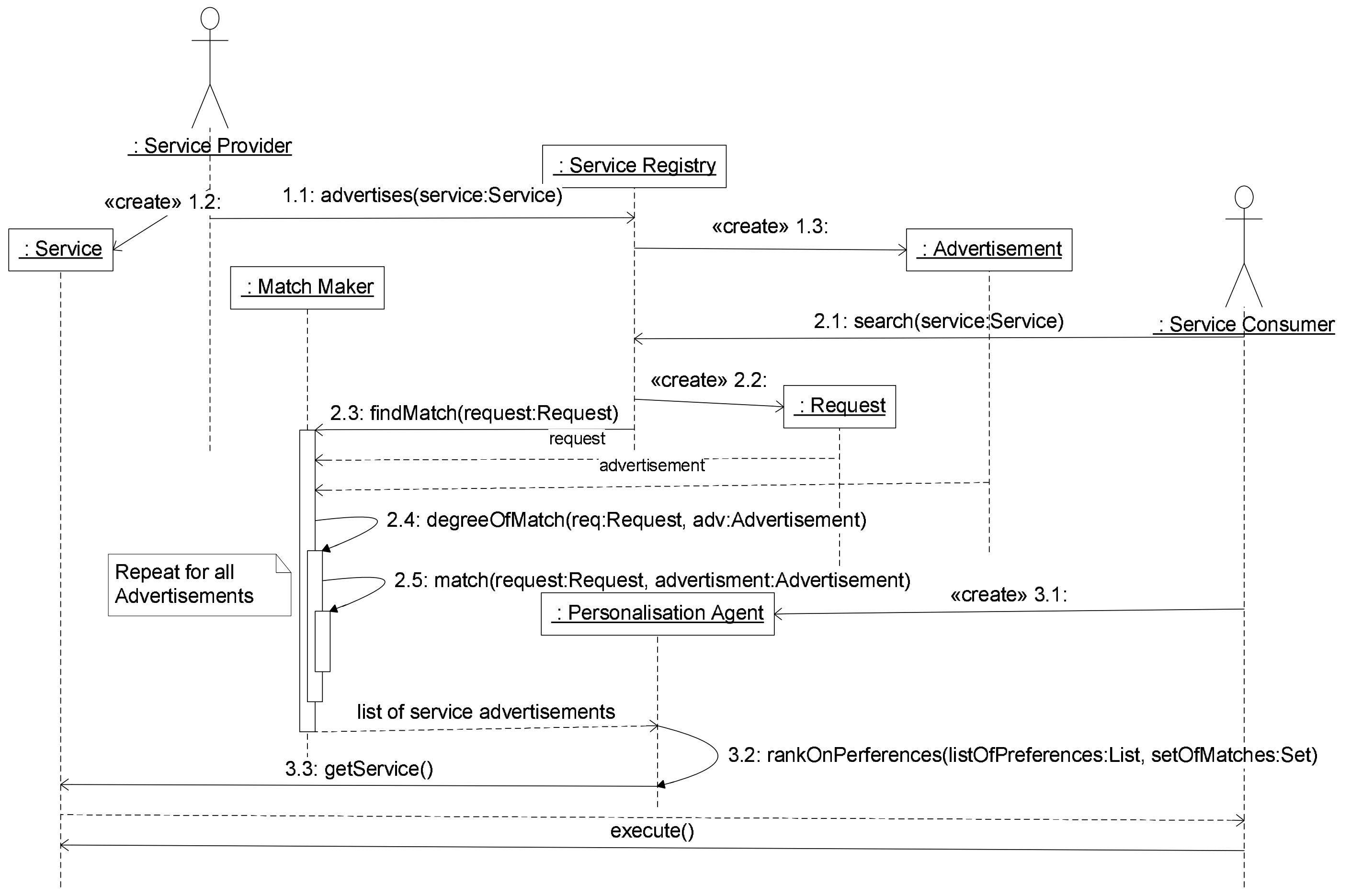

: Service Provider
: Service Registry
«create» 1.2:
1.1: advertises(service:Service)
: Service
«create» 1.3:
: Advertisement
: Match Maker
2.1: search(service:Service)
: Service Consumer
«create» 2.2:
: Request
2.3: findMatch(request:Request)
request
advertisement
2.4: degreeOfMatch(req:Request, adv:Advertisement)
Repeat for all Advertisements
2.5: match(request:Request, advertisment:Advertisement)
«create» 3.1:
: Personalisation Agent
list of service advertisements
3.2: rankOnPerferences(listOfPreferences:List, setOfMatches:Set)
3.3: getService()
execute()

Figure 6.1: Service discovery general architecture

which are equally compatible with the service request. To select the best suitable service among them is the task of the personalisation agent. Personalisation will select the best suitable service based on the preferences provided by the service consumer.

6.2 Survey on selection and composition

Most of the recent research found Web service discovery, selection and composition as an important problem, but they have paid little attention to web services selection and personalisation compared to dynamic composition. Some of the interesting researches are described bellow

- Agents representing autonomous service consumers and providers collaborating to dynamically configure and reconfigure services based software applications. Web Service Agent Framework (WSAF) [52] incorporates service selection agents that use the QoS ontology and an XML policy language .
- User centred, Mobile agent based, Fault injection-equipped and Assertion oriented (UMFA) approach which assist service requesters to select trustworthy web service components [68].
- Resource Description Framework (RDF) [50] and Web Ontology Language (OWL) [49] based automated web service client augmented with semantic models [19].

Some of the key multi-agent system (MAS) components which will be directly influenced by Service Oriented Computing (SOC) are ontology, process model, choreography, directors and facilitators, service level agreement and QoS. Service level agreements and QoS measures are directly related to automated negotiation and flexible service execution in dynamic environment. This is one of the challenges SOC presents which are able to tackle without MAS concepts and techniques [48]. For example, Web Service Agent Proxy [51] is one of the very early web service selection architectures proposed. Middle agents serve as proxies for web services to assist an application in selecting implementations that best match the equality criteria of the application is the approach of this architecture. Proxy agent is instantiated for each service that is planned to use. User can have a list of preferred service providers that are to be chosen when no clear reputable service is found. Using a ranking scheme, proxy agents can use various algorithms to infer the reputation of the service for instance and decide on the selection of the service for this particular interaction.

In UMFA approach service selection implies an established level of trust between these components. The testing and certification of a web service by different service requesters may differentiate with each other because requesters may intend to use different scenarios and environments. Therefore web service selection should be user-centred.

In semantic approach, richer semantics can support greater automation of service selection and composition. Annotation ontology OWL-S [24] provides a semantic based language which is capable of specifying the function of an operation and semantic types of each of the inputs and outputs of the service. OWL-S is an OWL ontology with three interrelated sub-ontologies known as *service profile*, *process model* and *grounding* as shown in the Fig. 2.2. Profile is used to express "What service does?" for purpose of advertising, constructing service requests, and matchmaking. The service profile provides high level view of a given web service. Service profile is divided into three main parts

1. A textual service description and provider's contact information. These information are captured by the human users.

2. A functional description of the service, which contain semantic information base on the IOPEs.

3. List of additional parameters use to describe other features of the service. For example, QoS.

The process model describes "how it works?" to enable invocation, enactment, composition, monitoring and recovery. The process model specifies the tasks a Web service promises to accomplish. The grounding maps the constructs of the process model onto WSDL. OWL constructs are capable of capturing knowledge artifacts modelled using the UML augmented with propositional-logic-based constructs asserted as inter-link constraints [61]. For the specification of process' precondition and effects, OWL-S allows for the use of language such as RuleML [12] which is more expressive than OWL. OWL-S and UDDI complement each other [23]. Because UDDI provides a very weak discovery mechanism which does not allow the discovery of any web service based on service description that is OWL-S provides, OWL-S doesn't provide support for querying for services based on their service description [1]. OWL-S attempts to fill the gap between the OWL and Web services by adding formal content representations and reasoning about interactions and capabilities to Web service specification.

6.3 Solution Architecture

Dynamic composition of services is not provided by OWL-S. Proposed framework is capable of dynamic composition because it provides more planning logic in order to enable the services themselves to compose as composite at the framework level. A solution is proposed whereby the designer is freed at least to some extent from the selection process, allowing both the user and the system to work together to find and use the best service available. The *Service annotation ontology* such as OWL-S define semantic models describe functionality, execution flow, invocation details and so on. Negotiations such as user preferences may be the simplest way to select web services which fulfil the requirements. Semantic preference are shared ontologies. Combine use of annotation ontology such as OWL-S and domain ontology such as Semantic preferences avoid the user introversion. Perhaps two or more web services may have identical syntax but for different purposes. This is called *semantic suitability problem*. The information provided by an OWL-S description of a web service includes an ontological description of the input, output, pre-conditions and effects are simply called IOPEs defined as follows

- *Inputs:* Required input for service execution.
- *Outputs:* Accepting result after the service execution.
- *Preconditions:* Conditional prerequisites for service execution.
- *Effects:* Side effects of the service execution. This imply the change of the state after successful and unsuccessful execution.

Semantic suitability is more problematic. IOPEs information is not enough to resolve this problem by itself. For example, consider equation $2 \quad opt \quad 2 = 4$. In this example *opt* may be either addition or multiplication because IOPEs are same for both operations. Consider a service that takes a geographic region as input and produces the names of different wines as output. But this input/output couple can be used by two different services which are supposed to accomplish different tasks such as one report which wines are produced in that region and the other reports wines that are sold in that region is another good example explained in [23] about semantic suitability problem. Therefore more information such as user preferences is more important in selecting right choice. As shown in the Fig. 6.2, Matchmaker will return an ordered list of web services which are most closely matched with the desired service. Services are matched based on the ontological classifications of their inputs and outputs. These services are ranked according to the best fit which is the best match of the advertised service by the Service Profile but not what the user prefers. Few services may semantically be suitable if they score closely on the top of the list. The QoS experience of past uses could be measured to decide the best fit is one option. The method used to evaluate services based on the evaluation function is another way. In this thesis we propose to optimized the algorithm which is called *cluster analysis*.

Evaluation function guides matchmaker to rank web service among semantically similar services. Evaluation function can be either rule based, Nave Bayes or

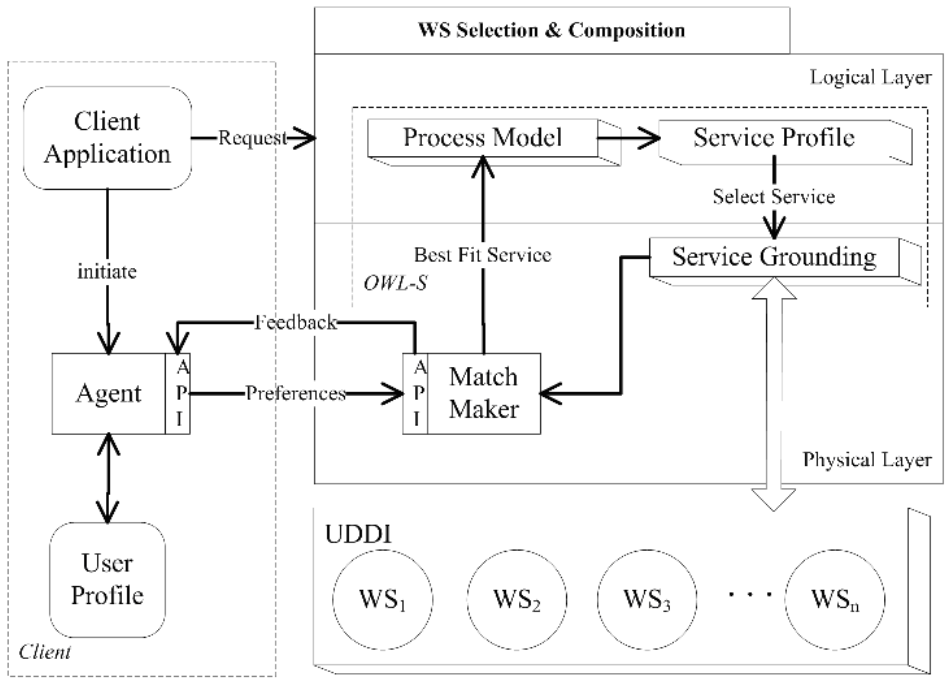


Figure 6.2: OWL technology stack for Matchmaker Architecture

simple statistic based. User preferences are input to all these methods to promote more personalised web services to select while ranking. As shown in the Fig. 6.2, the proposed framework based on the agent technology which will automatically update the user profile in a timely manner is an advantage for B2B applications. User preferences are recorded in the *user profile*. However, user can change the baseline of the user profile whenever he wants. An agent needs to be initiated by the client application.

Primary design concern is separation between abstractions and its implementation. Both the service profile and the process model are abstract representation, only the grounding deals with the concrete level of the specification [24]. At implementation level, basic building block of the WSDL is an operation, OWL-S provide a analogous building block that is atomic process which is primarily characteristic in term of IOPEs. The flexibility of having one or more grounding for a single semantic web service implies common rule of possibility of having many implementations for an abstraction contract. According to this architecture, UDDI must be extended to have semantic capabilities [64]. UDDI can be extended mapping standard semantic annotations (example service profile) UDDI entries via tModel(see section 6.4.4)UDDI. The main constraint is not to deviate from standard UDDI API in this approach.

6.3.1 Grounding related issues

Grounding related issues are discussed in this section. WSDL does not provide any sort of formal way to integrate with OWL. As a result, finding the location of a web service base on specification is possible but not based on the basis of what problems it solves. OWL-S groundings provide the mechanism by which OWL-S semantic web services are mapped to executable services. In this sense, OWL-S complements the UDDI which is the repository of WSDL files. UDDI matchmaker can be extended with OWL matchmaker which is shown in Fig. 6.2. Defining grounding is a cumbersome work, but there are tools [29] that can be used.

6.3.2 Capability Matching

This framework is an extension to UDDI where web services advertise their capabilities. As explained in the above section, integration of UDDI matchmaker with OWL matchmaker allows capability matching. Capability matching compares the requester capabilities with the capabilities advertised by any of the services. The perfectly matching services are unacceptable but very closely matching services can be found. Matching algorithms use service descriptions of WSDL and ontologies provided by service profiles to decide whether there is a match between request and advertisements.

6.4 Approach to Dynamic Composition

In this section, we will propose theoretical framework based on the solution architecture described in the above section. First we describe the existing standards and available technologies for semantic WSs.

6.4.1 Advertise Services

Service profile [24] for advertising and discovering services. Service profile describes the incorporated functionalities of the service by public interface. Each service may have one or more functionalities incorporated. However, there is no constraint to advertise all the functionalities. The decision of what functionalities to publicise is depends on how the service is used. OWL-S provides following information

- Properties of service profile: Links to the Service class and process model.
- About profile: Contact information.

- Functionality Description [24]: Capabilities of the system.
- Profile attributes [24]: Additional details such as QoS attributes (response delay, cost).
- Service Category: Categories of services

Functionality description of the service describes the service in terms of IOPEs. For example, amazon.com like online book selling service need to have either the title of the book or the ISBN to directly query the book. After execution, output should be confirmation that the order has been received. Before execution precondition should be satisfied such as valid credit card. At the end effect will be charged from credit card.

6.4.2 Semantic Service Discovery

Semantic service annotations such as OWL-S has been introduced in order to automate the whole service lifecycle which is advertisment to service execution. Three factors mainly affect service discovery

1. The ability of service providers to describe their services.
2. The ability of service requester to describe their requirements.
3. The intelligence of the service matchmaking algorithm.

The OWL-S service profiles rely on ontologies to specify what type of information the Web service reports and its consequences at the end of any execution at any time. On request, the discovery process selects Web service provider profiles that match the request. Capabilities are encoded to UDDI using tModel [2] (tModel is used to define the technical specification for a service). Reasoning engine in Matchmaker inference the capabilities to differentiate between web services. The Matchmaker receives Web service advertisements and service requests to compare a service request against available service advertisements.

6.4.3 Operate Services

The OWL-S process model is a workflow of processes. Each process is described IOPE. OWL-S has atomic and composite processes [24]. Control flow is composed of composite processes. Atomic process can be invoked directly in a single interaction. Atomic process provides the grounding. Simple process can be used as an abstraction of atomic process or simplified version of composite process. Composite processes composed from other processes and control constructs [24].

6.4.4 UDDI Integration

WSs are described using Web Service Description Language (WSDL). The mapping is fixed, that is for one service for one WSDL which contain service name and its overall service functionality. In addition to that, WSDL contain details about each operation's name and its inputs and outputs parameters. WSDL is important because that is the only document that describe the WS according to the current standards. UDDI is the global access point where WSDLs are published. Therefore UDDI functionality is to represent data and metadata about Web services [65, 56]. There are four primary data type in a UDDI registry, those are;

- **businessEntity**: The businessEntity provides information about a business of publisher(service provider), and can contain one or more businessServices. Such a information includes provider's name, description and contact detail.
- **businessService**: The business descriptions for a Web service are defined in a businessService. BusinessService contains bindingTemplates.
- **bindingTemplate**: The bindingTemplate contains technical details of Web service, including reference to one or more tModels.
- **tModel**: The tMode define the technical specification for a Web service. This is usually a reference to external technical specifications (for example, WSDL).

Inquiry and publishing functions represent the core data management tools of a UDDI registry [2].

6.4.5 Matchmaker

In our work, Matchmaker can not be a simple as searching by keywords or tokens rather it must match semantic descriptions of capabilities. A particular advertisement is suitable to the request then only the service relevant to that advertisement will be selected as solution to fulfil the service requested. But matching for suitability is complicated by the factor that providers and requesters have different objectives. Therefore service requesters get a set of services which are selected on the Degree of Matching (DoM). Following are the responsibilities of the Matchmaker

- Based on the available ontologies, Matchmaker needs to compare a request and advertisements to select the most suitable web services.
- Matchmaker must measure the quality of the service by averaging the difference between what the service really does compared to what it advertises.

- Service need to be stop of using if difference is considerably high.

The matching algorithm is responsible to do *cluster analysis*. This will cluster the services into categories based on their association strength. As shown in the sequence diagram in the Fig. 6.1, the `degreeOfMatch()` function which distinguishes multiple DoMs. The DoM will be explained in the section 6.5. Using OWL, the matching process performs inferences on the subsumption hierarchy leading to the recognition of semantic matches.

Service matching

Let be the set of all advertisements stored by the registry and be the request; the `findMatch()` function can be defined as

$$findMatch(R) = \{A \in \alpha \mid degreeOfMatch(A, R)\} \quad (6.1)$$

The `findMatch()` function will return a set of advertisements which can be a further subset on their degree of matching level assigned to each advertisement. The best level is level 1 then level 2 and so on up to the level 5. Level 5 is the state of incompatible therefore advertisements which are ranked as level 5 will not be selected. As shown in the Algo. 6.1, `findMatch()` function expecting to select only the set of best (lowest level) services based on the cluster analysis.

As shown in the Algo. 6.1, the request R is matched against all the advertisements available in the registry. The match between an advertisement and a request consist of the match of all the outputs of the request against the outputs of the advertisement and all the inputs of the advertisement against the inputs of the request. Let see pseudocode of degreeOfMatch function shown in Algo. 6.2.

The degreeOfMatch function returns one of the above levels base on the Description Logic based inferences. As shown in Algorithm 6.2, the degree of match between outputs or inputs depends on the relationship between the concepts associated.

1. **Exact** level if the advertisement and request are equivalent concepts. Exact level rating is equals to 1.

2. **PlugIn** level if the request is a sub-concept of the advertisement. PlugIn level rating is equals to 2.

3. **Subsume** level if the request is a super concept of advertisement. Subsume level rating is equals to 3.

4. **Intersection** level if the intersection of advertisement and request is satisfied. Intersection level rating is equals to 4.

5. Otherwise **disjoint** level. Disjoint level rating is equals to 5.

Algorithm 6.1. *findMatch Algorithm*

```
findMatch(R) [ $R$ - Request]

        $setOfMatches = \emptyset$ [ Initialize to empty set ]
        $\alpha = \{A_1, A_2, ..., A_n\}$ [ $\alpha$ - Set of all UDDI advertisements ]
        $clusterLeve = 5$ [ Initialize cluster level to 5 that is the disjoint level ]

        for $\forall A_i \in \alpha$ do [ $A_i$ - Advertisement ]
            $level = 5$ [ $level$ is initialised to 5 ]
            $level =$ degreeOfMatch$(R, A_i)$ [ call compatible method ]

            if $(level < clusterLevel)$ [ $0 < level \leq 5$ ]
              then
                    $clusterLeve = level$ [ Set new cluster level ]
                    $setOfMatches = \emptyset$ [ Initialize to empty set ]
            end if

            if $(level == clusterLeve)$ [ $0 < level \leq 5$]
              then $setOfMatches = setOfMatches \cup \{A_i\}$
            end if

        end for

        return $setOfMatches$
```

The cost of matching between requests and advertisements is fixed and cheap compared to expensive ranking of web services based on preferences. In this algorithm demote the current best preferred service when find a new service which is more qualified than prior. In other words, the new service takes the leading place in the `setOfMatches` list. The algorithm has been optimized by ignoring the evaluation of low ranked services if `setOfMatches` already contains the cluster of services ranked as high level. Cluster analysis decides the place of a new service within the cluster if that service is at the same level of current cluster level so far achieved. If the new service is at higher level than the current cluster, level then form a new cluster which is at the same level of the service by discarding prior low level cluster entirely. The worst case is to evaluate all the services, if services come in increasing order of low level(level 4) to high level (level 1).For n number of services incurred cost is nC_s for worst case selection. Probability analysis [16] can be used to analyse the running time of this algorithm, because ranking forms an uniform random permutation of $n!$.

Assume that service arrive in random order. Let X is the random variable which is equals to the number of services evaluated and added to the clusters.

Algorithm 6.2. *degreeOfMatch Algorithm*

```
degreeOfMatch($R$, $A$)

        $l$ = match($R$, $A$)

        if ($l$ ==(A $\equiv$ R )) then
                return 1 [Advertisement is equal lent to request]
        end if

        if ($l$ == (R $\sqsubseteq$ A)) then
                return 2 [Advertisement subsumes request ]
        end if

        if ($l$ == (A $\sqsubseteq$ R)) then
                return 3 [Request subsumes the advertisement]
        end if

        if ($l$ == (A $\sqcap$ R)) then
                return 4 [Advertisement and request intersect each other]
        end if

        if ($l$ == (A $\equiv \neg$R)) then
                return 5 [Advertisement and request are disjoint to each other]
        end if
```

Let X_i be the indicator random variable associated with the event in which the i^{th} web service is added to the clusters (current cluster and discarded clusters). Thus,

$$X_i = l\{\text{service } i \text{ is selected }\} = \begin{cases} 0, & \text{if service is not selected} \\ 1, & \text{if service is selected} \end{cases}$$

Service i is selected, when service i is better than each of the services 1 through $i-1$. Service i has a probability of $1/i$ of being qualified than service 1 through $i-1$. Assume that services are presented in a random order. According to the analysis If there are n number of services then service selection cost is $O(C_s \ln n)$ of services which is a significant improvement over the worst case cost.

All these DoM are defined in the section 6.5. Practically, OWL abstract semantic [59] has been used to explain `degreeOfMatch()` function in Algorithm 6.2. The `degreeOfMatch()` function returns any on of the levels explained above. Advertisement is added to the list with the level it has been ranked. Furthermore, only the highest ranked subset should be selected according to cluster analysis.

This subset may contain one or more services.

Rank on preferences

There is no choice to select the best suitable service in a situation where only one advertisement is contained in the subset. The complicated situation is when there is another service advertisement available in the subset. Considered to select the best fit service from the best suited services found. Our solution is preferences based.

As explained above cluster considered to be a set of homogeneous level elements. There are M number of services are selected as best suited services. Suppose i^{th} service is S_i and $i = 1 \dots M$. In user profile,s suppose there are N number of define preference attributes denoted by P_j, here $j = 1 \dots N$. Mapping function $f(x)$ is binary function which is defines as

$$f(x_{ij}) = \begin{cases} 0, \text{ if service preference is not found from the } S_i \text{ Service} \\ \quad \text{for } P_j \text{ user preference attribute} \\ \\ 1, \text{ if service preference is found from the } S_i \text{ Service} \\ \quad \text{for } P_j \text{ user preference attribute} \end{cases}$$

The value of the $f(x)$ depends on the success and failure of the ontological matching of service preferences to user preference attribute. User profile defines preferred weight in $w(y)$ function for each attribute in personalisation ontology. Hence, $Total(S_i)$ is the weighted sum for the service S_i

$$Total(S_i) = \sum_{j=1}^{N} f(x_{ij}).w(y_j) \tag{6.2}$$

The lowest $Total(S_i)$ would be the best suitable service. Suppose the best fit service is S_k, then

$$S_k = min(Total(S_1), Total(S_2), Total(S_3) \dots Total(S_M)) \tag{6.3}$$

Weighted sum of the all services need to be calculated based on the equation 6.2 before finding the best fit from the best services.

Let us consider the `rankOnPreferences()` function given in the Algo. 6.3 which will find the best fit service based on the equation 6.2. The behaviour of this function is described in the sequence diagram shown in the Fig. 6.1. Following assumption are considered

- Preferences are in the List, that is input to the `rankOnPreferences()`. This list *listOfPreferences* is ordered on highest priority preference to lowest priority item.

Algorithm 6.3. *rankOnPreferences Algorithm*

```
rankOnPreferences(List $listOfPreferences$, Set $setOfMatches$)

    $Total(bestFit) = \infty$ [ Initialise $bestFit$ ]
    for $\forall A_i \in setOfMatches$ do [ $A_i$ - Service Advertisement ]

        for $\forall p_j \in listOfPreferences$ do
            $singlePreference = \text{head}(preferences)$
            $preferences = \text{tail}(preferences)$
            $S = A_i$
            $Total(S) = 0$

            while $(S \; != \; \emptyset)$ do [ implement the equation 6.2 ]

                if $(f(x_{ij}) \; != \; 0)$ [ here $x \in S$ ]
                then
                    $Total(S) = Total(S) + f(x_{ij}).w(p_j)$
                    $S = S - \{x\}$
                end if

            end while

            if $(Total(bestFit) > Total(S))$
            then
                $bestFit = A_i$
            end if

        end for

    end for

return $bestFit$
```

- This function will not rank web services which are clustered under the *disjoint* level.
- Due to the similar services (services are different only on name), only the first evaluated service will be only selected. For example, when duplicate services exists in different names.

The complexity of the Algo. 6.3 is $\emptyset(n^3)$ in the worst case. Algorithm can be optimized by reducing the number of preference attributes to be evaluated.

6.5 Matchmaking Algorithm

The skeleton of the Matchmaker has been explained in the above section. In this section we are going to put forward the matchmaking heuristics which will provide the guideline to implementation the Matchmaker algorithm explained above. For simplicity, this section is clearly separated from the above (skeleton of the framework). This algorithm is the implementation of the functions `match` shown in the Algo. 6.2.

Matchmaker has to service for two parties, service consumer and service provider. Since consumer request(`R`) and service provider advertise(`A`) their services, our objective is to measure the level of matching between the request and the advertisement. This situation can be easily represented by a *bipartite graph*[30]. Then two classes of the bipartite graph are request and advertisement.

6.5.1 Exact Level

As explained in the section 6.4.5, Exact level is the situation when the request and the advertisement matches perfectly.

According to the Hall's Marriage Theorem[30], let G be a directed, bipartite graph with disjoint vertex sets V and W in which the edges are directed from vertices in V to vertices in W. There exists an exact matching in G if and only if

$$|S| \leq |X(S)| \texttt{ for all } S \subseteq V \tag{6.4}$$

here,

$$X(S) = \{w \in W | v \in S \texttt{ and } (v, w) \texttt{ is an edge in } G\}$$

.

The theorem 6.4 means that a bipartite graph has a exact matching if and only if $|V| = |W|$ and for any subset of say k nodes (set S) of V there are at least k nodes of W that are connected to at least one of them.

The theorem 6.4 deduce important result that is, if every node of a bipartite graph has the same degree($d \geq 1$), then two sides are perfectly matching as shown in the figure 6.3 where $|R| = |S|$ and $R = \{r_1, r_2, ...\}$ and $A = \{a_1, a_2, ...\}$.

The level *Exact* where all the requirements of request `R` and all the requirements of advertisement `A` are one to one map each other. Now we are in the position to define other levels.

6.5.2 Other Levels

In this section we are primarily interested in the *maximum matching problem*; that is, the problem of finding a matching of the maximum cardinality. Based on the maximum matching, plugin, subsumed and intersection going to be defined.

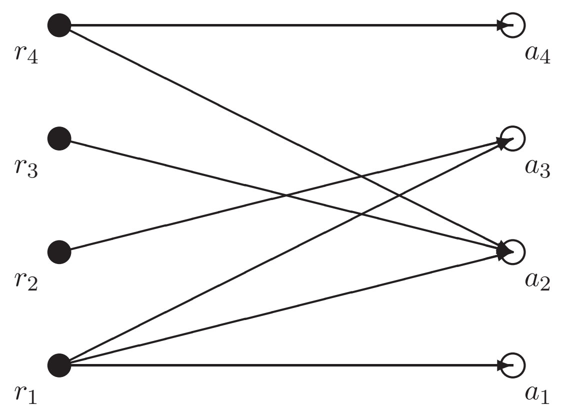


Figure 6.3: The Exact graph

A matching of a graph $G = (V, E)$ is a subset M of E with the property that no two edges in M share a common end. According to Konig Theorem[30], In a bipartite graph the size of a maximum matching is the minimum size of a cover. Thus, each edge of G is incident with at least one vertex of the minimum size of the cover. Moreover, as M is a matching, no two edges of M are incident with a common vertex in the minimum size of cover. If C is a cover of a graph $G = (V, E)$ then, by definition, there are no edges having both ends in $V - C$. A subset Y is called *stable* if there is no edge having both ends in Y. Hence, Y is the complement of the C.

We can find the maximum matching using Ford-Fulkerson algorithm[30] in an undirected bipartite graph $G = (V_r, V_a, E)$ where $V = V_r \cup V_a$, in time polynomial in $|V|$ and $|E|$. As explained in the section 6.4.5, the next level, *plugin* can be defined as follows.

As explained in the section 6.4.5, the next level, *plugin* can be defined as follows.

> The *plgin* is a matching level which is considered only in the occasion of complete matching. Suppose V_a is the set of requirements of advertisement `A` and V_r is the set of requirements of request `R`, then matching level is *plugin* in the context of either $|V_r| > |V_a|$ or $|V_r| < |V_a|$, if $\sum d_r < \sum d_a$. Here, d is the degree of a vertex belongs to minimum cover of $G = (V_r, V_a, E)$ bipartite graph.

According to the result, $\mathtt{R} \sqsubseteq \mathtt{A}$, simply advertisement `A` subsumes request `R`. Therefore, `A` and `R` are in plugin matching.

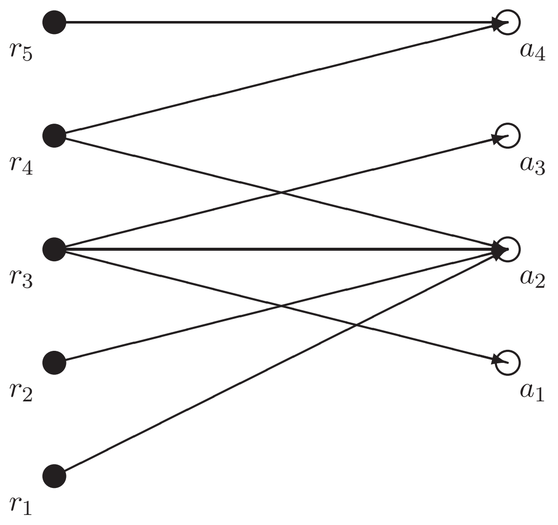


Figure 6.4: The Subsume graph

> The *subsume* is a matching level which is considered only in the occasion of complete matching. Suppose V_a is the set of requirements of advertisement `A` and V_r is the set of requirements of request `R`, then matching level is *subsume* in the context of either $|V_r| > |V_a|$ or $|V_r| < |V_a|$, if $\sum d_r > \sum d_a$. Here, d is the degree of a vertex belongs to the minimum cover of $G = (V_r, V_a, E)$ bipartite graph.

For example, The figure 6.4 shows subsume where $C = \{a_2, r_3, a_4\}$ and $M = \{r_1a_2, r_3a_3, r_4a_4\}$. In this example $\sum_{i \in V_r} d_i = 7 > \sum_{i \in V_a} d_i = 6$. According to the result, `A` $\sqsubseteq$ `R`, simply the advertisement `A` is subsumed by request `R`. Therefore, `A` and `R` are in subsume matching.

Obviously, G has a perfect matching if and only if $|V = 2v(G)|$, so Exact (perfect) matching explain in the section 6.5.1 is especial case of the maximum matching. In this case, $v(G)$ denote the size of the maximum matching.

Let $G = (V, E)$ be a bipartite graph with bipartition (V_r, V_a), and let $(z_e : e \in E)$ be algebraically independent commuting indeterminate. Then define the matrix X which is V_r by V_a, such that $X_{i,j} = z_e$ if $ij = e \in E$ and $X_{ij} = 0$ otherwise. We call X the *bipartite-matching matrix* of G.

$$X = \begin{pmatrix} z_{a_1r_1} & z_{a_1r_2} & \cdots \\ z_{a_2r_1} & z_{a_2r_2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

For complete matching,

$$z_{a_ir_1} \vee z_{a_ir_2} \vee \ldots \vee a_ir_n \neq 0$$

or

$$z_{a_1r_j} \vee z_{a_2r_j} \vee \ldots \vee a_mr_j \neq 0$$

Here, $i = 1, 2, 3, \ldots m$ and $j = 1, 2, 3, \ldots n$.

As explained above perfect matching is an especial case. The Exact matching($A \equiv A$) needs to considered when $m = n$ otherwise needs to consider the plugin or the subsume matching.

What will happen if $z_{a_ir_1} \vee z_{a_ir_2} \vee \ldots \vee a_ir_n = 0$ or $z_{a_1r_j} \vee z_{a_2r_j} \vee \ldots \vee a_mr_j = 0$? We can define *disjoint* matching as follows

> Let $G = (V, E)$ be a bipartite graph with bipartition (V_r, V_a), and let $(z_e : e \in E)$ be algebraically independent commuting indeterminate. Then define the matrix X which is V_r by V_a, such that $X_{i,j} = z_e$ if $ij = e \in E$ and $X_{ij} = 0$ otherwise. For all m, if $z_{a_ir_1} \vee z_{a_ir_2} \vee \ldots \vee a_ir_n = 0$ and for all n, if $z_{a_1r_j} \vee z_{a_2r_j} \vee \ldots \vee a_mr_j = 0$, then nothing match. Here, $i = 1, 2, 3, \ldots m$ and $j = 1, 2, 3, \ldots n$. Therefore $|E| = 0$ and this matching is defined as *disjoint*.

We can define *intersection* matching as follows

> Let $G = (V, E)$ be a bipartite graph with bipartition (V_r, V_a), and let $(z_e : e \in E)$ be algebraically independent commuting indeterminate. Then define the matrix X which is V_r by V_a, such that $X_{i,j} = z_e$ if $ij = e \in E$ and $X_{ij} = 0$ otherwise. For some m, if $z_{a_ir_1} \vee z_{a_ir_2} \vee \ldots \vee a_ir_n \neq 0$ and for some n, if $z_{a_1r_j} \vee z_{a_2r_j} \vee \ldots \vee a_mr_j \neq 0$, then there are some matching of request R to advertisement A. Here, $i = 1, 2, 3, \ldots m$ and $j = 1, 2, 3, \ldots n$. This matching is defined as *intersection* because $|E| > 0$.

In the graph $G = (V, E)$ explained above, edge $e \in E$ incident two nodes if and only if two edges are matching each other. For example, as shown in the figure 6.4, a_4 degree is $d = 2$ because two edges are incident on the node a_4 which are going out from the nodes r_4 and r_5. Next section we are going to look at the basic question how to match two nodes based on request and advertisement requirements.

6.5.3 Matching

Let us consider the more atomic level matching which will be the based for service matching explained in the above section 6.5. Matching of concepts against patterns is well discussed in the [8]. This section summarise the semantic matching concept. Semantic matching of concepts against patterns is a inference task

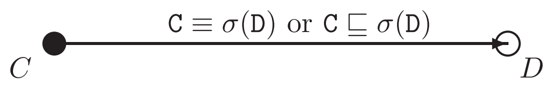

$\mathtt{C} \equiv \sigma(\mathtt{D})$ or $\mathtt{C} \sqsubseteq \sigma(\mathtt{D})$
C
D

Figure 6.5: The Matching graph

in DL. Mostly, semantic matching uses as a tool for detecting redundancies in knowledge bases and to support in integration of knowledge bases. Our objective is to extend the same concept to use in the area of web services personalisation.

Concept descriptions containing variables. For example, assume that we want to find concepts that are concerned with individuals having some common characteristics. This can be expressed by a pattern

$$\mathtt{D} := \exists\mathtt{available}.(\mathtt{X} \sqcap \mathtt{Desktop}) \sqcap \exists\mathtt{available}.(\mathtt{X} \sqcap \mathtt{Laptop}).$$

where X is a variable standing for the common characteristic. The concept description,

$$\mathtt{C} := \exists\mathtt{available}.(\mathtt{Location} \sqcap \mathtt{Desktop}) \sqcap \exists\mathtt{available}.(\mathtt{Location} \sqcap \mathtt{Laptop})$$

if we replace the variable `X` by the description `Location` the pattern become equivalent to the description. This is called *matcher modulo equivalence* σ of the matching problem $\mathtt{C} \equiv^? \mathtt{D}$.

$$\sigma := \{\mathtt{X} \mapsto \mathtt{Location}\}$$

In this example, matcher itself tell us what is the common characteristic of the `Desktop` and `Laptop` availability.

if σ is matcher, then we can define this concept as follows

> Given a concept pattern `D` and a concept description `C`, a *matching modulo equivalence*, $\mathtt{C} \equiv^? \mathtt{D}$, asks for a substitution σ such that $\mathtt{C} \equiv \sigma(\mathtt{D})$.

Such an exact match is not the only way of matching. The concept *matching modulo subsumption* is analogous to the concept *matching modulo equivalence*.

> Given a concept pattern $\mathtt{D}$ and a concept description $\mathtt{C}$, a *matching modulo subsumption*, $\mathtt{C} \sqsubseteq^? \mathtt{D}$, asks for a substitution σ such that $\mathtt{C} \sqsubseteq \sigma(\mathtt{D})$.

Note that since σ is a matcher for $\mathtt{C} \sqsubseteq^? \mathtt{D}$ iff it is one of $\mathtt{C} \equiv^? \mathtt{C} \sqcap \mathtt{D}$, matching modulo subsumption is a special case of matching modulo equivalence.

Purposefully we have avoided further explanation of semantic matching due to the availability of comprehensive enough details in the classic article [8] which is about how to matching in language $\mathcal{ALC}$ and Existential restriction.

In this research, *matching modulo equivalence* and *matching modulo subsumption* are used to define matching between two vertices in bipartite graph as shown in the figure 6.5. This results are directly applicable to implement the function $f(x_{ij})$ of the equation 6.2 for Algo. 6.3 discussed under the section 6.4.5.

Chapter 7

Conclusion

In this chapter, we enumerate the work of this research. In addition to that limitations and open issues of ontology based web service personalisation and semantic base service matching and composition has been outlined.

7.1 This work

Throughout this research, the importance of the ontology has been emphasised. Our approach is more formal look at to the web services discovery, selection and composition. This work went on the following order;

1. First, we stared with the literacy survey to find the web services composition. We have found plenty of research resources on this topic. But as we see, remaining problem was that how to select the best fit service for composition from the set of best fit services not addressed. By the further investigation we have found that the most possible solution was personalisation. Then the survey had been focused to find the personalisation solution for web services composition in the chapter 6.

   Semantic approach has been selected as the final solution for personalisation. Based on the semantic solution, we have proposed semantic based match maker with the relevant algorithms [42].

2. Second step was to introduce the personalisation ontology which is in chapter 5. Personalisation ontology [41] was central to semantic web services discovery and selection has been explained in the chapter 6.

3. In the second step, the main problem was ontology validation. Therefore, we have introduced formal mechanism to validate any ontology [43] which was explained in the chapter 4.

This research's primary intention to solve the problem of the web services matching and selection based on the semantics and user preferences. There are number of possible technical implementations that can be proposed as we have only defined a logical framework. However, we have used minimum set of standard technologies in this thesis which is unavoidable. For example, we have intentionally avoided easy technical solutions such as WSDL-S [58] because that is a proprietary IBM solution.

7.2 Future Work

During the exploration of semantic web services we have found we several issues to developed in the future.

7.2.1 Personalisation Ontology

In this thesis we have given a attention to the OWL 1.1 version which is not capable to represent full DL expressiveness. Therefore, the hypothetical level we have reach with DL will not yet be implementable in the current version of OWL at the time of writing this thesis. Therefore, the personalisation ontology (see chapter 5) has been defined to express in the OWL 1.1 version. But in the future versions of OWL probably may have more close to the complete DL expressiveness which will case to change the personalisation ontology in a way, which is more expressive.

The main constraint of the current architecture is that both consumer and the provider should conform to the same personalisation ontology. But in this environment, both the consumer and the provider don't know each other. If there is multiple personalise ontologies, match maker will be invalidated because request can be evaluated against only the advertisements both conform to same personalise ontology, rest of the other advertisements are ignored although they provide right services.

For both service matching and personalisation, we have used semantic approach. But there are other conventional techniques for personalisation which are deviated from the ontology. For example, history analysis, fuzzy logic [14], or Bayesian decision theory [21].

7.2.2 Matchmaker

Matchmaking algorithm significantly affect to the discovery quality and overall registry performance. Higher the discovery quality drastically reduce the overall registry performance. Both the service request and advertisement are specified in service description languages with certain expressiveness. Such a expressiveness highly affects the quality of the discovery procedure. As shown in the Fig 6.1, currently personalisation agent resides in the client side. The advantage of moving agent to service registry is beneficial.

1. Centralised personalisation agent is easy to modify without influencing to the clients. This will reduce deployment cost also.

2. Due to the personalisation agent at client side make the client application is thick. Client will become thin and creating a browser client also possible if the personalisation agent is centralised.

3. Current implementation of the personalisation agent proposed based on the MAS due the limitations. But centralise client can be implemented as another web service if it is possible to access ontology via web services as explained in the [18].

4. In the current implementation of matchmaker spread both server as well as the client because personalised agent available at client side. Centralised personalise agent will avoid the distribution of matchmaker responsibilities and limited to server.

There are some advantage of using the current architecture of the personalise agent because client personalise agent is best if the matchmaker provide an API to client. Client can create the his/her own implementation which is bridge to access more localised user profile and transfer data to the Matchmaker via its API.

In this thesis, we consider only *centralised discovery architecture* having used UDDI as server, but we can consider how to use this in gird or Peer-to-Peer (P2P) environment. In a P2P SOC, each peer is a service provider as well as service consumer. Then each peer should contains advertisements of offered services and link to the personalisation ontology. Matchmaker should resides in each peer. The important point is that service requester query all the known peers to and collect advertisements locally. In this approach Matchmaker is efficient due to the local cache.

External matching is better than extending UDDI as Semantic UDDI. Matchmaker can be other web service as explained above. There can be number of matchmakers available from different vendors. In this approach, UDDI remains as conventional yellow pages and service requester is liberalise to get the service of any matchmaker he/she wish. However, there is a remaining question that is; how to find right matchmaker without human introversion? However, this architecture eliminate the need of installing the matchmaker in either on the server side (registry), or client side.

Complexity is an unavoidable factor of algorithms. In this thesis, web services discovery, selection and even composition is based on algorithmic techniques. To

```
<?xml version='1.0' encoding= ...
<!DOCTYPE uridef[
  <!ENTITY
<!ENTITY concepts "http://...Concepts.owl">
... ... ...
<profileHierarchy:BookSelling
... ... ...
    <profile:serviceParameter>
      <profile:ServiceParameter>
<profile:serviceParameterName
rdf:datatype="&xsd;#string">
SomeRating
</profile:serviceParameterName>
<profile:sParameter
rdf:resource="&concepts;# Good"/>
      </profile:ServiceParameter>
    </profile:serviceParameter>
... ... ...
</profileHierarchy:BookSelling>
```

Figure 7.1: Quality Rating of Book Selling profile

```
<owl:Class rdf:ID="GoodRating">
</owl:Class>

<owl:Class rdf:ID="QualityRating">
    <owl:oneOf rdf:parseType="Collcection">
        <QualityRating rdf:ID="Excellent"/>
        <QualityRating rdf:ID="Good"/>
... ... ...
</owl:Class>
```

Figure 7.2: Personalisation Ontology

avoid this complicity the best approach is to find the way which simply exploit IOPE parameters in some elaborate way.

Use the list of additional parameters available in the OWL-S *service profile* for ontology is another alternative. In user profile there can be *n* number of preferences, example consider *Quality Rating* of service rating may be excellent, good, average and poor. Before using a service, a requester may check with what kid of service that is going to deal with. Service provider's responsibility is to publish its ratings. The service profile publishes the services as shown in the Fig. 7.1.

As shown in the Fig. 7.1, Book selling service publishes the quality rating for a service (in this example value of QualityRating is "GoodRating"). Book selling service ontology refers to the Concept (available at http://.../Concept.owl)ontology for the QualityRating as shown in the Fig.7.2. Therefore user profile ontology also should refer to the same ontology concept to set the value for QualityRating.

Note that only constraint is both the service profile and the user profile must refer to a same ontology such as Concept ontology. Mediation layer can be removed.

Appendices

Appendix A

Abbreviations

Abbreviation	Meaning
.NET	Microsoft Enterprise Framework
AI	Artificial Intelligence
B2B	Business to Business
DL	Description Logic
DoM	Degree of Match
FOL	First Order Logic
HTML	Hipper Text Markup Language
HTTP	Hipper Text Transfer Protocol
IOPE	Inputs, Outputs, Preconditions, Effects
J2EE	Java 2 Enterprise Edition
KD	Knowledge Discovery
P2P	Peer-to-Peer
MAS	Multi-Agent System
RDF	Resource Description Framework
OWL	Ontology Web Language
OWL-S	Semantic Markup for Web Services
QoS	Quality of Service
REST	
SGML	Standard Generalized Markup Language
SOAP	Simple Object Access Protocol
SOC	Service Oriented Computing
SWS	Semantic Web Service
TCP	
UDDI	Universal Description, Discovery and Integration

Abbreviation	Meaning
UDP	
UMFA	User centred, Mobile agent based, Fault-injection equipped and Assertion oriented
URI	Unified Resource identifier
W3C	World Wide Web consortium
WS	Web Service
WSAF	Web Service Agent Framework
WSC	Web Service Composition
WSDL	Web Service Description Language
XML	eXtended Markup Language

Appendix B

Preliminaries

B.1 Semantic Web

This section aims to provide the reader with an overview of the current state of the art in Semantic Web technologies. Searching the information urge but causes to infuriating experience of the current form of the web. The idea of semantic web [67] is machine understandable web resources. The need of sharing and understanding information among different agents need a standardised markup language. As a next step, need to make sure that different agents have common understanding of these terms; *Ontology* in which these terms are described. The web ontology language (OWL) provides mechanisms for creating all the components of an ontology from concepts, instances, properties (or role) and axioms. Two type of properties can be defined which are *object properties* and *datatype properties*. Hence, *object property* relate instance to instance while *datatype property* relate instance to data type values. Concepts can have super-concept as well as sub-concepts, thus providing a mechanism for subsumption reasoning and inheritance of properties. Axioms are used to provide information about concepts and properties, for example to specify the equivalence of two concepts (classes) or the range of a property (role). OWL builds on the Resource Description Framework (RDF) which is essentially a data modelling language defined by the W3C. Preliminary RDF is graph-based, but usually serialised as XML. Essentially, it consists of triples which are *subject*, *predicate*, *object*. The design of OWL was influenced by more than 10 years of Description Logic research.

B.2 Description Logic

Description Logic (DL) is the name for a family of a knowledge representation formalism that represent the knowledge of the domain by its terminologies and

descriptions. For knowledge representation systems , DL is the abstract language to define it and inference over it. Terminologies are concepts which specify properties of objects and individuals occurring in the domain is the *description.* Only one name is permitted for *definition* * and *definition* should be acyclic. These are the very necessary assumption about the terminologies. Placing new concept definition in a proper place of the taxonomic hierarchy is called *classification* which is the basic task of terminology construction .
As name of the DL implies, they are equipped with a formal, logical based semantics. One of the distinguished feature of DL is reasoning capabilities. DL languages are then viewed as the core of knowledge representation(KR) systems, considering both the structure of a DL knowledge base and reasoning services. Reasoning may also be used when ontology (see B.3 in page 81) deployed. Understanding of DL is best before arguing why DLs are good candidates for such an ontology language.

DLs are descended from so-called *structured inheritance networks* which were introduce to overcome the ambiguities of early semantic networks and frames [27].

- The basic syntactic building blocks are *atomic concepts* (unary predicates), *atomic roles* (binary predicate), and individuals(constants) .

- The expressive power of the language is restricted in that it uses a rather small set of constructors for building complex concepts and roles.

- Implicit knowledge about concepts and individuals can be inferred automatically with the help of inference procedures.

The KR system based on DL provides facilities to set up knowledge base, to reason about their content, and to manipulate them. A knowledge base (KB) comprises two components which are the TBox and the ABox , respectively represent *intentional knowledge* and *extensional knowledge*. The TBox for terminology which is considered to be the vocabulary of an application domain and TBox contains all the *definitions* which are not time bound. The ABox contains *assertions* about named individuals in terms of this vocabulary. In ABox *membership assertion* is in two flavours as *concept assertion* and *role assertion* respectively for concepts and roles. Important to remember that in concept assertion the *concept expressions* are generally allowed not like *role expressions* which are only allowed in very expressive languages. TBox reasoning are *subsumption* and *satisfiability* while *instance checking* is the basic reasoning of ABox. The DL language has *model theoretic semantics* , therefore statements in the TBox and in the ABox can be identified with formula in first-order logic [27]. In addition to terminol-

*The basic declaration of TBox is *definition.*

ogy and assertion, DL offer service to reason about them. Some of the typical reasoning tasks for a terminology listed below

- Determine whether a description is *satisfiable*.
- One description *subsume* other.
- Find out whether set of assertions is *consistent* in the ABox.
- Check whether the particular individual is an instance of a given concept description.

Satisfiability check of descriptions and consistency checks of sets of assertions are useful to determine whether a knowledge base is meaningful at all.

B.2.1 Description Languages

The expressive power of OWL is determined by the class constructors supported, and by the kinds of axioms that can occur in ontology. Elementary descriptions are *atomic concepts* and *atomic roles*. We can create complex descriptions using *concept constructors* shown in the B.1 table.

Syntax	Construct Name	language
$C, D \longrightarrow A$	atomic concept	
R	atomic role	
$\top$	universal concept	
$\bot$	bottom concept	
$\neg A$	atomic negation	$\mathcal{AL}$
$C \sqcap D$	intersection	
$\forall R.C$	value restriction	
$\exists R.\top$	limited existential quantification	
$C \sqcup D$	union	$\mathcal{U}$
$\exists R.C$	full existential quantification	$\mathcal{E}$
$\geq nR$	at-least number restriction	$\mathcal{N}$
$\leq nR$	at-most number restriction	
$\neg C$	negation	$\mathcal{C}$

Table B.1: The basic description languages.

In abstract notation A and B for atomic concepts and letter R for atomic role and letter C and D for concept descriptions. The Attribute Language ($\mathcal{AL}$) is the minimal description language. We can name each $\mathcal{AL}$-language by form of the string $\mathcal{AL}[\mathcal{U}][\mathcal{E}][\mathcal{N}][\mathcal{C}]$, where the letter name stands for the presence of the corresponding constructor. Not all these languages are distinct. Union and full existential quantification can be expressed using negation as $C \sqcup D \equiv \neg(\neg C \sqcap \neg D)$

Syntax	Construct Name	language
$\mathcal{ALC}$ constructs		
$R \in \mathbf{R}_+$	transitive role	$\mathcal{S}$
$R \sqsubseteq S$	role hierarchy	$\mathcal{H}$
R^-	inverse role	$\mathcal{I}$
$\geq nR.C$	at-least qualifying number restriction	$\mathcal{Q}$
$\leq nR.C$	at-most qualifying number restriction	

Table B.2: Syntax and semantics of the $\mathcal{SI}$ family of DLs.

and $\exists R.C \equiv \neg\forall R.\neg C$. Therefore we can use $\mathcal{ALC}$ instead of $\mathcal{ALUE}$ and $\mathcal{ALCN}$ instead of $\mathcal{ALUEN}$. The logics we will discuss are all based on an extension of the well known DL $\mathcal{ALC}$ to include transitively closed primitive roles. We will call this logic $\mathcal{S}$ [35] due to its relationship with the proposition model logic $\mathbf{S4}_{(m)}$. The basic DL can be extended in variety of ways as shown in the table B.2.

The role hierarchy is very important for OWL as well transitive property (role) also very important for semantic application development. Members of the $\mathcal{SI}$ family includes the influential $\mathcal{SHIQ}$ as shown in the table B.2. The language $\mathcal{SHOQ}(\mathbf{D})$ defines the language which adds the ability to define a class by enumerating its instances and support for data types and values. $\mathcal{SHOQ}(\mathbf{D})$ interpretations include additional interpretation domain for data values $\Delta^{\mathcal{I}}_{\mathbf{D}}$ which is disjoint from the domain of individuals ($\Delta^{\mathcal{I}}$). Data type properties such as `age` are interpreted as binary relation between $\Delta^{\mathcal{I}}$ and $\Delta^{\mathcal{I}}_{\mathbf{D}}$ (i.e., subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}_{\mathbf{D}}$).

B.2.2 Terminological axioms

The *terminological axioms* makes the statements about how concepts or roles are related to each other. Let's use `C` and `D` to denote two concepts and `R` and `S` for atomic roles. In the most general case, *terminological axioms* have the from

$$\mathtt{C} \sqsubseteq \mathtt{D} \quad (\mathtt{R} \sqsubseteq \mathtt{S}) \tag{B.1}$$

This axiom is called *inclusion* which is basic inference on concept expressions in DLs is *subsumption*. The `D` *subsumer* is considered more general than the `C` *subsumee*. The *subsumption* checks whether the `C` concept denotes a subset of set denoted by `D` concept. Inclusion state the *necessary conditions* which is incomplete definition about the concept. This is not the same "IS-A"[*] relationship of monotonic inheritance hierarchy. DL is capable of to represent other kind of relationships that can hold between concepts, beyond IS-A relationships.

$$\mathtt{C} \equiv \mathtt{D} \quad (\mathtt{R} \equiv \mathtt{S}) \tag{B.2}$$

[*]Good example is Object-Oriented programing(OOP). In OOP the "IS-A" define hierarchy over the concepts.

This kind of axiom is called *equalities*. A *definition* is atomic concept which is left hand side of the *equality*. The important of the *definition* is it introduce new *symbolic names* for complex descriptions. A set of definitions should be unequivocal. The equality defines the *necessary and sufficient* condition which is the strongest axiom and it is the complete definition about the concept. If the definition is cyclic (for example $C \equiv A \sqcap R.C$), adapt *fixpoint* semantics to make them unequivocal.

Consider $\mathcal{T}$ is a *terminology*(TBox) in which atomic concepts can be divided into two sets, *name symbols* $\mathcal{N}_{\mathcal{T}}$ (defined concept) at left hand side of a axiom and the *base symbols* $\mathcal{B}_{\mathcal{T}}$ (primitive concept) at right hand side of the axiom. The terminology defines the *name symbols*($\mathcal{N}_{\mathcal{T}}$) in terms of the *base symbols*($\mathcal{B}_{\mathcal{T}}$). Let $\mathcal{J}$ is a *base interpretation* that interpret only base symbols while $\mathcal{I}$ is *name interpretation* that interprets name symbols is an extension of $\mathcal{J}$ only if $\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}$ ($\Delta^{\mathcal{I}}$ is domain of interpretation). Then $\mathcal{T}$ *definitorial* if every $\mathcal{J}$ has exactly one extension that is model of $\mathcal{T}$. In other words, if we know what $\mathcal{B}_{\mathcal{T}}$ stand for, and $\mathcal{T}$ is definitorial, then the meaning of the $\mathcal{N}_{\mathcal{T}}$ is completely determined. All definitorial $\mathcal{ALC}$ terminology is equivalent to an acyclic terminology. However, there are some situations where intuitively cyclic definitions are meaningful such as *transitive closure.* In the case of cyclic definitions descriptive semantics do not resolved the problem along. Fixedpoint semantics are motivated by this factor. The intuition can be captured by least or greatest fixedpoint semantics. For further examples refer to the section 3.4 in page 12.

B.2.3 Semantics

The key feature of DLs is that they are logic, i.e., formal languages with well defined semantics. The standard technique of *model theoretic semantic* (explicit the relationship between the language syntax and the intended models of the domain) specifying the meaning of a DL. For interpretation function $\mathcal{I}$ a model consist of a domain $\Delta^{\mathcal{I}}$ which is the set of objects and interpretation function is a mapping from individual, class and property names to elements of the domain, subsets of the domain and binary relations on the domain respectively. For example,individual i ,$i^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, for a class `Person`, here $\texttt{Person}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and for a property `parentOf`,hence, $\texttt{parentOf}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

Interpretation function $\mathcal{I}$ can be extended from class names to complex class descriptions. Objects themselves don't have any meaning nor does the choice of any particular set of objects that make up the domain. Instead important is relationship between objects and set of objects. For a given model i is an individual that is instance of class C. The i can be interpreted as an element of the interpretation of a class C (i.e, $i^{\mathcal{I}} \in C^{\mathcal{I}}$) and C is subclass of a class D just in case the interpretation of C is a subset of the interpretation of D (i.e, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$).

A DL knowledge base consist of a set of axioms as explained above. The meaning of these axioms is given by corresponding constraint on models. For example, knowledge base contain the axiom that stating `Person` $\sqsubseteq$ `Animal` (`Person` is subclass of `Animal`). Then in a model of the knowledge base the interpretation of concept `Person` must always be subset of the interpretation of concept `Animal`. Model considered to be *inconsistent* if there is no possible interpretation or always interpretation is empty set. If the relationship specified by the given axiom must hold in all interpretations of the knowledge base, then that axiom said to be entailed by the knowledge base, and if one knowledge base entails every axiom in another knowledge base then the first knowledge base entail the second knowledge base. The meaning of a knowledge base derives from features and relationships that are common to all possible models[34].

B.2.4 Open World Assertions

The ABox or $\mathcal{A}$ is the other component of the knowledge base. ABox is specification of affairs of application domain in terms of concepts and roles. Suppose individuals names are a, b, c and C and R are respectively concept and role. There are two types of assertion:

$$C(a) \tag{B.3}$$

$$R(b, c) \tag{B.4}$$

The equation B.3 is *concept assertion* while equation B.4 is *role assertion*. In equation B.4, c is a *filler* of the role R for b. The interpretation $\mathcal{I}$ *satisfies* the concept assertion if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and it satisfies role assertion if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. We assume that distinct individual names denote distinct objects. Throughout this thesis, *unique name assumption*(UNA) being used, that is, if a, b are distinct names, then $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.

B.2.5 Additional constructors

Sometime it is necessary to use *individual names* (also called *nominals*) in description language. The most basic constructor is "set" (or *one-of*) constructor which is written as

$\{a_1, ..., a_n\} = a_1 \sqcup ... \sqcup a_n$, which is for $\{a_1, ..., a_n\}^{\mathcal{I}} = \{a_1^{\mathcal{I}}, ..., a_n^{\mathcal{I}}\}$

The "fill" is another constructor which does not add anything new, since $R : a \equiv \exists R.\{a\}$. The operator *fill* allows one to express role assertions through concept assertions: an interpretation satisfies $R(a, b)$ iff it satisfies $(\exists R.b)(a)$. It is important to note that if these constructors are not available, *instance checking* is harder than *satisfiability* and *subsumption* problems.

B.3 Ontology for semantic web

The hart of the semantic web is Ontology. An ontology is an explicit and formal specification of a conceptualisation of a domain of interest. Here the key points are formal conceptualisation and inference capabilities. Ontology can be defined as

$<C, R, I, A>$ here

C	Concept
R	Role
I	Individual
A	Axiom

The use of ontologies require a well defined, well designed and existing web standards compatible markup language which is human intuitive and should support reasoning tools. Early work on defining ontologies languages has now converged under the aegis of the W3C, to produce a Web Ontology Language, OWL. Reasoning ensure the quality of the ontology, therefore expressive power of the language should not infeasible the reasoning capability. However, most important issue regarding ontology is interoperability and integration of ontologies. The construction of an ontology can be a time-consuming process, requiring the services of experts both in ontology engineering and the domain of interest. If the generation of ontologies is time-consuming, even more is this the case of metadata extraction. Ontologies need to change, as knowledge changes and as usage changes. The evolution of ontologies is nontrivial. A commonly misunderstand about the Semantic Web is that it depends on the creation of monolithic (not result of collaborative effort) ontologies which requiring agreement from many parties for use. Although It is good design practice to reuse existing ontologies wherever possible, particularly where an ontology enjoys wide support. However, in many cases we need to construct mappings between ontologies describing the same domain, or alternatively merge ontologies to form their union. Both approaches rely on the identification of correspondences between the ontologies, a process known as *ontology alignment* , and one where automatic (or semiautomatic) techniques are needed. Today need is for a distributed evolution of ontologies. Typically individual users may create their own variations on a basic core ontology, which then needs to be kept in step to reflect the best of the changes introduced by users.

B.3.1 Introduction to Knowledge Discovery

Knowledge Discovery (KD) is a research area which is intended to develop techniques that enable computers to discover new and valuable information from raw data. As usually the initial output from KD is further refined via an iterative process with a human in the loop in order to get knowledge out of the raw data.

So far researchers develop methods which are semi-automatic processing of complex data. These methods are becoming possible to extract hidden and useful pieces of knowledge which can be further used for different purpose including semi-automatic ontology construction. As ontologies are taking a significant role in the Semantic Web, the problem of semi-automatic ontology construction supported by KD gaining momentum. We have discussed different approaches of KD in section 3.2 in page 11.

B.3.2 DL as ontology language

High quality ontologies are crucial for the semantic web. DLs are the ideal candidate for semantic web because DL provides well defined semantics and powerful reasoning tools [25].

Appendix C

Ontologies

C.1 Family Ontology

The family ontology defined as follows in the DL,

$$
\begin{aligned}
&\texttt{Woman} \equiv \texttt{Person} \sqcap \texttt{Female}\\
&\texttt{Man} \equiv \texttt{Person} \sqcap \neg\texttt{Woman}\\
&\texttt{Mother} \equiv \texttt{Woman} \sqcap \exists\texttt{hasChild.Person}\\
&\texttt{Father} \equiv \texttt{Man} \sqcap \exists\texttt{hasChild.Person}\\
&\texttt{Parent} \equiv \texttt{Father} \sqcup \texttt{Mother}\\
&\texttt{GrandMother} \equiv \texttt{Woman} \sqcap \exists\texttt{hasChild.Parent}\\
&\texttt{MotherWithManyChildren} \equiv \sqcap \geq \texttt{3hasChild}\\
&\texttt{MotherWithoutDaughter} \equiv \texttt{Mother} \sqcap \forall\texttt{hasChild.}\neg\texttt{Woman}\\
&\texttt{Wife} \equiv \texttt{Woman} \sqcap \exists\texttt{hasHusband.Man}
\end{aligned}
$$

Let us see the implementation of the family ontology in OWL,

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/family.owl#"
  xml:base="http://www.owl-ontologies.com/family.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Mother">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:ID="hasChild"/>
            </owl:onProperty>
            <owl:someValuesFrom>
              <owl:Class rdf:ID="Person"/>
            </owl:someValuesFrom>
          </owl:Restriction>
          <owl:Class rdf:ID="Woman"/>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="Man">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class>
            <owl:complementOf>
              <owl:Class rdf:about="#Woman"/>
            </owl:complementOf>
          </owl:Class>
          <owl:Class rdf:about="#Person"/>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="Father">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:onProperty rdf:resource="#hasChild"/>
            <owl:someValuesFrom rdf:resource="#Person"/>
          </owl:Restriction>
          <owl:Class rdf:about="#Man"/>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="MotherWithManyChildren">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:onProperty rdf:resource="#hasChild"/>
            <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
            >3</owl:minCardinality>
          </owl:Restriction>
          <owl:Class rdf:about="#Mother"/>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="Parent">
    <owl:equivalentClass>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Father"/>
          <owl:Class rdf:about="#Mother"/>
        </owl:unionOf>
      </owl:Class>
    </owl:equivalentClass>
```

```
</owl:Class>
<owl:Class rdf:ID="Grandmother">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty rdf:resource="#hasChild"/>
          <owl:someValuesFrom rdf:resource="#Parent"/>
        </owl:Restriction>
        <owl:Class rdf:about="#Mother"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:about="#Woman">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:ID="Female"/>
        <owl:Class rdf:about="#Person"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Wife">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasHusband"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#Man"/>
    </owl:Restriction>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="#Woman"/>
</owl:Class>
<owl:Class rdf:ID="MotherWithoutDaughter">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty rdf:resource="#hasChild"/>
          <owl:allValuesFrom>
            <owl:Class>
              <owl:complementOf rdf:resource="#Woman"/>
            </owl:Class>
          </owl:allValuesFrom>
        </owl:Restriction>
        <owl:Class rdf:about="#Mother"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:ObjectProperty rdf:about="#hasHusband">
  <rdfs:range rdf:resource="#Man"/>
</owl:ObjectProperty>
<Person rdf:ID="HARRY"/>
<Father rdf:ID="PETER">
  <hasChild rdf:resource="#HARRY"/>
</Father>
```

```
  <Mother rdf:ID="MARRY">
    <hasChild rdf:resource="#PETER"/>
  </Mother>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.2, Build 355)  http://protege.stanford.edu -->
```

C.2 Common Ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY common.owl "file:/C:/Personalisation/common.owl">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&common.owl;"
         xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;">

<!-- Ontology Information -->
  <owl:Ontology rdf:about=""
                rdfs:label="Common Ontology"
                owl:versionInfo="1.0"/>

<!-- Annotation Properties -->
  <owl:AnnotationProperty rdf:about="&rdfs;label"/>
  <owl:AnnotationProperty rdf:about="&owl;versionInfo"/>

<!-- Object Properties -->
  <owl:ObjectProperty rdf:about="#hasPart"/>
  <owl:ObjectProperty rdf:about="#partOf">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <owl:inverseOf rdf:resource="#hasPart"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

C.3 Temporal Ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY calendar.owl "file:/C:/Personalisation/calendar.owl">
  <!ENTITY common.owl "file:/C:/Personalisation/common.owl">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY temporal.owl "file:/C:/Personalisation/temporal.owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&temporal.owl;"
         xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;">

<!-- Ontology Information -->
  <owl:Ontology rdf:about=""
                rdfs:label="Temporal Ontology"
                owl:versionInfo="1.0">
    <owl:imports>
      <owl:Ontology rdf:about="&calendar.owl;"/>
    </owl:imports>
    <owl:imports>
      <owl:Ontology rdf:about="&common.owl;"/>
    </owl:imports>
  </owl:Ontology>

<!-- Classes -->
  <owl:Class rdf:about="#ClassExpression878935533"/>
  <owl:Class rdf:about="#DateTime">
    <owl:intersectionOf rdf:parseType="Collection">
      <rdf:Description rdf:about="#Time"/>
      <rdf:Description rdf:about="&calendar.owl;#Date"/>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="#Duration">
    <rdfs:subClassOf rdf:resource="#Interval"/>
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:onProperty rdf:resource="&common.owl;#hasPart"/>
        <owl:someValuesFrom rdf:resource="#Second"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&common.owl;#hasPart"/>
        <owl:someValuesFrom rdf:resource="#Minute"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&common.owl;#hasPart"/>
        <owl:someValuesFrom rdf:resource="#Hour"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="#DurationToTime">
    <owl:intersectionOf rdf:parseType="Collection">
      <rdf:Description rdf:about="#Duration"/>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        <owl:onProperty rdf:resource="#ends"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="#Hour">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">23</owl:maxCardinality>
        <owl:onProperty rdf:resource="#unitTime"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&common.owl;#hasPart"/>
        <owl:someValuesFrom rdf:resource="#Minute"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">0</owl:minCardinality>
        <owl:onProperty rdf:resource="#unitTime"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="#Instance">
    <rdfs:subClassOf rdf:resource="#TemporalEntity"/>
    <owl:disjointWith rdf:resource="#ProperInterval"/>
  </owl:Class>

  <owl:Class rdf:about="#Interval">
    <rdfs:subClassOf rdf:resource="#TemporalEntity"/>
  </owl:Class>

  <owl:Class rdf:about="#Minute">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">0</owl:minCardinality>
        <owl:onProperty rdf:resource="#unitTime"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:allValuesFrom rdf:resource="#Hour"/>
        <owl:onProperty rdf:resource="&common.owl;#partOf"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">59</owl:maxCardinality>
        <owl:onProperty rdf:resource="#unitTime"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="#ProperInterval">
    <rdfs:subClassOf rdf:resource="#Interval"/>
    <owl:disjointWith rdf:resource="#Instance"/>
  </owl:Class>

  <owl:Class rdf:about="#Second">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">0</owl:minCardinality>
```

```
        <owl:onProperty rdf:resource="#unitTime"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">59</owl:maxCardinality>
        <owl:onProperty rdf:resource="#unitTime"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:allValuesFrom rdf:resource="#Minute"/>
        <owl:onProperty rdf:resource="&common.owl;#partOf"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="#TemporalEntity"/>
  <owl:Class rdf:about="#Time">
    <rdfs:subClassOf rdf:resource="#Instance"/>
    <owl:intersectionOf rdf:parseType="Collection">
      <rdf:Description rdf:about="#Hour"/>
      <rdf:Description rdf:about="#Minute"/>
      <rdf:Description rdf:about="#Second"/>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="#TimeForDuration">
    <owl:intersectionOf rdf:parseType="Collection">
      <rdf:Description rdf:about="#Duration"/>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        <owl:onProperty rdf:resource="#begins"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="#TimeToTime">
    <rdfs:subClassOf rdf:resource="#Interval"/>
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        <owl:onProperty rdf:resource="#ends"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        <owl:onProperty rdf:resource="#begins"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="&calendar.owl;#Date"/>
  <owl:Class rdf:about="&owl;Nothing"/>

<!-- Annotation Properties -->
  <owl:AnnotationProperty rdf:about="&rdfs;label"/>
  <owl:AnnotationProperty rdf:about="&owl;versionInfo"/>

<!-- Datatype Properties -->
  <owl:DatatypeProperty rdf:about="#unitTime"/>

<!-- Object Properties -->
  <owl:FunctionalProperty rdf:about="#begins">
```

```
    <rdf:type rdf:resource="&owl;ObjectProperty"/>
    <rdfs:domain rdf:resource="#TemporalEntity"/>
    <rdfs:range rdf:resource="#DateTime"/>
    <owl:equivalentProperty rdf:resource="#ends"/>
  </owl:FunctionalProperty>

  <owl:ObjectProperty rdf:about="#ends">
    <owl:equivalentProperty rdf:resource="#begins"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&common.owl;#hasPart"/>
  <owl:ObjectProperty rdf:about="&common.owl;#partOf">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <owl:inverseOf rdf:resource="&common.owl;#hasPart"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

C.4 Calendar Ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY calendar.owl "file:/C:/Personalisation/calendar.owl">
  <!ENTITY common.owl "file:/C:/Personalisation/common.owl">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&calendar.owl;"
         xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;">

<!-- Ontology Information -->
  <owl:Ontology rdf:about=""
                rdfs:label="Calendar Ontology"
                owl:versionInfo="1.0">
    <owl:imports>
      <owl:Ontology rdf:about="&common.owl;"/>
    </owl:imports>
  </owl:Ontology>

<!-- Classes -->
  <owl:Class rdf:about="#CalendarDate">
    <owl:intersectionOf rdf:parseType="Collection">
      <rdf:Description rdf:about="#Day"/>
      <rdf:Description rdf:about="#Month"/>
      <rdf:Description rdf:about="#Year"/>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="#Date">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:onProperty rdf:resource="&common.owl;#hasPart"/>
        <owl:someValuesFrom rdf:resource="#Month"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&common.owl;#hasPart"/>
        <owl:someValuesFrom rdf:resource="#Day"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&common.owl;#hasPart"/>
        <owl:someValuesFrom rdf:resource="#Year"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="#Day">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">31</owl:maxCardinality>
        <owl:onProperty rdf:resource="#unitDay"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:allValuesFrom rdf:resource="#Month"/>
        <owl:onProperty rdf:resource="&common.owl;#partOf"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
        <owl:onProperty rdf:resource="#unitDay"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="#Month">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">12</owl:maxCardinality>
        <owl:onProperty rdf:resource="#unitDay"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:allValuesFrom rdf:resource="#Year"/>
        <owl:onProperty rdf:resource="&common.owl;#partOf"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
        <owl:onProperty rdf:resource="#unitDay"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="#OrdinalDate">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:allValuesFrom rdf:resource="#Year"/>
        <owl:onProperty rdf:resource="&common.owl;#partOf"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
        <owl:onProperty rdf:resource="#unitDay"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">366</owl:maxCardinality>
        <owl:onProperty rdf:resource="#unitDay"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="#WeekOfYear">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:allValuesFrom rdf:resource="#Year"/>
        <owl:onProperty rdf:resource="&common.owl;#partOf"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">52</owl:maxCardinality>
        <owl:onProperty rdf:resource="#unitDay"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
        <owl:onProperty rdf:resource="#unitDay"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>
```

```
  <owl:Class rdf:about="#Year">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:onProperty rdf:resource="&common.owl;#hasPart"/>
        <owl:someValuesFrom rdf:resource="#Month"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&common.owl;#hasPart"/>
        <owl:someValuesFrom rdf:resource="#Day"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="&owl;Nothing"/>

<!-- Annotation Properties -->
  <owl:AnnotationProperty rdf:about="&rdfs;label"/>
  <owl:AnnotationProperty rdf:about="&owl;versionInfo"/>

<!-- Datatype Properties -->
  <owl:DatatypeProperty rdf:about="#unitDay"/>

<!-- Object Properties -->
  <owl:ObjectProperty rdf:about="&common.owl;#hasPart"/>
  <owl:ObjectProperty rdf:about="&common.owl;#partOf"/>
</rdf:RDF>
```

C.5 Locative Ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY common.owl "file:/C:/Personalisation/common.owl">
  <!ENTITY locative.owl "file:/C:/Personalisation/locative.owl">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY temporal.owl "file:/C:/Personalisation/temporal.owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&locative.owl;"
         xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;">

<!-- Ontology Information -->
  <owl:Ontology rdf:about=""
                rdfs:comment="Locative Ontology"
                owl:versionInfo="1.0">
    <owl:imports>
      <owl:Ontology rdf:about="&common.owl;"/>
    </owl:imports>
    <owl:imports>
      <owl:Ontology rdf:about="&temporal.owl;"/>
    </owl:imports>
  </owl:Ontology>

<!-- Classes -->
  <owl:Class rdf:about="#Area">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">3</owl:minCardinality>
        <owl:onProperty rdf:resource="#hasPoint"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="#Degree">
    <owl:equivalentClass>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">60</owl:minCardinality>
        <owl:onProperty rdf:resource="#hasMinute"/>
      </owl:Restriction>
    </owl:equivalentClass>
  </owl:Class>

  <owl:Class rdf:about="#Latitude">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
        <owl:onProperty rdf:resource="#unitDegree"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">180</owl:maxCardinality>
        <owl:onProperty rdf:resource="#unitDegree"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="#Longitude">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">360</owl:maxCardinality>
        <owl:onProperty rdf:resource="#unitDegree"/>
      </owl:Restriction>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
        <owl:onProperty rdf:resource="#unitDegree"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="#Point">
    <owl:intersectionOf rdf:parseType="Collection">
      <rdf:Description rdf:about="#Latitude"/>
      <rdf:Description rdf:about="#Longitude"/>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:about="&temporal.owl;#Minute"/>

<!-- Annotation Properties -->
  <owl:AnnotationProperty rdf:about="&rdfs;comment"/>
  <owl:AnnotationProperty rdf:about="&owl;versionInfo"/>

<!-- Object Properties -->
  <owl:ObjectProperty rdf:about="#hasMinute">
    <rdfs:domain rdf:resource="#Degree"/>
    <rdfs:range rdf:resource="&temporal.owl;#Minute"/>
    <rdfs:subPropertyOf rdf:resource="&common.owl;#hasPart"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasPoint">
    <rdfs:range rdf:resource="#Point"/>
    <rdfs:subPropertyOf rdf:resource="&common.owl;#hasPart"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#unitDegree">
    <rdfs:range rdf:resource="#Degree"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&common.owl;#hasPart"/>
</rdf:RDF>
```

Appendix D

Sequent Calculus for DL

The sequent is an expression of the form $\Gamma \vdash \Delta$, where Γ(antecedent) and Δ(succedent) are first-order formulae. The symbol $\vdash$ is called ”entailment”. Therefore ”Γ entails Δ means that from all formula in Γ some of the formulae in Δ follows. Formally, the notion of (classical) satisfaction for the sequent $\Gamma \vdash \Delta$ isdefined as the satisfaction in First-Order Logic of the formula $\Gamma_{\wedge} \rightarrow \Delta_{\vee}$; it is called falsifiable otherwise.. A rule is a relation among sequents $S_1, ..., S_n$ and S

$$\rho\frac{S_1, ..., S_n}{S},$$

where ρ is the name of the rule, $S_1, ..., S_n$ are its premises and S is its conclusion. If n=0 the rule has no premises and we call it an axiom. A system in the sequent calculus is a finite set of rules, that we can participation as follows: the axiom, the set of left rules and the set right rules. Rules come in pairs for every logical connective, giving meaning to it depending on whether the connective appears at the left or at the right of the entailment symbol. A derivation for a formula is obtained by successively applying instances of the given rules; if all branches of the derivation tree reach axioms then the derivation is a proof.

Operator	Left Rule	Right Rule
$Weakening$	$\frac{\Gamma \vdash \Delta}{\alpha, \Gamma \vdash \Delta}$ W_l	$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, \alpha}$ W_r
$Contraction$	$\frac{\Gamma, \alpha, \alpha \vdash \Delta}{\Gamma, \alpha \vdash \Delta}$ C_l	$\frac{\Gamma \vdash \alpha, \alpha, \Delta}{\Gamma \vdash \alpha, \Delta}$ C_r
$Cut - rule$	$\Gamma, \alpha \vdash \Delta, \alpha$ A_l	$\frac{\Gamma_1, \alpha \vdash \Delta_1; \Gamma_2 \vdash \Delta_2, \alpha}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}$ A_r
$\vee$	$\frac{\Gamma, \alpha \vdash \Delta; \Gamma, \beta \vdash \Delta}{\Gamma, \alpha \vee \beta \vdash \Delta}$ $\vee_l$	$\frac{\Gamma \vdash \Delta, \alpha, \beta}{\Gamma \vdash \Delta, \alpha \vee \beta}$ $\vee_r$
$\wedge$	$\frac{\Gamma, \alpha, \beta \vdash \Delta}{\Gamma, \alpha \wedge \beta \vdash \Delta}$ $\wedge_l$	$\frac{\Gamma \vdash \Delta, \alpha; \Gamma \vdash \Delta, \beta}{\Gamma \vdash \Delta, \alpha \wedge \beta}$ $\wedge_r$
$\rightarrow$	$\frac{\Gamma, \beta \vdash \Delta; \Gamma \vdash \alpha, \Delta}{\Gamma, \alpha \rightarrow \beta \vdash \Delta}$ $\rightarrow_l$	$\frac{\Gamma, \alpha \vdash \Delta, \beta}{\Gamma \vdash \Delta, \alpha \rightarrow \beta}$ $\rightarrow_r$
$\neg$	$\frac{\Gamma, \alpha \vdash \Delta}{\Gamma \vdash \Delta, \neg \alpha}$ $\neg_l$	$\frac{\Gamma \vdash \Delta, \alpha}{\Gamma, \neg \alpha \vdash \Delta}$ $\neg_r$
$\forall$	$\frac{\Gamma, \forall_x A(x), A(t) \vdash \Delta}{\Gamma, \forall_x A(x) \vdash \Delta}$ $\forall_l$	$\frac{\Gamma \vdash \Delta, A(a)}{\Gamma \vdash \Delta, \forall_x A(x)}$ $\forall_r$
$\exists$	$\frac{\Gamma, A(a) \vdash \Delta}{\Gamma, \exists_x A(x) \vdash \Delta}$ $\exists_l$	$\frac{\Gamma \vdash \Delta, \exists_x A(x)}{\Gamma \vdash \Delta, A(t), \exists_x A(x)}$ $\exists_r$

Table D.1: First order classical sequent calculus

Bibliography

[1] Ontology web language for services (owl-s). Technical report, SEI, Carnegie Mellon University. [Online]. Available: http://www.sei.cmu.edu/isis/guide/technologies/owl-s.htm. [cited at p. 50]

[2] Introduction to uddi: Important features and functional concepts. Technical report, OASIS, October 2004. [cited at p. 54, 55]

[3] E. Franconi A. Artake and N. Guarino. Open problems with part-whole relations. In *Proceedings of 1996 International Workshop on Description Logics*, 1996. [cited at p. 33]

[4] H. Peter Alesso and Craig F. Smith. *Thinking on the web: Berners-Lee and Godel and Turing*. A John Wiley & Sons Inc., 2006. [cited at p. 1]

[5] Fortuna B, Mladenic D, and Grobelnik M. Semi-automatic construction of topic ontology. In *Proceedings of the ECML/PKDD Workshop on Knowledge Discovery for Ontologies*. [cited at p. 11]

[6] J. Yang B. Orriens and M.P. Papazoglou. A framework for business rule driven web service composition. In *International conference on Conceptual Modeling (ER), Chicargo, IL*, 2003. [cited at p. 47]

[7] F. Baader, B. Ganter, U. Sattler, and B. Sertkaya. Completing description logic knowledge base using formal concept analysis. Ltcs-report, Institute of Theoretical Computer Science, Dresden University of Technology, Germany, 2006. [cited at p. 2]

[8] Franz Baader and Ralf Kusters. Matching in description logics with existential restrictions revisited. LTCS-Report, 13, LuFg Theoretical Computer Science, RWTH Aachen, Germany, 1999. [cited at p. 65, 66]

[9] W. Balke and M. Wanger. Towards personalized selection of web services. In *WWW 2003*, pages 20–24, Budapest, Hungary, May 2003. [cited at p. 47]

[10] Sean Bechhofer. Owl reasoning examples. Technical report, University of Manchester, 2003. [Online] Available: http://owl.man.ac.uk/2003/why/20031203. [cited at p. 23]

[11] Sean Bechhofer and Ian Horrocks. *The WonderWeb Ontology: Language Layer*. University of Manchester, Kilburn Building, Oxford Road, 1.1 edition, February 2003. IST Project 2001-33052 WonderWeb. [cited at p. 3]

[12] H. Boley and S. Tabet. Ruleml rules lite concrete syntax. In *RuleML Initiative*, 2003. [cited at p. 50]

[13] Alex Borgida, Deborah McGuinness, Enrico Franconi, Ian Horrocks, and Peter F. Patel-Schneider. Explaining alc subsumption. Technical report. [cited at p. 21]

[14] K. Chao and M. Younas. Fuzzy matchmaking for web services. In *19 International Conference on Advanced Information Networking and Applications (AINA)*, pages 721–726, Taipei, Taiwan, 2005. IEEE Computer Society Press. [cited at p. 68]

[15] Kerber R Khabaza T Reinartz T Shearer C Wirth R. Chapman P, Clinton J. *CRISP-DM 1.0: Step-by-step data mining guide*, 2000. [cited at p. 11]

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L.Rivest, and Clifford Stein. *Introduction to Algorithms*, chapter 5. Probabilistic Analysis and Randomised Algorithms. MIT Press, 2001. [cited at p. 57]

[17] M. Chen D. Zhang and L. Zhou. Dynamic and personalized web services composition in e-business. *ISM-Journal*, 2005. [cited at p. 46]

[18] Olivier Dameron, Natalya F. Noy, Holger Knublauch, and Mark A. Musen. Accessing and manipulating ontologies using web services. Technical report, Stanford Medical Informatics, Stanford University, 251 Campus Drive, x-215, Stanford, CA 94305, USA. [cited at p. 69]

[19] J. Day and R. Deters. Selecting the best web service. In *In Proceedings of the 14th Annual IBM Centers for Advanced Studies Conference (CASCON)*, pages 293–307, 2004. [cited at p. 49]

[20] Alexander T. Borgida Deborah L. McGuinness. Explaining subsumption in description logic. In *In Proc. of the IJCAI'95*, pages 816–812, 1995. [cited at p. 20]

[21] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* John Wiley & Sons, Inc., second edition, 2001. [cited at p. 68]

[22] B. C. Grau E. Sirin and B. Parsia. From wine to water: Optimizing description logic reasoning for nominals. Technical report, Maryland Information and Network Dynamic Lab. [cited at p. 39]

[23] D. Martin et al. Bringing semantics to web services: The owl-s approach. [cited at p. 50, 51]

[24] D. Martin et al. *OWL-S: Semantic Markup for Web Services (Release 1.1)*. W3C, 2005. [Online]. Available: http://www.daml.org/services/owl-s/1.1/overview/. [cited at p. 2, 7, 12, 50, 52, 53, 54]

[25] I. horrocks F. Baader and U. Sattler. Description logics as ontology languages for the semantic web. Technical report, RWTH Achen, Germany and University of Manchester, UK, 2003. [cited at p. 10, 82]

[26] Smith P Uthurusamy R Fayyad U, Piatetski-Shapiro G. *Advances in Knowledge Discovery and Data Mining*. MIT Press: Cambridge, MA, 1996. [cited at p. 11]

[27] Deborah L. McGuinness Daniele Nardi Franz Baader, Diego Calvanese and Peter F. Patel-Schneider. *The description logic handbook: theory, implementation, and application*, chapter 2.Basic Description Logic. Cambridge University Press, 2003. [cited at p. 6, 12, 40, 76]

[28] Deborah L. McGuinness Daniele Nardi Franz Baader, Diego Calvanese and Peter F. Patel-Schneider. *The description logic handbook: theory, implementation, and application*, chapter 4.Relationships with other Formalisms. Cambridge University Press, 2003. [cited at p. 21]

[29] Gerald C. Gannod, Raynette J. Brodie, and John T. E. Timm. Interactive approach for specifying owl-s groundings. In *In Proceedings of the 2005 Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05)*. IEEE, 2005. [cited at p. 53]

[30] Jim Gleelen. Matching theory. *Euler Institute for Discrete Mathematics and its Applications University of Waterloo*, March 2001. Mini Course of Matching Theory. [cited at p. 61, 62]

[31] T. R. Gruber. A translation approach to portable ontology specification. *Knowledge Acquisition*, 5(2):199–220, 1993. [cited at p. 2]

[32] Jerry R. Hobbs and Feng Pan. An ontology of time for the semantic web. Technical report, Information Sciences Institute, University of Southern California. [cited at p. 33]

[33] Natalya F. Noy Holger Knublauch, Ray W. Fergerson and Mark A. Musen. The protege owl plugin:an open development environment for semantic web applications. Technical report, Stanford Medical Informatics, Stanford University, CA. [Online] Available: http://protege.stanford.edu. [cited at p. 12]

[34] Ian Horrocks, Peter F. Patel-Schneider, and Frank v. van Harmelen. From shiq and rdf to owl: The making of a web ontology language. *Journal of Web Semantics*, pages 7–26, 2003. [cited at p. 80]

[35] Ulrike Sattler Ian Horrocks and Stephan Tobies. Practical reasoning for expressive description logics. [cited at p. 78]

[36] D Trastour J. Gonzalez-Castillo and C. Bartolini. Description logics for matchmaking of services. Technical report, Trusted E-Service Laboratory, Hewlett-Packard, 2001. [cited at p. 46]

[37] A. Kalyanpur, B. Parsia, E. Sirini, B. Cuenca-Grau, and J. Hendler. Swoop: A web ontology editing browser. *Journal of Web Semantics*, 2, 2006. [cited at p. 3]

[38] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen. The protege owl plugin: An open development environment for semantic web applications. In *In Proc. of the Third International Semantic Web Conference*, 2004. Hiroshima, Japan. [cited at p. 3]

[39] Holger Knublauch. Ontology-driven software development in the context of the semantic web: An example scenario with protege/owl. Technical report, Stanford Medical Informatics, Stanford University, CA. [cited at p. 12]

[40] Holger Knublauch, Olivier Dameron, and Mark A. Musen. Weaving the biomedical semantic web with the protege owl plugin. Technical report, Stanford Medical Informatics, Stanford University, CA. [Online] Available: http://protege.stanford.edu. [cited at p. 12]

[41] Ojitha Kumanayaka and Nalin Ranasinghe. Ontology based web service personalization. In *IEEE second International Conference on Information and Automation 2006*, Galadari Hotel and Colombo and Sri Lanka, December 2006. IEEE Sri Lanka Section. [cited at p. 67]

[42] Ojitha Kumanayaka and Nalin Ranasinghe. Personalized web services creation via dynamic composition. In *IEEE First International Conference on Industrial and Information Systems*. Faculty of Engineering, University of Peradeniya, August 2006. [cited at p. 67]

[43] Ojitha Kumanayaka and Nalin Ranasinghe. Formalism for ontology validation. In *IEEE Second International Conference on Industrial and Information Systems*. Faculty of Engineering, University of Peradeniya, August 2007. [cited at p. 67]

[44] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web ontology. Technical report, University of Manchester. [cited at p. 33]

[45] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web ontology. Technical report, University of Manchester. [cited at p. 40]

[46] Werner Kießling. Foundation of preferences in database systems. report 2001-8, INSTITUT FUR INFORMATIK, UNIVERSITAT OF AUGSBURG, October 2001. [cited at p. 43]

[47] Werner Kießling and Bernd Hafenrchter. Optimizing preference queries for personalized web services. [cited at p. 43]

[48] M. Singh M. Huhns and et a. Research directions for service-oriented multiagent systems. *IEEE Internet Computing*, pages 65–70, November 2005. [cited at p. 49]

[49] C. Welty M. K. Smith and D. L. McGuinness. Web ontology language guide. Technical report, W3C, 2004. [cited at p. 2, 3, 7, 49]

[50] F. Manola and E. Miller. Rdf primer. Technical report, W3C, February 2004. . [Online] Available: http://www.w3.org/TR/rdf-primer/. [cited at p. 6, 49]

[51] E. M. Maxikilien and M. P. Singh. Agent-based architecture for autonomic web service selection. In *In AAMAS'03 Workshop on Web Services and Agent-Based Engineering (WS-ABE)*, 2003. [cited at p. 49]

[52] E. M. Maximilien and M. P. Singh. A framework and ontology for dynamic web service selection. *IEEE Internet Computing*, pages 84–93. [cited at p. 49]

[53] Franz Baader Ralf Molitor and Stefan Tobies. On the relation between conceptual graphs and description logics. Technical report, Aachen University of Technology Research group for Theoretical Computer Science, 11 1998. [cited at p. 28]

[54] Natalya F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford University, Standford, CA, 94305. [cited at p. 10]

[55] Feng Pan and Jerry R. Hobbs. Time in owl-s. In *In Proceedings of the AAAI Spring Symposium on Semantic Web Services*, pages 29–36. Stanford University, CA, 2004. [cited at p. 33]

[56] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycarar. Importing the semantic web in uddi. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA. [cited at p. 55]

[57] Massimo Paolucci and Katia Sycara. Autonomous semantic web services. *IEEE Internet Computing*, pages 34–41, September-October 2003. [cited at p. 2, 8]

[58] Massimo Paolucci and Matthias Wagner. Grounding owl-s in wsdl-s. In *IEEE International Conference on Web Services (ICWS'06)*. IEEE Computer Society, 2006. [cited at p. 68]

[59] P. F. Patel-Schneider, Patrick Hayes, and lan Horrocks. Owl web ontology language semantics and abstract systax. Technical report, W3C candidate recommendation, August 2003. [cited at p. 58]

[60] Alan Rector and Chris Welty. Simple part-whole relations in owl ontologies. Technical report, 2005. [online]. Available: http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/. [cited at p. 33]

[61] D. J. Russomanno and C. R. Kothari. Expressing inter-link constraints in owl knowledge bases. In *Expert Systems*, 2004. [cited at p. 50]

[62] Julian Seidenberg and Alan Rector. Web ontology segmentation: Analysis, classification and use. Technical report, World Wide Web Conference Committee, Edinburgh, Scotland, 2006. ACM 1-59593-323-9/06/0005. [cited at p. 10]

[63] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Journal of Web Semantics*, 2006. To be published. [cited at p. 12]

[64] N. Srinivasan, M. Paolucci, and K. Sycara. An efficient algorithm for owl-s based semantic search in uddi. In *Semantic Web Services and Web Process Composition (SWSWPC)*, San Diego, California, July 2004. [cited at p. 52]

[65] Naveen Srinivasan, Massimo Paolucci, and Katia Sycara. Adding owl-s to uddi implementation. Technical report. [cited at p. 55]

[66] Umberto Straccia. A sequent calculus for reasoning in four-valued description logic. Technical report, Istituto di Elaborazione della Informazione. [cited at p. 21]

[67] O. Lassila T. Berners-Lee, J. Hendler. The semantic web. *Scientific American, 284(5):34-43*, 2001. [cited at p. 1, 75]

[68] J. Zhang, L. Zhang, and J. Chung. An approach to help select trustworthy web services. In *Proceeding of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business (CEC-East'04)*, 2004. [cited at p. 49]

Index

Δ (multiset), 21
Γ (multiset), 21
$\mathcal{ALC}$, 21
$\mathcal{AL}$, *see* Attribute Language
$\mathcal{A}$, *see* ABox, *see* ABox
$\mathcal{I}$, *see* interpretation function
$\mathcal{L}$, 21
$\mathcal{SHIQ}$, 17, 78
$\mathcal{SHOQ}(\mathbf{D})$, 78
$\mathcal{SI}$, 78
$\mathcal{S}$, 28
$\mathcal{T}$, 12, *see* TBox
.NET, 4

disjoint vertex, 61
matching algorithm, 56

ABox, 17, 76, 80
    instance checking, 20
abstract characterisation, 46
abstract syntax, 13, 14
agent
    enterprise application agent, 47
    mobile agent, 47
algebraically independent, 63
artificial intelligences, 1
assertions (Σ), 23
atomic concept, 76, 77
atomic role, 76–78
Attribute Language, 77
automated process, 47
axiom, 77, 78

binary predicate, *see* binary predicate
bipartite graph, 28, 61
    exact matching, 61
bipartition, 63

Calendar date, 38
choreography, 49
classification, *see* taxonomic hierarchy
cluster analysis, 56, 57
computational complexity, 1
concept assertion, 76, 80
concept expressions, 76
concept node, 28
Conceptual Graph, 28
    Simple Conceptual Graph, 28
credit card, 54
CRISP-DM, 11
    Business understanding, 11
    data mining, 12
    Data preparation, 12
    Data understanding, 11
    Deployment, 12
    Evaluation, 12
    Modeling, 12

Data types, 46
database, 11
debugging, 20
deduction tree, 23
deductive framework, 20
degree of matching, 55
derivation tree, 22
Description Logic, *see* DL
descriptive logic, 1
directory service, 47
DL, 6, 10, 75
    base symbol, 79
    classification, 6
    Classifier, 19
    complement, 13
    complementOf, 13
    declaration, 13
    defined concept, 79
    definition, 79

disjointness, 19
equality, 79
fill, 80
fixedpoint, 79
individual, 20
inverse role, 18
knowledge base, 79
name symbol, 79
one-of, 80
reasoner, 12
role, 20
satisfiability, 17
sub role, 18
subsume, 78
subsumee, 78
subsumer, 78
subsumption, 6, 17, 21, 56
symbolic name, 12
transitive role, 19
Transitiveness, 33
unionOf, 14
DoM, *see* degree of matching, 56, 58
disjoint, 56
Exact, 56, 61
Intersection, 56
PlugIn, 56, 61, 62
Subsume, 56, 61
domain, 47

E-business, 47
edge relation, 28
entailment, 23
entailment($\vdash$), 21
exact matching, 64
execution flow, 46
existential quantification, 77
existential quantification ($\exists$), 27
expressiveness, 46
extensional knowledge, 76

Family Ontology, 12
fixpoint, 79
flexibility, 46
Ford-Fulkerson algorithm, 62
free variable, 21
functional requirements, 2, 32
fuzzy logic, 1

generic, 28
marker, 28
Gregorian calendar, 37

Hall's Theorem, 61
hard-coded, 8
human
intervention, 6

inclusion, 78
individual concept node, 28
individuals, 76
inference, 1
ABox inference, 20
Class inference, 23
engines, 6
inference procedure, 76
inferences, *see* inference
instance checking, 23, 76
intentional knowledge, 76
interoperability, 7
interpretation function, 79
Intersection, 61
inverse, 33
IOPE, 50–52, 54
IS-A relationship, 78

J2EE, 4

knowledge
acquisition, 11
alignment, 11
configuration management, 11
construction, 10
discovery, 11
implicit, 17
integration, 11
merging, 11
knowledge bases, 65
knowledge discovery, 81
knowledge representation, 76
knowledge representation systems, 76
Konig Theorem, 62

locative model, 38
logic, 7
equivalence, 12
FOL, 6, 7, 21, 76
predicate logic, 7, 21
proposition model logic, 78
logical consequence, 23

machine intelligence, 1
markup language, 75
Pyramid, 7
MAS, 49, *see* multi-agent system

matching modulo subsumption, 66
matchmaker, 2, 6, 46, 47, 51, 55, 61
matrix, 63
maximum cardinality, 61
maximum matching, 61, 63
membership assertion, 76
methodology, 10
middle agent, 49
model theoretic semantics, 17, 76
monotonic inheritance hierarchy, 78
multi-agent system, 49

necessary and sufficient, *see* sufficient and necessary, *see* sufficient and necessary
necessary condition, 13
necessary conditions, 78
negation, 77
neural networks, 1
nominal, 39, 80
nonfunctional properties, *see* nonfunctional requirements, 32
nonfunctional requirements, 2
normalisation-Compare, 20

object properties, 75
ontological matching, 59
ontology, 1, 2, 6, 46, 81
  analyse, 21
  axiom, 75
  common, 47
  concepts, 75
  conceptualise, 10
  construction, 10, 11
  development, 11
  domain expert, 11
  engineer, 11
  engineering, 10
  formalise, 10
  implement, 11
  instance, 75
  language, 10
  life cycle, 10
  management, 10
  personalisation, 2, 10, 59
  properties, 75
  specify, 10
  user, 11
ontology alignment, 81
ontology construction
  fully-automatic, 11
  semi-automatic, 11
optimization mechanisms, 46
Ordinal date, 38
ordinal dates, 37
OWL, 6, 8, 23, 33, 49, 50, 56, 75, 77
  abstract semantic, 58
  intersectionOf, 13
  OWL-DL, 7
  OWL-full, 7
  OWL-Light, 7
OWL-S, 8, 50
  atomic process, 54
  composite process, 54
  control flow, 54
  grounding, 50
  process model, 8, 50
  service grounding, 8
  service profile, 8, 50, 51, 53
    inputs, *see* IOPE
    outputs, *see* IOPE
    preconditions and effects, *see* IOPE
  services, 2

perfect matching, 64
personalisation, 2
  agent, 49
  information, 47
  policies, 3
Personalisation ontology, 12
personalised service, 46
preferences, 46
preserving, 21
primitive concept, 29
primitive role, 29, 78
probabilistic reasoning, 1
probability analysis, 57
proxy agent, 49

QoS, 47, 49, 51
query analysis, 23

ranking scheme, 49
RDF, 6, 49, 75
  object, 75
  predicate, 75
  subject, 75
RDF Schema, 33
reasoning
  automatic, *see* inference
reasoning capabilities, 76
relation node, 28
requester, 2
role assertion, 76, 80

role expressions, 76
role filler, 29
role hierarchy, 78

satisfiability, 76
semantic
    preference, 51
semantic agreement, 46
semantic frame, 76
Semantic level of agreement, 46
semantic matchmaker, 46
semantic networks, 76
semantic suitability, 51
semantic web, 1, 75
    architecture, 6
    expressiveness, 2
    infrastructure, 8
    languages, 7
    markup language, 6
    ontology development, 6
    service discovery, 46
sequence diagram, 56
sequent calculus, 23
service
    advertisement, 47
    consumer, 47
    description, 47
    discovery, 32, 47
        quality, 47
    functionality, 47
    instance, 47
    lifecycle, 54
    negotiation, 32
    personalised, 47
    provider, 47
    registry, 47
        performance, 47
    request, 47
    requester, 32, 47
    selection, 32, 47
    substitution, 32
service annotation ontology, 51
service availability, 33
service composition
    dynamic, 47, 49, 50, 53
    plug-and-play, 47
    static, 47
Service Oriented Computing, 49
service profile, 47
service provider, 2
SGML, 4
shared knowledge, 1
SOC, *see* Service Oriented Computing
soft-computing, 1
software agents, 8
static profile, 47
structured inheritance networks, 76
Subsume, 63
subsume matching, 64
sufficient and necessary, 12, 19

Tarski-style, 21
taxonomic hierarchy, 76
taxonomies, 2
TBox, 17, 76
temporal availability, 32
temporal duration, 38
temporal model, 33
terminological definition, 20
terminology, 76
    acyclic, 76, 79
    cyclic, 79
terminology construction, 76
time ontology, 33
time polynomial, 62
transitive property, 78

UDDI, 50, 52
    businessEntity, 55
    businessService, 55
    integration, 55
    tModel, 52, 54, 55
UMFA, 49, *see* User centred, Mobile agent based, Fault injection-equipped and Assertion oriented
UML, 50
unary predicate, *see* atomic concept
unique name assumption(UNA), 80
universal quantification (∀), 27
upper ontology, 8
User centred, Mobile agent based, Fault injection-equipped and Assertion oriented , 49
user introversion, 51

variable free syntax, 21

W3C, 6
web intelligence, 1
web service, 1
    client paradigm, 47
    compose, 8
    composition, 46, 50

description, 32
discover, 8
dynamic composition, 11
enactment, 50
grounding, 50
infrastructure, 7
interaction, 8
pattern, 8
invocation, 50
invoke, 8
matching, 51
monitoring, 50
recovery, 50
semantically suitable, 2
Web Service Agent Framework (WSAF), 49
Web Service Agent ProxyArchitecture, 49
Week date, 38
Whole and part relation, 33
worst case, 58
WSDL, 50, 52, 55

XML, 4, 7
schema, 6