

Web Services Search and Composition by Combining Web 2.0 and Semantic Web Technology



研究生：楊德威
指導教授：蔡益坤

Department of Information Management
National Taiwan University

July 18, 2008



THESIS ABSTRACT
GRADUATE INSTITUTE OF INFORMATION MANAGEMENT
NATIONAL TAIWAN UNIVERSITY

Student: Yang, Te-Wei
Advisor: Tsay, Yih-Kuen

Month/Year: June, 2008

**Web Services Search and Composition by
Combining Web 2.0 and Semantic Web Technology**

Web Services which are specific functionalities and can be combined to meet a particular user's needs have become a mature technology in the past few years. However, the discovery and search mechanism provided by UDDI based on keyword matching may lead to an ambiguous answer. It is a challenge to target the suiting Web services precisely. Semantic Web technology provides another option for service matching. It enables a service profile to be described according to its functionalities in OWL, which is based on Description Logics. Recently, researchers are dedicated on studying Semantic Web technology as a primary tool for ontology-based Web Services searching and invocation. With help of precise semantics description, Web Services are able to be utilized automatically.

Under such a Semantic Web search mechanism, Web Services profile and domain ontology are both described by Description Logics. However, potential users often do not have any knowledge about Description Logics. That creates a huge gap and critically imposes high entrance barriers for the user. Besides, ontology maintenance is another important issue for Semantic Web applications. Ontology maintenance is a time-consuming job. Ontology maintenance is usually controlled by a small group of people. But it has several drawbacks: (1) the addition can be time-consuming and lack of completion and (2) the ontology maintainer read the concept in the different manner from how potential user does. Accordingly, sometimes concepts become obsolete by the time they enter the ontology. In the long run, ontology maintenance cannot be ignored especially in such a Semantic Web application.

In this thesis, we proposed: (1) an open system architecture to lower the entrance barriers of Semantic Web applications, (2) a practical approach to ontology maintenance, and (3) a new prototype system. The Traveller was implemented based on our ontology-based architecture and related methodologies. With the service composition and execution architecture, the user is able to find suiting Web Services, invoke services by defining BPEL4WS, and participate in collaborative ontology maintenance without knowing any Semantic languages.

Keywords: AJAX, BPEL, BPEL4WS, Description Logics, Ontology, OWL, Semantic Web, Semantic Web application, Semantic Web Service, Service Execution, SWRL, Web Services, Web 2.0, Protégé,

Contents

1	Introduction	1
1.1	Background	1
1.2	Motivation and Objectives	2
1.3	Thesis Outline	4
2	Related Work	5
2.1	Web Services	5
2.1.1	Web Service Description Language(WSDL)	6
2.1.2	UDDI	8
2.1.3	SOAP	9
2.2	Semantic Web	10
2.2.1	Resource Description Framework(RDF)	10
2.2.2	Web Service Modeling Ontology(WSMO)	11
2.2.3	OWL-S	12
2.2.4	Modeling Ontology of Time and Value	14
2.3	Service Matching and Ranking	16
2.3.1	Service Matching	16
2.3.2	Service Ranking	18
2.4	Web Services Composition	20
2.5	Related Projects	20
2.5.1	SATINE Project	20
2.5.2	European Semantic Systems Initiative (ESSI)	21
2.5.3	EON Architecture	23
3	Preliminaries	25
3.1	Description Logics	25
3.1.1	Description Logics Syntax and Semantics	27
3.2	OWL	30
3.3	Semantic Web Rule Language: SWRL	31
3.3.1	SWRL Editor	31
3.4	Quantitative Relations	32
3.5	Web Service Description Language(WSDL)	36
3.6	Web Services Business Process Execution Language(WS-BPEL)	38
3.7	Web 2.0 Technology	39

4	Service Composition and Execution Based on Semantic Technology	41
4.1	Overview of Web Services Composition Architecture Based on Semantic Technology	41
4.1.1	Web Services Composition	42
4.1.2	Design of Web Services Composition Architecture Based on Semantic Technology	46
4.2	Service Composer	52
4.2.1	Design of the Service Composer	52
4.2.2	Architecture of the Service Composer	54
4.3	Knowledge Base Management System	58
4.3.1	Design of the Knowledge Base Management System	58
4.3.2	Architecture of the Knowledge Base System	58
4.4	Ontology Modeling	62
4.4.1	Service Composition Mechanism	67
4.4.2	Service Execution Based on Semantic Technology	69
4.5	Constraint Handling	69
4.5.1	Constraints	70
4.5.2	Time Constraint	71
4.5.3	Value Partition	72
4.6	Ontology Maintenance	73
4.6.1	Wiki-supported Ontology Engineering	73
4.6.2	The Model of Ontology Maturing	75
4.6.3	Wiki Community Component and Ontology Maintain Procedure	76
4.7	Service Execution	79
4.7.1	Design of the Service Execution Module	79
4.7.2	Architecture of the Service Execution Module	79
4.7.3	Development of the Business Process Execution Language	81
5	Implementation - The Traveller	85
5.1	The System Design	85
5.2	Service Description	87
5.2.1	Trip Requirement Description	88
5.2.2	Service Advertisement Description	89
5.3	Implementation of the Traveller	90
5.3.1	Implementation of the Service Composer	92
5.3.2	Implementation of the Knowledge Base Management System	94
5.4	Ontology Design	97
5.4.1	The Tourism Domain Ontologies	97
5.4.2	The Spot Ontology	97
5.4.3	The Requirement Ontology	100
5.4.4	The Advertisement Ontology	103
5.5	Constraint Checking	106
5.5.1	Time Constraints	108
5.5.2	Budget Constraints	110
5.6	Constraint Rules	112
5.6.1	PAL Rules	112

5.6.2	SWRL Rules	113
5.7	The Traveller Demonstration	115
5.7.1	Matching Service Process	115
6	Conclusion	118
6.1	Contributions	118
6.2	Future Work	119



List of Figures

1.1	Web Services Architecture	2
2.1	Web Services Architecture	6
2.2	WSDL Binding Example	7
2.3	Relation between UDDI data structures	9
2.4	Top level of the Service Ontology	13
3.1	Architecture of a knowledge representation system based on Description Logics.	26
3.2	The SWRL Editor in Protégé	33
3.3	Relationship between SWRL and Ontology	34
4.1	Web Services Composition Architecture	47
4.2	Architecture of the Service Composer	54
4.3	Architecture of the Knowledge Base System	59
4.4	Requirement Modelling - The TBox Approach	64
4.5	Requirement Example - The TBox Approach	65
4.6	Ontology Design of the Architecture - Requirement, Advertisement and Common Ontology	66
4.7	Example of Subsumption Reasoning	67
4.8	The Classification of Constraints	71
4.9	The ValuePartition Ontology	72
4.10	Architecture of the Service Execution Module	81
4.11	Life-cycle of the Business Process Execution Language	84
5.1	The Implementation of the Service Composer	94
5.2	The Implementation of the Knowledge Base Management System	96
5.3	The Tourism Ontology Design	98
5.4	Design of the Spot Ontology (Part)	99
5.5	The Ching Jing Farm Service Profile	100
5.6	The Requirement Ontology in the Traveller	102
5.7	The Requirement Example in the Traveller	103
5.8	The Requirement Concept Definition in the Traveller	104
5.9	The Trip Requirement of the scanrio in the Traveller	105
5.10	The Advertisement Ontology in the Traveller	106
5.11	The Advertisement Example in the Traveller	107
5.12	The Advertisement Concept Definiton in the Traveller	108
5.13	The Trip Package Example	109

5.14	Constraint Handling and Checking in the Implementation System	109
5.15	Relationship between leq5000 and leq500 in the ValuePartition Ontology	111
5.16	The User Interface of Matchmaker	117



List of Tables

2.1	Comparison between WSMO and OWL-S	15
3.1	Description Logics Concept Constructors	28
3.2	Terminological and Assertional Axioms	29
3.3	Abstract Syntax of SWRL	32
3.4	Semantics of Quantitative Concepts	34
3.5	Semantics of Quantitative Relations	36
5.1	Implementation of the components of the Traveller	91



Chapter 1

Introduction

1.1 Background

Web Services which are specific functionalities and can be combined to meet a particular user's needs have become a mature technology in the past few years. Service Oriented Architecture(SOA) [4] provides a distributed, loosely coupled, and open standard architecture which is able to create a platform-independent mechanism over the Internet, combine services together and reuse them to achieve business applications. Aside from Service Oriented Architecture, W3C has developed several related standards to support Web Service, such as Simple Object Access Protocol (SOAP) [11], Web Service Description Language (WSDL) [15], Universal Description, Discovery, Integration (UDDI) [2], and Web Services Business Process Execution Language (WS-BPEL) [8]. SOAP is a communication protocol adapted in Web Services today. Through XML-based message communication, applications are able to exchange information in a decentralized, distributed, and heterogenous environment. Using WSDL, a user can locate Web Services and invoke any of its publicly available function. UDDI enable businesses to publish service lists and have them to be found on the Internet. In other word, UDDI provides a platform-independent service registry for businesses worldwide. The role of WS-BPEL is to define a specific Web Services by composing a set of existing services. Figure 1.1 shows that the basic architecture of Web Services involves three main roles: service provider, service registry, and service requestor.

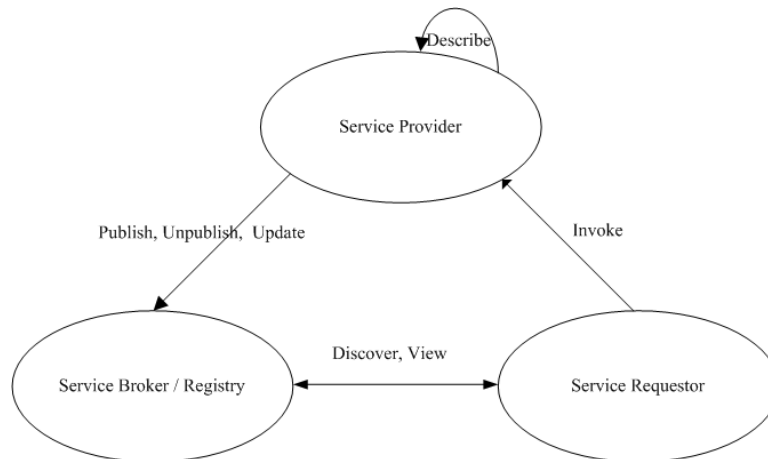


Figure 1.1: Web Services Architecture

1.2 Motivation and Objectives

Just as previous stated, Web Services development is continually booming. However, the discovery and search mechanism provided by UDDI based on keyword matching may lead to an ambiguous answer. It is a challenge to target the suiting Web services precisely. Keyword search mechanism is not an efficient methodology, because there may be different meanings to a single keyword and cause a failed search outcome. Therefore, we try to combine Semantic Web and Web Services technology together, which means Web Services description written in XML format can be annotated with semantics. By this method, search machines like computers are able to understand the exact semantic of Web Service, and find out the best description that matches the user's need. There are many researches on the Semantic Web Services. They try to use expressive Description Languages, such as Description Logics [9] to equip service profile with semantics. Description Language, which is a formal logic-based system and developed to conceptualize knowledge and to model Ontology, has extended many Ontology languages applied in Sematic Web Service, such as RDF [35], DAML+OIL [25], and Web Ontology Language (OWL) [18]. Recently, researchers are dedicated on studying Semantic Web technology as a primary tool for ontology-based Web Services searching and invocation rather than keyword-based search mechanism. With help of precise semantics description, Web Services are able to be utilized automatically.

Under such Semantic Web mechanism, Description Logics play an important role. Web Services profile, domain ontology are both described by Description Logics. Our matching scheme also relate with Description Logics. However, potential users often do not have any knowledge about that [44]. It creates a huge gap and critically imposes high entrance barriers for Semantic Web applications. Ontologies in Semantic application need constant updates and maintenance so that they correctly represent the real world. But it is a time-consuming job for those administrators. The managements of ontology need to be more efficient and more organized so that computers can use those plentiful and correct ontologies in reasoning tasks to deduce the right results. In the traditional approach, ontology maintenance is controlled by a small group of people. But the fact shows that small group constructs the ontology for a bigger group has several drawbacks: (1) the addition can be time-consuming and lack of completion (2) the ontology maintainer read the concept in the different manner from how potential user does. Accordingly, sometimes concepts become obsolete by the time they enter the ontology. In the long run, ontology maintenance cannot be ignored especially in such a Semantic Web applications.

The goal of this thesis is to propose a new architecture based our previous architecture of "automatic service composition and execution for Web Services" [57]. In our previous architecture, we have identified some issues from casual users' perspective and administrators' perspective. We want to provide new features that not only make our system more friendly for casual users and also for administrators. In new architecture, we want to create a open environment where every user participates in update and maintenance. We implement a new prototype system called the Traveller, which acts as a trip planner on the Internet that handles the customer's tourism requirements. By combining Web 2.0 technology [46], such as community and mashups, the entrance barrier of Semantic Web applications will be lower.

In this thesis, we proposed: (1) an open system architecture to lower the entrance barriers of Semantic Web applications, (2) a practical approach to ontology maintenance, and (3) a new prototype system. The Traveller was implemented based on our ontology-based architecture and related methodologies. With the new service composition and execution architecture, the user is able to find suiting Web Services, invoke services by defining BPEL4WS, and participate in collaborative ontology maintenance without

knowing any Semantic languages.

1.3 Thesis Outline

The remainder of this thesis is organized as follows:

- In Chapter 2, we introduce the development of the Web Services and Semantic Web in recent years. Then we review some related researches about Semantic-based service matching and service ranking. Besides, composition of Semantic Web Services and ontologies modeling of Time and Value are discussed with some relevant papers. In the end of chapter 2, we introduce Web 2.0 technology and some ongoing projects related to our thesis, such as SATINE [22] which is a travel domain project based on Semantic Web Services, European Semantic Systems Initiative (ESSI) which dominates six projects specialized in certain aspects of Semantic Web services.
- In Chapter 3, we introduce basic notations and axioms about Description Logic, basic inference problems in Description Logics. Otherwise, Web Ontology Language (OWL), an ontology language, Semantic Web Rule Language (SWRL) [26], which extends OWL axioms to support specific rules, and other related Web Service standard: WSDL and WS-BPEL, will be introduced.
- Chapter 4 explains service composition and execution based on semantic technologies. The relationships between requirements and advertisements are described. The architecture of ontology are detailed. We also proposed a classification of constraints and their handling approaches. Besides, the service execution module is proposed and explained in this chapter.
- Chapter 5 describes the prototype system, *the Traveller*, which is applied in tourism domain. The ontologies modelling of the system and interactions among components in the system will be represented in detail.
- Chapter 6 summaries the contributions of this thesis, and its future direction for further research.

Chapter 2

Related Work

In this chapter, We will review related works about the Web Service, Semantic Web Service, some researches in modeling Ontology, several papers about service matching and ranking, and related projects working recently.

2.1 Web Services

Web Services is a software system designed to support computers interaction over a distributed system environment. By accessing server service, it allows different applications from different sources to communicate with each other without time-consuming custom coding. In the following, We will introduce main architecture components.

- **Service Provider:** A service provider implements the service and make it available on the Internet by publishing its service descriptions to the service registry.
- **Service Requester:** A service requester utilizes an existing Web Services by searching service registry and invoking qualified services.
- **Service Registry:** A service registry serves as a service broker, which performs a matching scheme and sends qualified service descriptions back.

In order to ensure service interoperability. Every component in Web Services architecture exchanges message with Simple Object Access Protocol(SOAP) which is XML-based communication standard. Service profiles are described by Web Service Description Language(WSDL), and are stored at Universal Description, Discovery, Integration(UDDI)

Registry where Service Requester search for their Web Service. Finally, Service Requester make a remote procedure calls(RPCs) to invoke the business function on the Service Provider. Figure 2.1 shows how Web Services Architecture work.

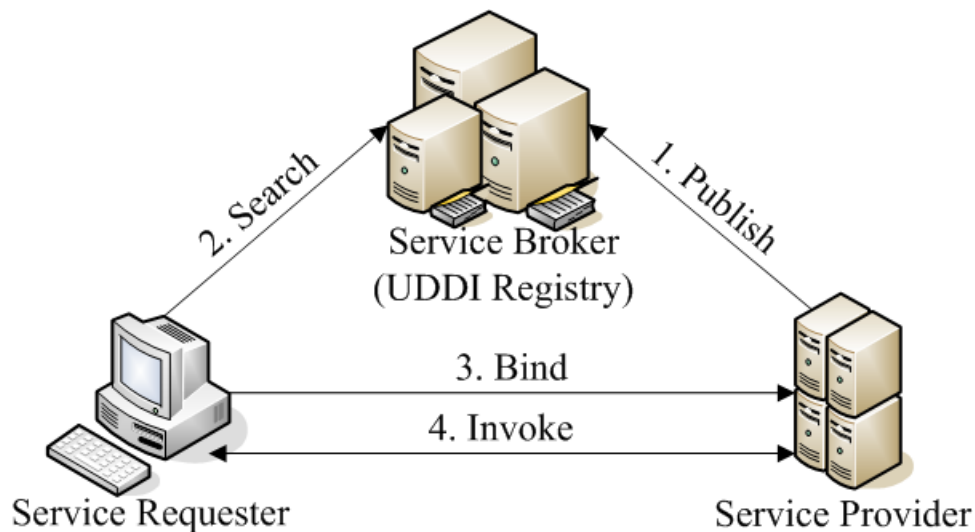


Figure 2.1: Web Services Architecture

2.1.1 Web Service Description Language(WSDL)

WSDL is an XML document describing a Web Service. WSDL document has two types of elements: **Abstract** and **Concrete**. The Abstract part, which is also known as interface definition, describes what data types and message format should be used, what functions are supported by the service. The Concrete part describes how the service will be used over network and where the service locates. In the Concrete part, services are well implemented in details, some implementation specifications are defined here

A service interface definition, which is reusable, describes a service in terms of message format and its operations. A service implementation definition provides actual implementation details, such as access points and service provider information.

The Abstract part definition includes four elements, **Types**, **Message**, **PortType**, and **Binding**.

1. **Types**, defines primitive data types used to form messages. WSDL prefers XML-Schema as the canonical type system.

2. **Message**, defines the name and content of the request/response messages, which are the parameters of operations. Message element will be removed in WSDL2.0 [15], since message types can be directly defined in Types element.
3. **PortType**, renamed as **Interface** in WSDL2.0, is a named set of operations involved.
4. **Binding**, defines message format and protocol details for operations and messages defined by a particular portType(Interface). Figure 2.3 introduces the concept of service binding.

The Concrete part definition includes two elements, **Port** and **Service**.

1. **Port**, renamed as **Endpoint** in WSDL2.0, describes the access point of service when perform data transmission.
2. **Service**, a collection of related endpoints, defines the address for invoking the specified service.

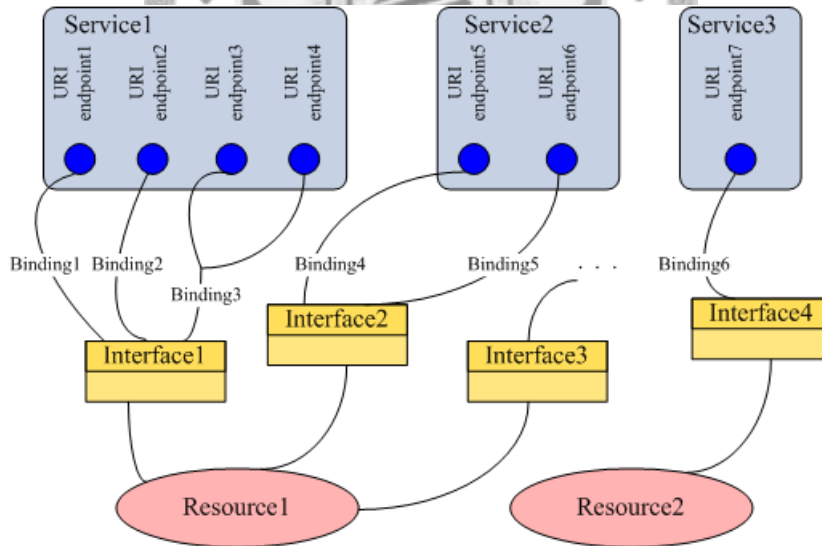


Figure 2.2: WSDL Binding Example

2.1.2 UDDI

WSDL has defined a standard for service description and service accessing. However, the user still needs a central registry that stores service profiles and provide service discovery in a efficient way. That's the functionality UDDI supply for. UDDI was developed by IBM, Microsoft, and Ariba. It defines a explicit specification for service discovery and service publishing, and solidifies the integration of Web Service.

UDDI's registration information consists of four data structure types including **businessEntity**, **businessService**, **bindingTemplate**, and **tModel**. Figure 2.3 describes the relations between these data structures.

- **businessEntity**: The businessEntity structure represents all known information about a business or entity that publishes information about the entity and what services it offers.
- **businessService**: The businessService structure describes services provided by a specified businessEntity, including service name, service key, and service description.
- **bindingTemplate**: The bindingTemplate structure, which was included in businessService, provide support for determining a technical entry point or optionally support remotely hosted services
- **tModel**: The tModel structure has two main uses. One is defining the technical specification and the other is defining an abstract namespace reference.

There are three general categories which are the way how UDDI stores service information.

- **White Pages**: White pages provide basic contact information about a company, such as the business name, address and contact information. White pages also provide unique business identifiers, such as domain name. In short, white pages allow customers and partners to discover business services based upon business identification.
- **Yellow Pages**: Yellow pages describe a business service using different categorizations (“taxonomies” in UDDI terminology). This information allows others to

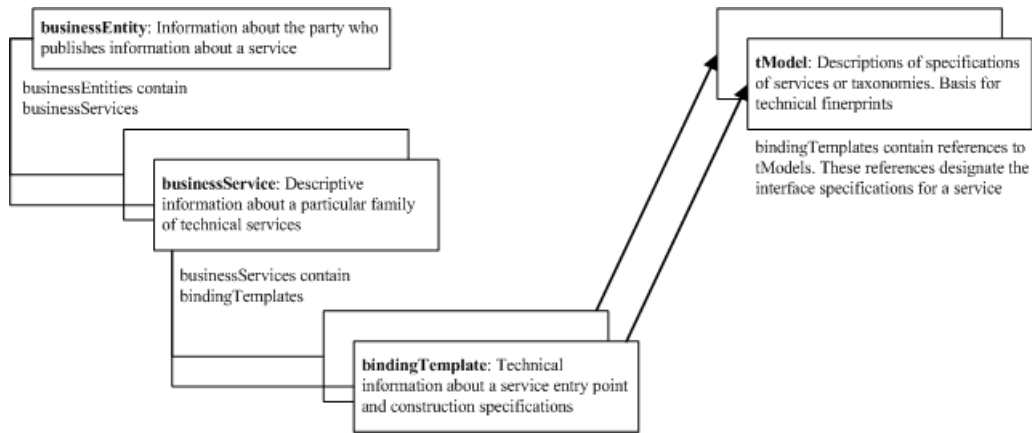


Figure 2.3: Relation between UDDI data structures

discover business services based upon its categorization, such as being in the manufacturing or software development business.

- **Green Pages:** Green pages provide technical information on the behaviors and supported functions of a business service hosted by a business. Green pages in UDDI are not limited to describing XML-based Web Services, but any business service type offered by a business entity, such as phone-based services, for example call center service.

As we mentioned in last chapter, although UDDI play an important role as a service broker, but we do not consider UDDI's keyword-based search as a good approach. Because different semantics may be referring to the same word, Keyword-based may cause ambiguous results sometimes. In our approach, we prefer semantic can be embedded into the search system, make service description more clear and more articulate.

2.1.3 SOAP

SOAP provides the definition of the XML-based information which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment. SOAP message is formally specified as an XML Information Set called XML Infoset, which provides an abstract description of its contents. However, SOAP provides the framework by which application-specific information may be conveyed in an

extensible manner. SOAP is mainly used for performing remote procedure calls (RPCs) transported via HTTP. Unlike other RPC technologies, such as CORBA, JAVA RMI and DCOM, SOAP messages are entirely written in XML. Therefore, services can be invoked without platform limitations.

2.2 Semantic Web

Semantic Web, which is brought up by Tim Berners-Lee, is considered as a next generation of World Wide Web. Tim Berners-Lee issued the call for the Semantic Web, because he found that HTML, the syntax of the Web, did not include enough meaning. He suggested a syntax that could capture the meaning expressed in our daily life in a way that computers could process. Therefore, the goal of Semantic Web is to make data over network understandable and accessible to human and machines. Under such mechanism, we can imagine that business process can be initiated by people and then proceed on its own. All the tasks within the process would be interactions between machines, without human participation. The other advantage it brings is the improvement of service discovery. By semantic-embedded service description, a service broker can match services according to their functionalities. Therefore, in this section, we introduce two standards of Semantic Web.

2.2.1 Resource Description Framework(RDF)

Resource Description Framework (RDF) is designed by World Wide Web Consortium (W3C). The original idea is establish a standard of metadata model but which has come to be used as a general method of modeling information, through a variety of syntax formats. The RDF metadata model is based upon the idea of making statements about resources in the form of subject-predicate-object expressions. In the other word, RDF can express the relationship between two terms, like Apples are a type of fruit; Homer is the father of Bart. By putting all relationships together, we can construct an ontology. Because of RDF's simple data model and ability to model disparate concepts, it has not only led to its increasing use in knowledge management applications but also animated the expand of Semantic Web activity.

2.2.2 Web Service Modeling Ontology(WSMO)

In order to create a Web Service Modeling Ontology [51], which is abbreviated to WSMO, for describing abundant Semantic of Web Service. The ESSI ¹ WSMO group works for explicit standardization in Semantic Web Services language, and try to use a common architecture to represent the standard of Semantic Web Service. The architecture called Web Service Modeling Framework(WSMF) consists of four different main elements: **Ontologies** that provide the terminology used by other components in WSMF, **Goal Repositories** that define the user's problems that should be solved by Web Services; **Web Services** descriptions that define various aspects of Web Services in detail; and **Mediators** which are responsible for interoperability problems in connecting heterogeneous datum, processes, and protocols between WSMF elements.

which provide the conceptual model for semantically accomplishing the functions of Web Services including automatic Web Services publishing, Web Service discovering, Web Services composition, and execution, the group aims at developing the language called Web Services Modeling Language (WSML) that formalizes the Web Services Modeling Ontology (WSMO) and focusing on a framework called Web Services Modeling Framework (WSMF) that develops a fundamental execution environment.

Detailed descriptions about these components are as follows:

- **Ontologies:** According to Gruber's definition about Ontology: An ontology is a formal, explicit specification of a shared conceptualization. [23] In WSMF, ontologies are used to define the terminology that is used by other elements of WSMF specifications. Therefore, they enable reuse of terminology as well as interoperability between components referring to the same or linked terminology. Ontologies, which was developed in Artificial Intelligence to facilitate knowledge sharing and reuse, are formal and consensual specifications of conceptualizations that provide a shared and common understanding of a domain, an understanding that can be communicated across people and application systems. It define formal semantics for information, consequently allowing information processing by a computer.
- **Goal Repositories:** The description of a goal specifies objectives of fulfillment

¹<http://www.essi-cluster.org/>

arrived by executing the Web Services and that user may have when he consults a Web Service. A goal specification consists of two elements: Pre-conditions describe what a Web Services expects for enabling it to provide its service. Post-conditions describe what a Web Services returns in response to its input.

- **Web Services:** Web Services represent service entities, which provide certain functional tasks in a domain. Web Services descriptions consist of its non-functional properties, its functional properties, and the behavioral aspects of a Web Service. These properties and aspects of Web Services are described by using the terminologies defined in Ontologies.
- **Mediators:** The Mediators allow one to link heterogeneous resources and are proposed to overcome the interoperability problem between different WSMO elements. The Mediators not only solve the data heterogeneity problem, but also deal with process, and protocol heterogeneity. WSMO defines four types of Mediators, such as ggMediators, ooMediators, wgMediators, and wwMediators.

2.2.3 OWL-S

As we mentioned above, WSDL document contains sufficient information for user to invoke a service, but semantic about service description is inadequate, such as "how service works", "what is its precondition" and postcondition, and even "who provides the services". Base on OWL and predefined language of DAML+OIL [16, 25], OWL-S [18] is proposed. However, OWL-S supplies Web Services providers with markup language constructs for increasing the expressive power and semantics of the properties and capabilities of their Web Services. . When information is in unambiguous, computer-understandable form, OWL-S markup of Web Services will facilitate the automation of Web Services tasks including:

- Automatic Web Services Discovery
- Automatic Web Services Execution
- Automatic Web Services composition and interoperation

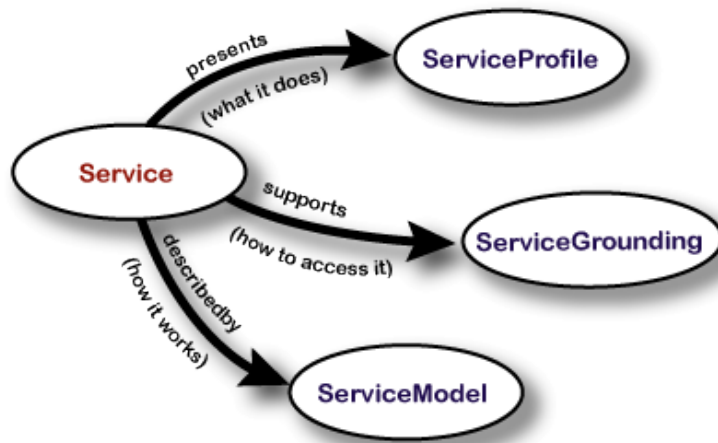


Figure 2.4: Top level of the Service Ontology

- Automatic Web Services execution monitoring

OWL-S does not aim to replace the current standard of Web Services, but attempts to increase the capability of semantic level interoperability. To fulfill the task, OWL-S constructs the Upper Ontology that consists of **Service Profile**, **Service Grounding**, and **Service Grounding** described below:

- **Upper ontologies:** Upper ontologies defines three types of knowledge about the different aspects of services; Service Profile, which defines what services are provided, Service Process Model, which defines how services work, Service Grounding, which defines how services interact. Figure 2.4 shows the top level of the Service Ontology.
- **Service Profile:** Service Profile represents what services are offered by the service and descriptions about the services. Based on OWL subclassing approach, the detailed information of services, such as who provides the service, what are the inputs, outputs, preconditions, consequences, a list of features of services, quality rating of a service, and an unbounded list of service parameters. Among registry information in terms of the capabilities and description of the services, Service discovering and matching can be implemented into Web Services standard like UDDI or other inferencing mechanisms.

- **Service Process Model:** In Service Model, there is a minimal set of control constructs used to represent a variety of process of Web Services. It is including Sequence, Split, Split-Join, Any-Order, Choice, If-Then-Else, Iterate, Repeat-While, and Repeat-Until,. The behavior that Service Model specified should be consistent with the descriptions in Service Profile.

A *Process* can be an atomic process or composite process. In the former one, it expects one (possibly complex) message and returns one (possibly complex) message in response. In the latter one, it maintains some state so that each messages from clients can be recognized and be arranged in the correct order.

- **Service Grounding:** Service Grounding gives a concrete level of service specification including the access point of the service, the communication protocols used, and the message passed during its execution.

Lara [34] made a comparison between WSMO and OWL-S. In a nutshell, OWL-S tries to construct the description of services in a broad sense, not focusing on a specific domain. WSMO aims to create an ontology for describing services in a more defined focus: solving the integration problem. Table 2.1 gives a brief comparison between the two standards.

2.2.4 Modeling Ontology of Time and Value

Some researchers refer ontology to the study of conceptualization of reality. Web Ontology Language (OWL) are used to model ontologies in order to represent the explicitly semantic of terms in vocabularies and the relationships between those terms on the Web. Basically, modeling various descriptive features, such as qualities, attributes, or modifiers, is a frequent requirement while constructing ontologies. For example, descriptive features are often be modeled as properties in OWL with specify ranges which define the constrains on the values. However, there are many restrictions and limitations on modeling property values while using OWL-DL or OWL-Lite, especially on handling time and value. Such properties need to be handled specifically. Some approaches, such as [12, 24], are proposed to solve problems relevant modeling time and value. The recently research [12] introduces CaTTS, the Calendar and Time Type System, which is based on predefined date and time

Table 2.1: Comparison between WSMO and OWL-S

Aspect	WSMO	OWL-S
Purpose	Focused goal, specific application domains	Wide goal, does not focus on concrete application domains
Coupling	Loose coupling, independent definition of description elements	Tighter coupling in several aspects
Requester needs and service capabilities	Two different points of view, modeled independently and linked through Mediators	Not separated, unified view in the service profile
Functional description	Explicit and complete description	Does not describe some aspects of the functionality
Mediation	Scalable mediation between loosely coupled elements	No mediation
Languages	F-Logic for logical expressions. Ontology language not imposed	Language for condition not defined. Ontology language OWL

types after the Gregorian calendar in XML Schema. CaTTS provides a generic Semantic Web application with methods to model and reason about time and calendar constraints. It comprises two languages, a *type definition language*, CaTTS-DL, for specifying the type of the calendars and a *constraint language*, CaTTS-CL, for parsing the constraints of languages. CaTTS contains a tool called *static type checking* for program analysis, which verifies the behavior of programs and/or systems under specific specifications. Another tool called *constraint solver* is used to annotate on arbitrary finite domains with calendars defined using CaTTS-DL. The W3C Working Group proposed two guidelines for modeling time and value. They are respectively *Value Partitions* and *Value Sets* in [50].

- **Value Partitions:** The Value Partitions considers the feature as an individual/instance. We introduce a scenario that describes the health status of a person is presented as an example. The Value Partitions takes the values of the health status as sets of individual. It defines the class `Health_Value` and its corresponding enumeration of the individual `good_health`, `medium_health`, and `poor_health`. The pattern is a simple and intuitive approach, but it has some limitations: it is impos-

sible to further subpartition the values because OWL only supports a dichotomy, i.e., there are only equalities or differences between individuals. Individuals with partial overlaps are not considered. Therefore, alternative partitions of the same feature space cannot be represented.

- **Value Sets:** The Value Sets considers the feature as a class representing a continuous space that is partitioned by the values in the collection of values. We take the same scenario to illustrate Value Sets approach. The Value Sets describes the health status as subclass partitions, `Poor_health_value`, `Medium_health_value`, and `Good_health_value`, of the `Health_value` feature class. Although this method is more complex, it provides more flexibility than the previous approach. The subclass can be made into further subpartition such that there can be several alternative partitions of the same feature space. The choice between these two patterns depends on the future maintenance and the expansibility of the ontology.

2.3 Service Matching and Ranking

As we mentioned above, service matching is the process that takes user requirement as an input and returns all qualified results. Since there may be multiple results according to different matching degrees, a ranking algorithm, which can be helpful for user to choose among these results, is essential. In this section, we will review several service matching frameworks first. Each of them employs a service description language along with a matching algorithm to perform service matchmaking tasks. Different definitions of matching degree are given according to the relations between a service requirement and an advertisement. Then we review related works on service ranking, which gives priority to several matching results.

2.3.1 Service Matching

Sycara *et al.* [56] define an agent capability description language called LARKS (**L**anguage for **A**vertisement and **R**equst for **K**nowledge **S**haring), which can be used to specify an advertisement, request, and matching agent capabilities. They define three types of matching in LARKS: *exact match*, *plug-in match*, and *relaxed match*. In addition, five dif-

ferent filters in LARKS are provided to carry out the matching process including context matching, profile comparison, similarity matching, signature matching, and constraint matching. These filters spans form text matching to semantic matching. All filters are independent and each of them narrows the set of matched candidates and different degrees of partial matching can result form using different combinations of filters. They propose a well-formed framework for service matchmaking. However, in this framework, no ranking function is provided except for relaxed match, which is determined by a numerical semantic distance value.

Paolucci *et al.* [48] propose a service matchmaking approach based on DAML-S. They make use of Service Profile section of DAML-S to describe the input, output, precondition and effect (IOPE) of a service. Their matching algorithm consists of matching all the outputs of the request against the outputs of the advertisement; and matching all the inputs of the advertisement against the inputs of the request. They compare the outputs first and use input matching only when there is an equal degree of match between outputs. They define four matching degrees as follows:

- **Exact:** For brevity, we use $outR$ to represent one of request's output and $outA$ to represent one of advertisement's output. There are two situations that the match will be labeled as EXACT. The first case is when $outR$ and $outA$ are equivalent, which is intuitive. The second case is when $outR$ is a subclass of $outA$, then they still mark the result as EXACT.
- **Plug-In:** If $outA$ subsumes $outR$, that is, $outA$ is a set that includes $outR$.
- **Subsume:** If $outR$ subsumes $outA$. This happens when the provider dose not completely fulfill the request.
- **Fail:** Failure occurs when no subsumption relation between $outR$ and $outA$ can be identified.

They also propose an architecture to apply their matching algorithm to incorporate with UDDI servers to equip UDDI registries with an additional semantic layer that performs a capability based matching.

Li *et al.* [37] design and implement a service matchmaking prototype system which combines a DAML-S based ontology and a Description Logics reasoner. They extend the matching degrees in [48] and propose a five-level matching degree:

- **Exact:** If advertisement A and request R are equivalent concepts, we call the match Exact; formally, $A \equiv R$.
- **PlugIn:** If request R is sub-concept of advertisement A , we call the match PlugIn; formally, $R \sqsubseteq A$.
- **Subsume:** If request R is super-concept of advertisement A , we call the match Subsume; formally, $A \sqsubseteq R$.
- **Intersection:** If the intersection of advertisement A and request R is satisfiable, we call the match Intersection; formally, $\neg(A \sqcap R \sqsubseteq \perp)$.
- **Disjoint:** Otherwise, we call the match Disjoint; that is, $A \sqcap R \sqsubseteq \perp$.

Paolucci and Li both think that **PlugIn** match is better than **Subsume** match because, under PlugIn match, the output can be used to substitute what the requester expects; while, under Subsume match, the requirement of the requester can only be partially fulfilled. However, we have different consideration. **PlugIn** match may suffer from the problem that an advertisement is too generic: a service provider may define his advertisement as general as possible to maximize the likelihood of being matched. On the other hand, the **Subsume** match works under the assumption that a requester may define his requirement with a general sense and can be satisfied with a specified kind of matched services. We think such a viewpoint would be better in order to fulfill the user's requirement. Therefore, in our approach we discard **PlugIn** match and reserve **Subsume** match.

2.3.2 Service Ranking

Stojanovic *et al.* [55] propose an approach to query results ranking in the Semantic Web. The rationale behind their ranking scheme is to score services by counting available fillers of properties. The more available fillers of a certain property, the lower score it will obtain.

If two property are connected by a and-connector, the obtained score is the sum of these two properties' scores. If two property are connected by an or-connector, the obtained score is the product of scores of these two properties. In short, they translate query results from a set of *concept instances* to a set of *returned relation instances* and compute the relevance value to rank query results.

Di Noia *et al.* [20] and [45] propose a logical approach based on "CLASSIC Description Logics" to support supply-demand matching. In their approach, both supply and demand are described as a conjunction of concepts. Their approach provide three types of match:

- **Exact match:** all requested characteristics are available in the description examined
- **Potential match:** some part of the request is not specified in the description examined
- **Partial match:** some part of the request is in conflict with the description examined

For potential match and partial match, a rank function is devised respectively. The main idea behind their rank function is to compare concept names between supply and demand. The algorithm computes a distance between concepts. The distance starts with an initial value 0, which means best ranking. The value gains with the syntactical difference between concept names and can be used for a measurement of ranking.

Base on [20], Di Noia, *et al.* present an extended semantic-based matchmaking algorithm [19]. They adopt two non-standard inference services in DL, *Concept Abduction* and *Concept Contraction*. In Concept Contraction, the user's requirement is divided into two parts, \mathcal{NG} and \mathcal{ST} . \mathcal{NG} stands for the part which is negotiable; while \mathcal{ST} represents for the part which should be strictly enforced. When potential match is unreachable, the algorithm utilizes Concept Contraction, which relaxes the negotiable part of requirement, to gain satisfiable result. Then, Concept Abduction computes the part of advertisement which should be refined to make requirement and advertisement complete satisfiable with each other. By Concept Abduction and Concept Contraction, the service matching scheme becomes more flexible under reasonable computational complexity.

2.4 Web Services Composition

The composition of Web Services can be considered from two aspects. The first is how component in a Web Services architecture cooperate with each other. We take this perspective as a role of the coordinator in the Web Services system. The coordinator manages the interaction between components of system. The other aspect is a process manager that combines many services that performs a complicated task. The composition of service specifies which operations need to be invoked and in what order. This aspect is worthy of research in Web Services based on Semantics therefore and is the focus of this thesis.

The composition of Web Services used to form new, aggregate services for completing more complicated tasks is the most important part in the interaction and inter-operation of Web services. How to compose and coordinate different services, and assemble them to support more complex services and goals, are major challenges. Service composition should consider the global constraints of the services involved in the composition, as well as the sequence of the services. The composition based on the logics of automation of Web Services is discussed in detail in [54, 17, 33].

The composition and execution of services should be considered simultaneously because the result of the composition may influence the method of execution. [49, 58] propose a technique for automated synthesis of new composite Web Services from a set of abstract BPEL4WS descriptions of component services to executable BPEL4WS processes automatically.

2.5 Related Projects

2.5.1 SATINE Project

The SATINE Project [22], which is applied on tourism industry, is dominated by Software R&D Center in Middle East Technical University in Turkey. Since activities of travel domain involves Business to Business (B2B), such as the relationship between travel agents and airline company or other partners, and also involves Business to Customer (B2C), means if a person wants to plan his/her trip spontaneously, he/she have to deal with a lot of services distributed in the network. If machines can search advertisements and book

their orders automatically, it would be convenient for those travelers. SATINE project is proposed to fulfill such requirements in travel domain and is motivated by Open Travel Alliance (OTA), which produced the XML schemas of message specifications to be exchanged between trading partners. The architecture of SATINE project provides secure and semantic-based interoperability framework for Web Services platform in peer-to-peer networks, and provides tools and mechanisms for publishing, discovering, composing, and invoking correlative Web Services. A trip plan comprises many sub-plans and related activities which invoke many Web services to complete a composed trip requirement. The creation of complex services for orchestrating many simple Web Services is an important task in travel business, because the execution order of Web Services may be very complicated, vary execution order may result different consequence. Therefore SATINE project also developed the framework of Semantic Web Services composition and execution to provide complex services. In SATINE project, a Semantic Wrapper for constructing and describing Web Services is proposed. That component is to wrap existing information resources and provide an easy tool for small/middle enterprises to collect and annotate Web Services conveniently. A Semantic Wrapper consists of two tools: the Web Services Creator and the Web Services Annotator. The Web Services Creator transforms the existing resources of Web Services and the Web Services Annotator describes a Web services with using OWL-S as semantic descriptions.

2.5.2 European Semantic Systems Initiative (ESSI)

The European Semantic Systems Initiative (ESSI) Cluster², which combines Web Services and Semantically empowered system solutions with semantically service-oriented architectures, is made of six European 6th Commission Framework Projects that works on European research and industry through world-wide standardisation. The ESSI Cluster research projects are listed below:

- **Adaptive Services Grid (ASG)**

ASG provides an architecture that aims to build a bridge between business-related requirements and current service-oriented IT-infrastructures for eliminating the gap of communication, protocol, and standard.

²<http://www.essi-cluster.org/>

- **Data Information and Process Integration with Semantic Web Services (DIP)**

DIP focuses on further development, combination, and enhancement of Semantic Web and Web Services technologies for producing a new infrastructure of Semantic Web Services that will provide data and process integration in eWork and eComerce.

- **Knowledge Web**

Knowledge Web promotes greater awareness and faster take-up of Semantic Web technology with the research activity for extending the capabilities of Semantic Web to help reduce time which is needed to transfer the technology to industry.

- **Semantically-Enabled Knowledge Technologies (SEKT)**

SEKT is an integrated project by combining the three core research areas, that is, ontology management, machine learning, and natural language processing.

- **Semantics Utilised for Process management within and between Enterprises (SUPER)**

SUPER raises Business Process Management (BPM) to the business level from the IT level of semantics of business experts. It focuses on managing the execution from a Business expert's view rather than from a technical perspective.

- **Triple Space Communication (TripCom)**

TripCom shares with the project of ESSI the approach to add machine-readable and machine-understanding descriptions to data and processes. It adds a new communication channel to existing efforts that is not covered by current Web Services technology by providing instant publication in distributed information system with Semantic Web Services.

Each project with specialized aspects about Semantic Web Services contributes on building the infrastructure, developing Semantic Web-based knowledge technologies, enriching existing Web Services with semantic description and supporting the transition mechanism of Ontology technology from the Academia to the Industry.

2.5.3 EON Architecture

The EON project, section on Stanford Medical Informatics, want to create an architecture consists of a set of software components and interfaces that provides developers a concrete approach to construct robust decision-support systems that based on ontology reasoning about guideline-directed care. They implemented the EON architecture by building three main components, such as the temporal database mediator for handling requests of time-dependent data from a patient database, the generic and extensible ontology for modeling clinical guidelines and protocols, a protocol-based therapy planner, and a mediator for explaining and visualizing the behavior of other EON components. Recent years, a practical DSS project called ATHENA uses the EON architecture for developing guideline-based decision-support systems . An application based on EON architecture may contain these components described below:

- **EON Problem-Solving Modules(Guideline Interpreter)** The EON Problem-Solving Modules consists of many submodules. All problem-solving modules access a guideline knowledge base consisting of models of clinical guidelines, patient data, and medical concepts created in and accessed through the Protégé, an ontology management system. There is a module called EON Guideline Interpreter is responsible for taking inputs such like standard clinical guideline description from clients' queries and relevant patient data from patient database then generating an output situation-specific recommendations.
- **ChronusII Temporal Mediator** a temporal database mediator, also called ChronusII, has been developed that serves as the conduit between the problem-solving modules and the clinical database which stores significant amount of temporal information, such as when a specimen for a laboratory test is obtained and when a prescription is written and filled. The ChronusII extends the standard relational model and the SQL query language to support temporal queries and provides an expressive general-purpose temporal query language that is tuned to the querying requirements of clinical decision-support systems.
- **EON Knowledge Base** EON knowledge base includes the EON Guideline Model,

which consists of a set of classes and attributes that describe concepts and relations with which the content of clinical guidelines are formalized The Medical-Concept Model, which defines the particular clinical interventions that are typical for a given area of medicine, and the types of patient findings and patient problems that are most commonly reported in a given medical discipline, and The Patient Data Model, which defines the classes and attributes of patient information required by the rest of the system. The EON knowledge base is manipulated through the Protégé editor.

Each project with specialized aspects about Semantic Web Services contributes on building the infrastructure, developing Semantic Web-based knowledge technologies, enriching existing Web Services with semantic description and supporting the transition mechanism of Ontology technology from the Academia to the Industry.



Chapter 3

Preliminaries

From the previous chapter, we not only review recently researches which focus on establishing related standards and ideas about Web Services and Semantic Web but also many developing projects which brought academic theories into practical domains. The Semantic Web enhances the usability and extensibility of Web Services life-cycle on automatically publishing services, discovering services, and invoking services because they have defined precise specifications of service descriptions and protocols. Furthermore, the Knowledge is represented by ontology, which is modeled to express the fact in the certain domain, is a fundamental task in Semantic Web Services environment. It would be helpful for us to understand the academic theories behind this mechanism. Therefore we will introduce some preliminaries of our thesis in this chapter, such as Description Logics (DL) [9] for knowledge representation and describing concepts and properties is introduced. In this section, syntax, semantics, and reasoning feature of DLs is represented briefly. After that, Web Ontology Language (OWL), which is the communication language between service components, and Semantic Web Rule Language (SWRL), which defines rules of relationship between Web Services semantically, is represented. In the end, Web Service Description Language(WSDL) and Web Services - Business Process Execution Language (WS-BPEL) are introduced in detail.

3.1 Description Logics

Description Logics (DL) [9] are a well-known family of knowledge representation formalisms which represent the knowledge of a domain. The essential elements of DLs are concepts (unary predicates, classes) and roles (binary relations). Complex concepts can

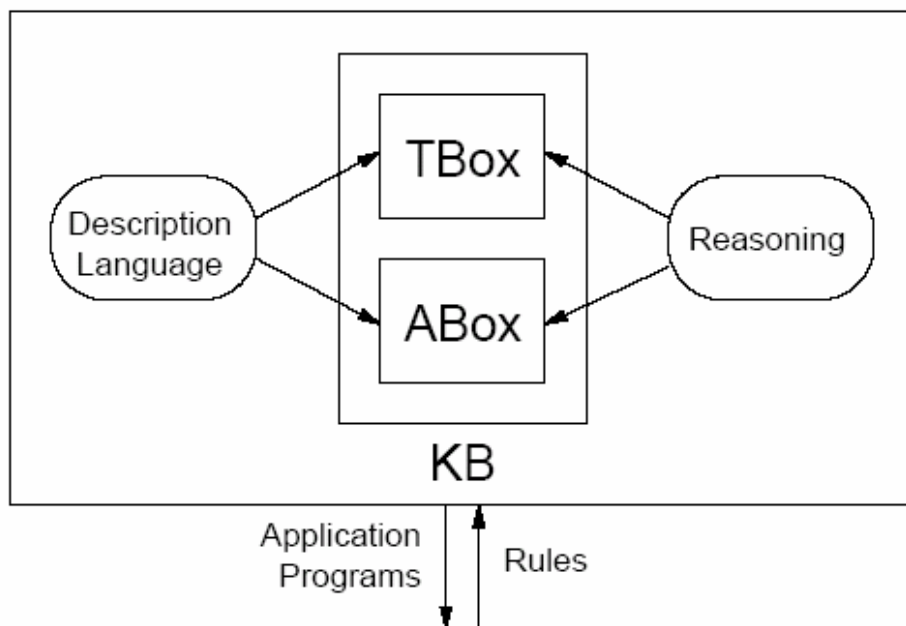


Figure 3.1: Architecture of a knowledge representation system based on Description Logics.

be defined by assembling atomic ones. With formal and logic-based semantics, reasoning is an important feature of Description Logics which allows inferring implicit knowledge from explicit knowledge stored in the knowledge base. In addition, the DL reasoner can check whether two concepts subsume each other (classify taxonomy). Figure 3.1 shows the architecture of a knowledge representation system based on Description Logics.

The knowledge representation system consists of a TBox and an ABox. In the TBox, it first defines the concepts in the application domain (the terminology used in the world) and then utilizes these concepts to define roles (binary relationship between concepts). Along with concepts and roles, TBox contains a set of axioms that are used for asserting relationship among concepts and roles. A concept can be viewed as a set of objects, which are instances of a certain object class. Therefore, in the ABox, we can assert that a certain instance (denoted as individual) belongs to the given concept or two individual have realized the relationship of a certain role. TBox supports reasoning service for checking subsumption and satisfiability among concepts, while ABox support reasoning services, such as consistency and instance checking. Both of them are described in Description Language, such as \mathcal{AL} , or the other extensive language. Another feature of the

Description Logics knowledge representation system is the emphasis on reasoning as its core services. Application programs and rules can interact with the systems in various way. Other applications can interact with the system by querying the Knowledge Base ,and by modifying it by adding and retracting concepts, roles, and assertions. Rules, which are extensional formalisms that enrich the knowledge base, are another way to access the DLs knowledge architecture. In the following sections, we list Description Logics notations, such as concept definitions and roles to represent ontology definitions. Detailed theoretical explanation can be referred in [9].

3.1.1 Description Logics Syntax and Semantics

Description Logics Syntax *Atomic concepts*, which are sets of unary predicate symbols that are used to denote, and *atomic roles*, which are sets of binary predicate symbols that are used to denote, are basic description elements of Description Logics. Complex descriptions can be built from them inductively with *concept constructors* and *role constructors*. In the following abstract notations, the capital letter A stands for an atomic concept, the capital letter R stands for atomic roles, and the capital letter C and D represent concept descriptions. The language \mathcal{AL} is a minimal and fundamental language that contains smallest set of concept constructors. Concept description in \mathcal{AL} are defined using the following syntax rules [9]:

$C, D \longrightarrow A$		(atomic concept)
\top		(universal concept)
\perp		(bottom concept)
$\neg A$		(atomic negation)
$C \sqcap D$		(intersection)
$\forall R.C$		(value restriction)
$\exists R.\top$		(limited existential quantification)

We present an example to illustrate the expressive of \mathcal{AL} : suppose that **Person** and **Female** are atomic concepts, then we can define **Person** \sqcap **Female** to represent *Woman*, which means a person who is female. Then **Person** \sqcap \neg **Female** represents *Man*, which means a person who is not a female. In addition, if *hasChild* is an atomic role, we can say

that $\text{Person} \sqcap \exists \text{hasChild}.\top$ express a person who has at least a child. Another example $\text{Person} \sqcap \exists \text{hasChild}.\text{Person} \sqcap \text{Female}$ expresses a person whose children are all female.

However, in practical applications and business domains, the \mathcal{AL} languages are not enough for describing cardinality constraints on roles, concrete domains, transitive roles, inverse roles, role hierarchies and so on. The extended language, such as \mathcal{ALC} which supports full concept negation(\mathcal{C}), based on \mathcal{AL} with more expressive power is proposed. A language that supports number restriction(\mathcal{N}) is named \mathcal{ALCN} while all member of the \mathcal{AL} -family include \mathcal{AL} as a sublanguage. Therefore, we use constructors, $\mathcal{SHIQ}(\mathcal{D})$, to describe our application domain. These added constructors include:

$$\begin{array}{llll}
=_{n} & \geq_{n} & \leq_{n} & \text{(concrete domain exactly/min/max restriction)} \\
=_{n} R & \geq_{n} R & \leq_{n} R & \text{(unqualified cardinality exact/atleast/atmost restriction)} \\
=_{n} R.C & \geq_{n} R.C & \leq_{n} R.C & \text{(qualified cardinality exact/atleast/atmost restriction)} \\
\neg C & & & \text{(arbitrary concept negation)}
\end{array}$$

To follow up the example presented before, we show some examples of the constructors. If $\text{Woman} \equiv \text{Person} \sqcap \text{Female}$, we can define **Mother** concept by $\text{Woman} \sqcap \exists \text{hasChild}.\text{Person}$. Here we can see that **Woman** and **Person** are arbitrary concept name. $\neg \text{Woman}$ means a set of individuals that are not women. Moreover, we can use $\text{Mother} \sqcap \geq 2 \text{hasChild}$ and $\text{Mother} \sqcap \geq 2 \text{hasChild}.\text{Female}$ to represent “a mother has 2 children” and “a mother has 2 daughters”.

Description Logics Semantics Here we summarize the Description Logics syntax and its semantics in the Table 3.1. The language description for each constructor is listed in column **Symbol**.

Table 3.1: Description Logics Concept Constructors

Name	Syntax	Semantic	Symbol
Top	\top	$\Delta^{\mathcal{I}}$	\mathcal{AL}
Bottom	\perp	\emptyset	\mathcal{AL}
Intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	\mathcal{AL}
Union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	\mathcal{U}

Name	Syntax	Semantic	Symbol
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	\mathcal{C}
Value restriction	$\forall R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$	\mathcal{AL}
Existential quant.	$\exists R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$	\mathcal{E}
Unqualified number restriction	$\geq nR$ $\leq nR$ $= nR$	$\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\} \mid \geq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\} \mid \leq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\} \mid = n\}$	\mathcal{N}
Qualified number restriction	$\geq nR.C$ $\leq nR.C$ $= nR.C$	$\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \mid \geq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \mid \leq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \mid = n\}$	\mathcal{Q}

Description Logics Terminologies *Terminologies* are statements describing the relationship between concepts and concepts or between roles and roles. The terminological axioms are represented in the following two forms of

- **Inclusion Axioms** $C \sqsubseteq D$ ($R \sqsubseteq S$)
- **Equality Axioms** $C \equiv D$ ($R \equiv S$)

where C, D are concept names (and R, S are role names). The first axiom is called *inclusion*, while the second one is called *equalities*. The relative semantics to *inclusion* and *equalities* are defined as follows: an interpretation \mathcal{I} satisfies $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and it satisfies $C \equiv D$ if $C^{\mathcal{I}} \equiv D^{\mathcal{I}}$. All the terminological (TBox) and assertional (ABox) axioms are listed in Table 3.2.

Table 3.2: Terminological and Assertional Axioms

Name	Syntax	Semantics
Concept Inclusion	$C \subseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Role Inclusion	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
Concept Equality	$C \equiv D$	$C^{\mathcal{I}} \equiv D^{\mathcal{I}}$
Role Equality	$R \equiv S$	$R^{\mathcal{I}} \equiv S^{\mathcal{I}}$
Concept Assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
Role Assertion	$R(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$

3.2 OWL

The Web Ontology Language(OWL) [18, 42] is a semantic markup language which is defined to describe and construct web ontologies. OWL supports greater machine interpretability of Web content. It is derived from the DAML+OIL Ontology language and is the recommended and standard ontology language in W3C. For the implementation of Semantic Web, OWL is based on XML and RDF and has ability to represent machine interpretable and understandable content on Web by providing additional vocabulary along with a formal semantics. It is written in XML document and defines its own syntax as a vocabulary extension of RDF. An OWL ontology includes descriptions of class, properties, and their individuals. Based on the logic-based semantics theory, OWL supports ontology reasoning.

The OWL currently provides three kinds of sublanguages: OWL-Lite, OWL-DL, and OWL-Full. OWL-Lite supports those users primarily needing a classification hierarchy and simple constraints. OWL-DL supports those users who want the maximum expressiveness while retaining computational completeness, which means all conclusions are guaranteed to be computable, and decidability, which means all computations will finish in finite time. OWL-DL includes all OWL language constructs but they can be used only under certain restrictions. OWL-DL is based on Description Logics to form the formal foundation of OWL. OWL-Full is defined for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. Since OWL-Full retain the most expressiveness and freedom on represent OWL, there are no reasoning software supporting complete reasoning for all features of OWL-Full.

These sublanguages are an extension of its particular simpler predecessor, both in what can be legally expressed and in what can be validly concluded. Such relationships between subsets are in the following sets which are not reversibly:

- Every legal OWL Lite ontology is a legal OWL DL ontology.
- Every legal OWL DL ontology is a legal OWL Full ontology.
- Every valid OWL Lite conclusion is a valid OWL DL conclusion.
- Every valid OWL DL conclusion is a valid OWL Full conclusion.

Ontology developers consider which OWL subset they adopt depending on their needs. The choice between OWL-Lite and OWL-DL is decided by the extent to which users require the more-expressive constructs provided by OWL-DL, while the choice between OWL-DL and OWL-Full depends on the extent to which users require the meta-modeling facilities of RDF Schema. Reasoning support is less predictable when using OWL-Full since complete OWL-Full implementations do not currently exist.

3.3 Semantic Web Rule Language: SWRL

Semantic Web Rule Language (SWRL) [26] is a rule language, which combines OWL and RuleML. It extends OWL axioms to support Horn-like rules. Rules are of the form of an implication between an antecedent (body) and consequent (head), which means whenever the conditions specified in antecedent holds, the conditions in consequent must also hold. Besides, an empty antecedent is treated as trivially true, while an empty consequent is treated as trivially false. For example, we can define a rule which asserts that the composition of two roles *hasParent* and *hasBrother* is *hasUncle*. This rule can be represented as the following human readable form:

$$parent(?x,?y) \wedge brother(?y,?z) \longrightarrow uncle(?x,?z)$$

where *?x*, *?y*, and *?z* stand for variables corresponding to individuals in ABox. The overall abstract syntax of SWRL are listed in Table 3.3

3.3.1 SWRL Editor

To integrate rules with ontologies, a SWRL rule editor called SWRLTab has been embedded into Protégé, which is the most popular OWL-ontology editor. SWRLTab works as a plug-in of Protégé. It provides a friendly graphical user interface to edit SWRL rules. Figure 3.2 shows the editing environment of the SWRL editor in Protégé.

The SWRLTab should be visible for all OWL knowledge bases that import the SWRL ontology ¹. The rules are stored as concepts within the same ontology where classes and properties are defined. Therefore, classes and properties defined in an ontology can be used directly by SWRL rules. Figure 3.3 shows the relationship between SWRL rules and

¹<http://www.daml.org/rules/proposal/swrl.owl>

Table 3.3: Abstract Syntax of SWRL

axiom	::= rule
rule	::= 'Implies('annotation antecedent consequent')'
antecedent	::= 'Antecedent('atom')'
consequent	::= 'Consequent('atom')'
atom	::= description '('i-object')' individualvaluedPropertyID '('i-object i-object')' datavaluedPropertyID '('i-object d-object')' sameAs '('i-object i-object')' differentFrom '('i-object i-object')'
i-object	::= i-variable individualID
d-object	::= d-variable dataLiteral
i-variable	::= 'I-variable('URIreference')'
d-variable	::= 'D-variable('URIreference')'

ontologies. Rules can be divided into two parts: head and body. Each part consists of zero or multiple atoms, which can be ClassAtom, IndividualPropertyAtom or others. Multiple atoms are treated as a conjunction. ClassAtom can be viewed as an unary predicate and the predicate name refer to a class name in an ontology; whereas IndividualPropertyAtom can be viewed as a binary predicate and the predicate name refer to an property name in the same ontology.

3.4 Quantitative Relations

In the previous section, we've introduce the Web Ontology Language (OWL) and Semantic Web Rule Language (SWRL). Although, OWL has been a standardize language to express ontology in the Semantic Web, it still have some drawbacks considering the support of concrete domain. Pan et al. [47] pointed out some limitations of current version OWL.

- It does not support user-defined XML Schema datatypes:e.g., $>_{15}$
- It does not support negated datatypes:e.g., $\neg >_{15}$
- Enumerated datatypes are the only user-defined datatypes supported by OWL.

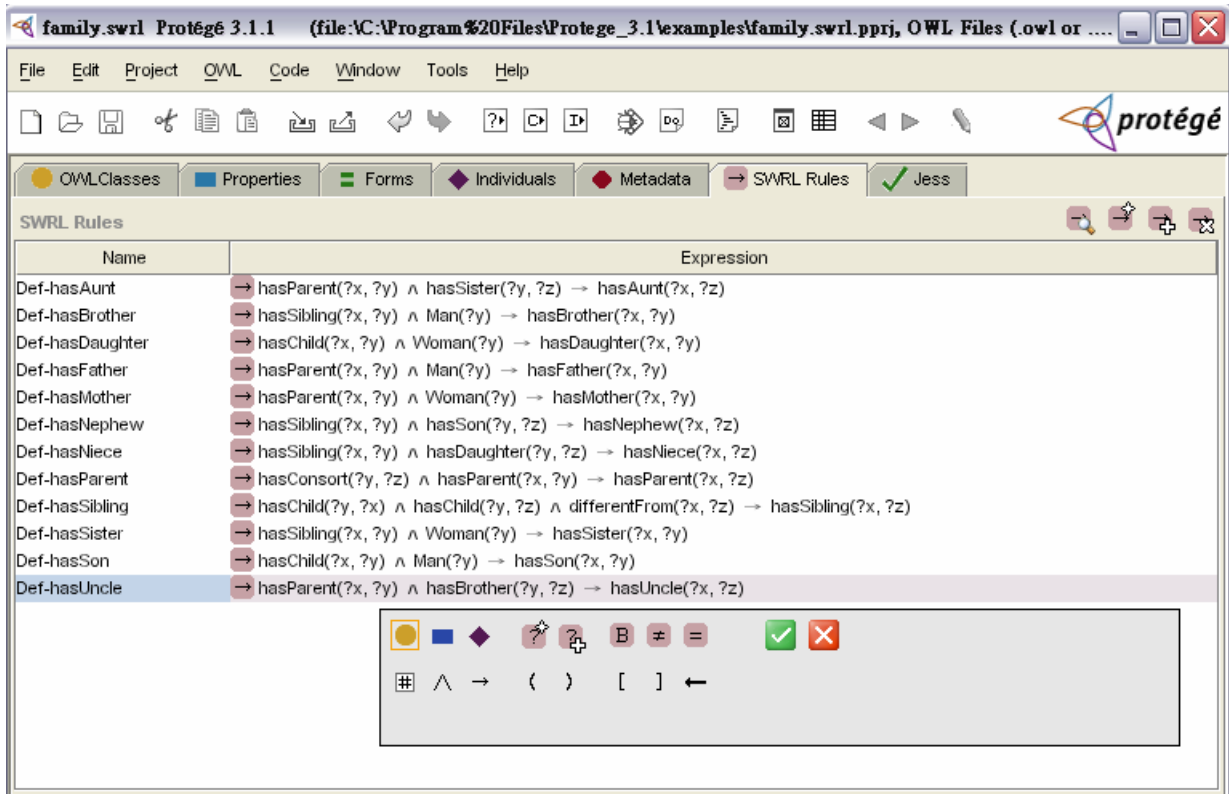


Figure 3.2: The SWRL Editor in Protégé

- We cannot name the enumerated datatypes in OWL.
- There is no n-ary datatype predicates:e.g., $+$.
- There is no user-defined datatype predicates:e.g., *sumNoLargerThan15*

From the above, we can see that the current version of OWL lacks support for the modeling for quantitative relations. For example, it is difficult for us to express the range of the trip budget in the form of concept expression. Lu [39] proposed an approach for modeling quantitative relations. The main idea is to transfer a linear inequality problem into concept subsumption checking. He proposed two methods to deal with such kind of problem.

In the first method, concrete values and intervals are modeled with four kinds of concepts. Each concept represents a range with an upper/lower bound and relations among these ranges can be defined with concept subsumptions. The semantics of these four kinds of concepts are defined by the following table.

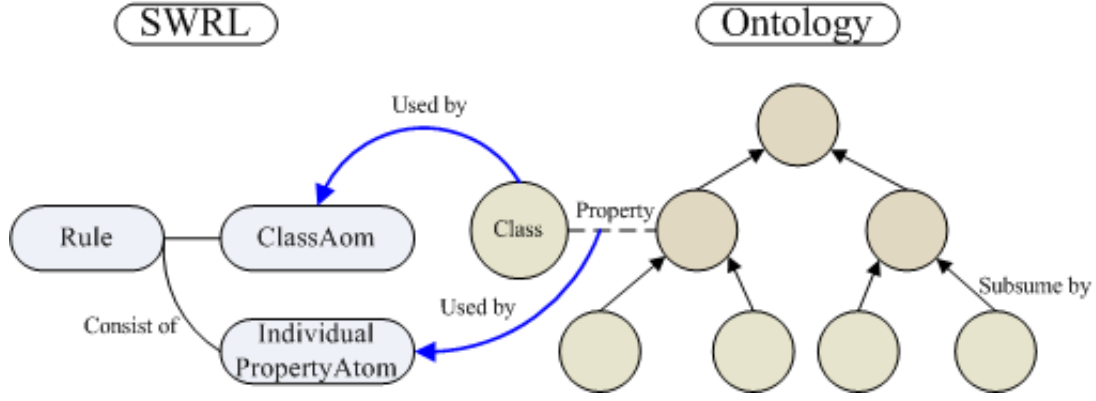


Figure 3.3: Relationship between SWRL and Ontology

Table 3.4: Semantics of Quantitative Concepts

Concept	Semantics
$leq\mathcal{X}$	$(-\infty, \mathcal{X}]$
$lessThan\mathcal{X}$	$(-\infty, \mathcal{X})$
$geq\mathcal{X}$	$[\mathcal{X}, \infty)$
$greaterThan\mathcal{X}$	(\mathcal{X}, ∞)

Therefore, given a concrete value \mathcal{X} , we have the following four types of concepts:

- $leq\mathcal{X}$ represents " \mathcal{X} and all concrete values less than \mathcal{X} "
- $lessThan\mathcal{X}$ represents " all concrete values less than \mathcal{X} "
- $geq\mathcal{X}$ represents " \mathcal{X} and all concrete values greater than \mathcal{X} "
- $greaterThan\mathcal{X}$ represents " all concrete values greater than \mathcal{X} "

Given two concrete values \mathcal{X} and \mathcal{Y} where $\mathcal{X} < \mathcal{Y}$, we can define relations among these four kind of concepts as follows:

- $lessThan\mathcal{Y} \sqsubseteq leq\mathcal{Y}$
- $greaterThan\mathcal{Y} \sqsubseteq geq\mathcal{Y}$
- $leq\mathcal{X} \sqsubseteq lessThan\mathcal{Y}$
- $lessThan\mathcal{X} \sqsubseteq leq\mathcal{Y}$
- $geq\mathcal{Y} \sqsubseteq greaterThan\mathcal{X}$
- $greaterThan\mathcal{Y} \sqsubseteq geq\mathcal{X}$

Based on these concepts and subsumption relationships, we can express all concrete values between \mathcal{X} and \mathcal{Y} with "*lessThan* $\mathcal{X} \sqcap$ *greaterThan* \mathcal{Y} ". A single point of concrete value is a special case of interval whose upper bound and lower bound are the same. A concrete value \mathcal{X} can be represented with "*leq* $\mathcal{X} \sqcap$ *geq* \mathcal{X} ". Such a method models concrete values and intervals well, but it cannot define ordinal relations between intervals. Therefore, Lu proposed a second method.

The second method is used for modeling temporal ordinal relations. Given a rational number line, we can divide it into equal length ranges called primitive intervals. Temporal relations between primitive intervals are defined as

- $t_1 < t_2$ means t_1 is *before* t_2
- $t_1 \leq t_2$ means t_1 is *before* or the *same* as t_2
- $t_1 > t_2$ means t_1 is *after* t_2
- $t_1 \geq t_2$ means t_2 is *after* or the *same* as t_2

Complex intervals consist of primitive intervals. A complex interval T is an interval that starts from one primitive interval (written as *begin*(T)) and ends at another primitive interval (written as *end*(T)). Temporal relations between two arbitrary intervals T_1 and T_2 can be defined as

- $T_1 < T_2$ means $end(T_1) < begin(T_2)$
- $T_1 > T_2$ means $begin(T_1) > end(T_2)$
- $T_1 \leq T_2$ means $(end(T_1) \leq end(T_2))$ and $(begin(T_1) \leq end(T_2))$
- $T_1 \geq T_2$ means $(begin(T_1) \geq begin(T_2))$ and $(begin(T_1) \geq end(T_2))$

In this method, an interval T are modeled as a concept (written as $C(T)$) in TBox. If T_2 is a sub-interval of T_1 , it means that $C(T_2)$ is a sub-concept of $C(T_1)$. Then, he defines two transitive object property *ends_before* and *begins_after* to express the above temporal ordinal relations. Their semantics are shown in the following table.

Note that, in the concept hierarchy, if we say an interval $C(T_1)$ ends before $C(T_2)$ (represented as $C(T_1) \sqsubseteq ends_before.C(T_2)$) and does not overlap $C(T_2)$ (represented as $C(T_1) \sqcap C(T_2) \sqsubseteq \perp$), $C(T_1)$ also ends before all sub-concept of $C(T_2)$. Temporal ordinal relations between two intervals can be checked by the following rules.

Table 3.5: Semantics of Quantitative Relations

Concept Expression	Semantics
$\exists ends_before.C(T)$	$(-\infty, end(T))$
$\exists begins_after.C(T)$	$(begin(T), \infty)$
$C(T_1) \sqsubseteq \exists ends_before.C(T_2)$	$end(T_1) < end(T_2)$
$C(T_1) \sqsubseteq \exists begins_after.C(T_2)$	$begin(T_1) > begin(T_2)$

$T_1 < T_2$ iff $(C(T_1) \sqcap C(T_2) \sqsubseteq \perp)$ and $(C(T_1) \sqsubseteq \exists ends_before.C(T_2))$ are both true.

iff $(C(T_1) \sqsubseteq (\neg C(T_2) \sqcap \exists ends_before.C(T_2)))$ is true.

$T_1 \leq T_2$ iff $(C(T_1) \equiv C(T_2))$ is true

or $(C(T_1) \sqsubseteq \exists ends_before.C(T_2))$ and $(C(T_1) \sqsubseteq \exists begins_after.C(T_2))$ are both true.

or $(C(T_1) \not\sqsubseteq \exists ends_before.C(T_2))$ and $(C(T_2) \not\sqsubseteq \exists ends_before.C(T_1))$ and $(C(T_2) \sqsubseteq \exists begins_after.C(T_1))$ are all true.

or $(C(T_1) \not\sqsubseteq \exists begins_after.C(T_2))$ and $(C(T_2) \not\sqsubseteq \exists begins_after.C(T_1))$ and $(C(T_1) \sqsubseteq \exists ends_before.C(T_2))$ are all true.

Lu's approach models quantitative relations using concepts and object properties. It has several flaws. For example, in the first method, for every concrete value, it will create four concepts: *leq*, *lessThan*, *geq*, and *greaterThan*. Second, it does not allow the user to specify intervals that are smaller than primitive intervals. For example, if the primitive interval is set to be one day, then it is impossible for user to describe schedule using hours. On the other hand, SWRL supports build-ins for dealing with quantitative relations. We can use the *less(X, Y)* function provided by SWRL build-in to compare the numeric value or time interval. Such a numeric value or time interval must be XML Schema datatypes. However, SWRL rules can be only applied to individuals and cannot be used in the level of TBox. In our system, we models service descriptions as concept expressions. Therefore, we adopt Lu's approach to model value partition and time ontology.

3.5 Web Service Description Language(WSDL)

The WSDL [3] is an XML based document for describing network services in abstract terms about operating on messages containing concrete data formats and network proto-

col. As communication protocols and message formats are standardized in W3C (World Wide Web Consortium), it becomes increasingly possible and important to be able to describe the communications in structured way. WSDL defines an XML grammar for describing services' specification in order to achieve its needs. Such network connection used to communicate is standardized with SOAP 1.1, HTTP GET/POST, and MIME.

A WSDL document simply specifies:

- *What* the Web Services consists of - (types, message, operation)
- *How* the Web Services is bound to a set of concrete protocol - (binding, port type)
- *Where* the Web Services are implemented - (port)

Above-mentioned specification is defined with some main elements for automating the details involved in application communications applied in distributed systems and services. Details description about these elements is in the following section:

- **Types:** A Type is a container for data type definitions using some type system, such as XSD.
- **Message:** Message is an abstract, typed definition of the data being communicated.
- **Operation:** Operation defines an abstract description of an action supported by the service.
- **Port Type:** Port Type is an abstract set of operations supported by one or more endpoints.
- **Binding:** Binding defines a concrete protocol and data format specification for a particular port type.
- **Port:** A Port means a single endpoint defined as a combination of a binding and a network address.
- **Service:** A Service represents a collection of related endpoints.

However, since WSDL recognizes the need for rich type systems for describing message formats and supports the XML Schemas specification (XSD), it does not support semantic description of services.

3.6 Web Services Business Process Execution Language(WS-BPEL)

The Web Services - Business Process Execution Language (WS-BPEL) [7], which is developed from Business Process Execution Language for Web Services (BPEL4WS) [30, 8] is a language for describing business processes in Web Services. It is based on WSFL and XLANG, provided by IBM and Microsoft respectively. Based on Web Service Description Language (WSDL) 1.1, BPEL-WS is also compatible with other Web Services standards of XML module definitions, XPath, and WS-Addressing. The goals of the BPEL4WS specification are as follows [36]:

- Define business processes that interact with external entities through Web Services operations.
- Define business processes using XML as the basic language.
- Define a set of Web Services orchestration concepts to be used by both external (abstract) and internal (executable) views of a business process.
- Provide both hierarchical and graph-like control strategies.
- Provide functions for the simple manipulation of data needed to define process relevant data and control flow.
- Support an identification mechanism for process instances.
- Support the implicit creation and termination of process instances as the basic lifecycle mechanism.
- Define a long-running transaction model.
- Use Web Services as the model for process decomposition and assembly.
- Build on compatible Web Services standards as much as possible in a composable and modular manner.

WS-BPEL refers to high-level state transition interaction of processes with an Abstract Process, which represents a set of publicly observable behaviors, including information like when to wait for messages, when to send messages, when to compensate for failed transactions, and so on. It also deals with short-lived programmatic behaviors, which are often executed as a single transaction and invoke access to local logic and resources, such as files and databases.

Business Process Execution Language for Web Services (BPEL4WS) is still the most adopted standards for Web Services composition. Essentially, it is comprised of Partner Links, Partner Link Types, Variables, Activities, Correlation Sets, Compensation Handlers, and Fault Handlers for describing business processes. *Partner Links* are Web Services interfaces that facilitate interaction between a business process and partner Web Services. *Partner Link Types* define the roles played by the services using the processes' WSDL. *Variables* define messages sent and received by partners. *Activities* can be divided into two types: *Primitive Activities*, namely assign, invoke, receive, reply, throw, and wait; and *Structure Activities*, including sequence, while, switch, flow and pick. *Correlation Sets* are sets of business data fields that capture the state of an interaction. *Compensation Handlers* are invoked to perform compensation activities. *Fault Handlers* are defined to catch exceptions.

BPEL4WS enables automated Web Services execution and is broadly used for Web Services composition. However, some shortcomings of BPEL4WS that limit its ability to provide flexible interoperability are reported in [40, 59]. BPEL4WS is a process-based language, so that process participants (partners' Web Services) must be defined and bound to the process flow during the design stage. The BPEL standard does not support Semantic Web Services; therefore, partner discovery and binding at run time are not possible. In [13, 32], the authors discuss some solutions for dynamical composition of Web Services execution.

3.7 Web 2.0 Technology

The Web 2.0 technology, as outlined in [46], allows for an easier distributed collaboration. In panel discussion at ISWC 2006, there is an interesting topic, "The Role of Semantic

Web in Web 2.0: Partner or Follower?”. It makes a lot of people start thinking of the possibility of combining Semantic Web and Web 2.0 technology. Including Tim Berners-Lee, the World Wide Web inventor, some researchers believe that these two ideas are complementary rather than competing . The goals of the Semantic Web vision and Web 2.0 are aligned, and each brings its own strengths into the picture. Semantic Web has good inference ability, and Web 2.0 technology bridges the user and Web application with responsive user interface and collaborative mechanism. Web 2.0 is distinguished from classical web technology by various characteristic features described below:

- **Community** Web 2.0 pages allow contributors to collaborate and share information easily. These sites may have an ”Architecture of participation” that encourages the user to add value to the application as they use it. Each contributor gains more from the system than he/she puts into it.
- **Mashups** Mashups combines data from more than one source into a single integrated tool. Certain services from different sites can be pulled together in order to experience the data in a novel and enhanced way. For example, we could embed Google Maps in our personal blog to add location information that enriches the content of the site.
- **Rich User Experience** Web 2.0 sites often feature a rich, user-friendly interface based on AJAX, Flex or similar rich media. Techniques such as AJAX, Adobe Flash, Flex, Java, and Silverlight have improved the user-experience in browser-based applications. These technologies allow the user to request an update for some part of web page’s content, and to alter that part in the browser, without needing to refresh the whole page at the same time.

AJAX is considered as the technological pillar of the Web 2.0 which allows to create responsive user interfaces, and thus facilitated both of the other pillars. For instance, the user likes to use community pages with slick user interfaces, and mashups that incorporate data from different web sites introduced asynchronous communication for more responsive pages.

Chapter 4

Service Composition and Execution Based on Semantic Technology

4.1 Overview of Web Services Composition Architecture Based on Semantic Technology

In the Web Services environment, service providers advertise their services based on UDDI search mechanism so that service requestors can find their suiting services that fit their needs. Due to the limitation of keyword-based search, sometimes it is hard to meet the user' complicated requirements precisely. While Web Services become mature in recent years. The division of labor between Web Services is emerging obviously. Web Services tends to be a composite service which offer value-add and integrated service, such as a travel agent which provides a integrated trip package, including transportation tickets, accommodation reservations, admissions for amusement park, to the customer. Otherwise, discovering and choosing an appropriate provider is usually time-consuming and error-prone. To increase flexibility and ability, [57] proposed "a Web Services composition architecture based on semantic technology" that provides accurate and automated matching.

In our previous architecture we states earlier, we found some drawbacks from casual users' and administrators' perspectives. Such as the high entrance barrier of Description Logics for casual users. And the management of ontology controlled by a small group is not efficient enough. Ontology maintenance need to be more organized so that machines can use those plentiful and correct ontologies in reasoning tasks to deduce the right results. In this chapter, we will review the fundamental design of Web Service composition

architecture and also illustrate new features to create a better environment for the user under semantic-based application.

4.1.1 Web Services Composition

Service composition is a solution to a specific problem and combines different Web Services into a integrated execution process. From service providers' perspective, they publish their services and annotate their services with service description like WSDL, which contains information of how service requestors access the service and what messages format should service requestors adopt. From the requestors' perspective, they would like to discover suitable services to fit their needs and invoke composite service automatically. So we start to discuss the relationship between providers' services and requestors' needs. We denote the things that service providers release are **Advertisements**, and requestors' needs as **Requirements**.

- **Requirements and Advertisements**

In general, a complicated requirement comprises different service attributes defined under **Advertisements**. With Semantic Web technology, Web Services descriptions are annotated by Web Ontology Language(OWL), and are able to be utilized for service matching with subsumption checking and composition. To match **Requirements** and **Advertisements**, we compare **Advertisements**' Web Services descriptions written in OWL with **Requirements** in the ontology. We assume that Web Services descriptions are mapped to ontology in advance.

According to the research [29]. There are some characteristics of **Requirements**:

- **incomplete**: Service requestors usually cannot describe their exact needs. They only mention parts of their requirements. That make a requirement input by the user is often incomplete.
- **ambiguous**: Service requestors often do not state their requirements clearly. They even do not know exactly what they want.
- **incrementally evolving**: The guidelines help the user state their requirements more clearly. With system suggestion, the user is also more involved in defining his/her requirements.

The characteristics of an advertisement are:

- **more complete:** Advertisements should provide detailed information to service requestors.
- **clear:** Advertisements that the user discovers and invokes should not be ambiguous. The service description of advertisements should be accurate and formal.
- **interrelated:** A package comprised of many advertisements contains more complicated and rich information, so the advertisements should be correlative.

Beside talking about service composition, requirement decomposition is another important process that classifies requestors' requirements into independently small pieces of requirements so that they can be matched with small advertisements.

- **Requirement decomposition** Users' requirements are usually ambiguous and comprised of various attributes. We considered those attributes what the user needs as many criterion for matching. With numbers of criterions, matching mechanism can be achieved accurately.

In other word, requirement decomposition is a process of extracting the attributes of requirements into more specific criterions according to different types of attributes. Take a trip plan requirement as an example, a trip requirement can be decomposed into many specific criterions according to dates, spot requirements, accommodation requirements, and so on.

- **Service Matching** To find appropriate advertisements and fulfill requestors' requirements, service matching is achieved by examining the subsumption relationship between the requirements and advertisement via Description Logics reasoning mechanisms. Consequently, the Web Services composition architecture needs a matchmaker component to handle the service matching process. We will introduce that in later half of this chapter. It interacts with inference engine and involves in ontology classification. In [28], the author proposed a service matching algorithm,

and we apply his matching approach in our architecture. After service matching, the matchmaker would response suggested advertisements for the user.

- **Synthesized Services** In [27], the author indicates that there are two kinds of aspects of synthesized services: Service Combination and Service Composition. To compose Web Services, we need to note what kinds of synthesized services we are handling. Different situations lead to different solutions.

- Service Combination: Service Combination involves combining Web Services that may be invoked independently without a particular order. It can be considered as feature matching between services. The features are mapped to *operation semantics composability* in the composability model proposed in [43]. We only check service profiles and match services without considering any data or control dependencies.
- Service Composition: Service Composition emphasizes the data/control flows of synthesized services from one service to another in a particular order. It can be regarded as process matching between services. The process involves *binding composability*, *operation mode composability*, and *message composability* in the composability model [43]. *Binding composability* can check the binding element in WSDL of the services that we want to compose. *Operation mode composability* can check the message dependency of message elements in WSDL. *Message composability* can check the types of message parameters in an advertisement description by WSDL.

- **Service Profile and Service Interface** In the Web Services architecture, adding semantic ability helps automatic discovery of accurate services. The goals of combining semantics technologies to Web Services are: to precisely discover desired Web Services, and to compose a sequence of tasks for a complicated service to fulfill requestors' requirements. Web Services based on Semantic technology achieve these goals with *Service Profiles* and *Service Interfaces*.

The service profile is semantic information about service characteristics, such as information of service providers and types of the service. Service profiles are ex-

pressed in Description Logics that formalize certain ontologies predefined by service providers and provide non-functional static descriptions in the Web Services environment. Therefore, either the service requestors or the service providers can express their service requirements and service advertisements if we give them the same abstract service descriptions, which are written in the same sets of ontologies. Under the same schema of ontologies the service requirements and the suitable service advertisements can be compared each other and matched then combined automatically.

Service interface is the other important part of service description. A service interface described in WSDL contains necessary information for invoking Web Services. Those functional descriptions offer requestors a avenue to access. A WSDL document is a standard document that describes how to use the Web Services. It specifies Web Services connection protocols so requestors can actually invoke the remote services, and denotes Web Services locations, operation names, input parameters, and output parameters to help requestors manipulate the services. In a real-world Semantic-based service environment, these functional properties of the Web Services are described in the following WSDL elements.

- <definitions>: specifies the WSDL document's format and its location.
- <message>: specifies the input and output variables of operations in WSDL. The input variable can refer to requestors' "Request" parameters of requirements and the output variable can refer to the "Response" for the requestors.
- <portType>: defined as a Java class name and its subelement <operation> is a function that requestors can call.
- <binding>: describes the specific transportation protocols that requestors can invoke.
- <service>: provides the essential information about how to find the service of interest and its actual location.

Based on the service description, it is possible to achieve machine-processable contents with meaning for humans and to implement communication between com-

ponents in the architecture or between service requestors and service providers. In addition, automatic Web Services discovery, composition, and invocation are facilitated with the ability of ontology reasoning

In this thesis, we assume Web Services are sort of service composition because we care about the order and data/control flows. A more detailed description of the service composition mechanism based on semantic technology is given in 4.4.2.

4.1.2 Design of Web Services Composition Architecture Based on Semantic Technology

Based on our previous studies and projects like [60], [27], [5], and [57]. A complete architecture providing service matchmaking and service invocation was proposed. It accepts the user's requirements, discovers suitable Web Services, dynamically assembles existing Web Services, provides customized services, and invokes these services. At this time we emphasize on the two parts. First, we strengthen the responsive interaction between the user and the system by combining web 2.0 technology, trying to make Semantic Web applications under the architecture easier and more friendly for the user. The second, we bring the idea of community-driven ontology engineering [41] into our architecture. Ontology maintenance is not longer controlled by a small group. Through Community Component, every user could join the community to collaboratively edit ontology. It makes ontology more flexible and agility to ensure that the concepts in ontology are not obsolete. The Web Services composition architecture provides automatic service discovery, collection, composition, and execution. The Web Services composition system consists of six main components based on the shared and pre-defined ontologies: Service Composer, AJAX component, Community component, Service Collector/Annotator, Inference Engine, and Execution Engine. Detailed descriptions are given below:

- **Ontologies**

Ontologies play an essential role in sharing and exchanging knowledge in the architecture of the Semantic Web Services composition. Each component in the architecture shares the same ontologies and communicates with other components by accessing ontologies. The ontologies should be defined generally and flexibly

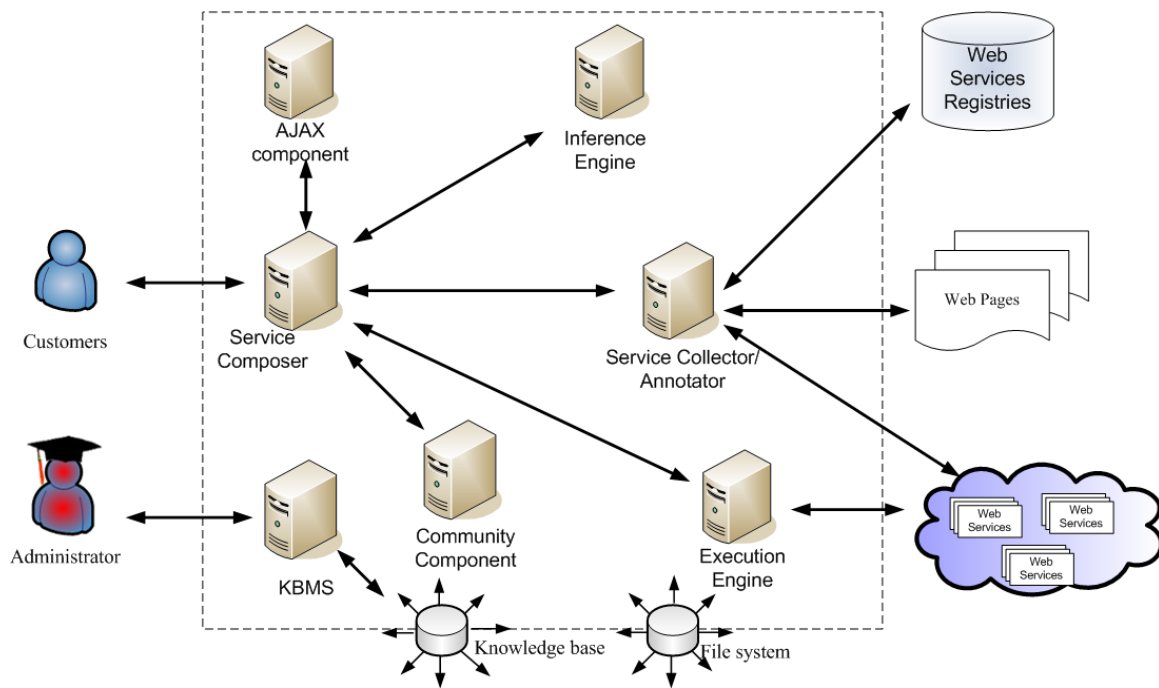


Figure 4.1: Web Services Composition Architecture

for broad and long-term use. Ontology maintenance is a time-consuming and complex job for administrators, especially when the ontologies are maintained by a small group. Obsolete concepts in Ontologies leads to a wrong deduction that would cause a huge damage to a reasoning-based system. Therefore, we have to notice whether Ontologies is consistent with the reality. Besides, there are many constraints need to be checked periodically to maintain ontologies consistent and correct. It is also a challenge to handle contiguous constraints, such as value and time.

In addition, designing the ontologies is the fundamental task in our architecture. Ontologies modeling are wide-vary in different domain applications. We introduce our detailed ontologies modeling approach in 4.4.

- **The Service Collector/Annotator**

The task of Service Collector/Annotator is automatically discover Web Services on the World Wide Web, collect information about service interfaces, and service profiles and then store it in the database of our system in order.

The database schemas in the architecture are designed based on the shared ontolo-

gies. To match for suiting service, service profiles stored in the database will be mapped to the ontologies as concept expression. The Service Collector/Annotator collects two types of information: static information and dynamic information. Static information is the corresponding WSDL file of the service, including the service providers' name, and the other related information. For example, the flight schedule in service profile of flight reservation service seldom changes. This information can be stored in database as local data as a service description for other components to use. The static information is updated at mid to long-term period, say every few weeks or months. In contrast, dynamic information/data, such as real-time data in providers' Web pages link, is changed daily or as short notice.

The Service Collector/Annotator collects services from Web Services registries like UDDI. It also retrieve the data from the Web pages or existing databases of service providers. The Service Collector/Annotator parses Web pages and obtains the necessary data, and maps the service data from the database of service provider into the shared ontologies in the architecture. Also the Service Collector/Annotator should handle problems related to the mapping between different ontologies. For example, if the service providers have different ontology schemas from ours, the Service Collector/Annotator is responsible for mapping those knowledge into the same terms of the shared ontologies. The mapping process will not create a new ontology but import the knowledge from outside into our existing ontology. [14], [61], and [38] illustrate the detailed mapping process.

In a word, the Service Collector/Annotator attempts to collect static and dynamic information on the Internet and are designed to handle ontology mapping if the service profiles are described in the different ontology schema. On receiving a request from the Service Composer, which is a core component in the architecture, the Service Collector/Annotator will query the local database for suited service profile. If the requested data exists, it transforms the data into the format defined in the shared ontologies and sends it back to the Service Composer. It will trigger a request to the corresponding service provider for the latest service profiles if the requested data does not exist.

- **The Service Composer**

The Service Composer plays as a core component in this Web Services composition system. It is a bridge that connects the customer and the whole architecture of the system. We define the term "customers" is those causal users who do not have any Description Logics background. They want to manipulate the system in a easy way. And we define another term named "administrator" who are charged with the ontology maintenance. Because different domains have different kinds of requirements and different business processes. A implementation of Service Composer is designed for a specific domain. It contains modules that collaborate interaction between components in the architecture. Although the implementation of Service Composer is vary according to different application, but the fundamental architecture of Service Composer we propose is generally suiting for all kinds of ontology-Based automation of Web Services composition.

In general, the Service Composer coordinates with the user, AJAX component, Community component, the Inference Engine, and the Execution Engine. It has four main subtasks: First, the customer defines their incomplete and ambiguous requirements by using integrated user interface. The module of Integrated user interface communicate with AJAX component to converts the customer's requirements into formal semantic language. Second, the Service Composer passes formal requirements to Matchmaker responsible for communicating the Inference Engine to find suitable advertisements. Third, Matchmaker return suitable advertisement to the Service Composer. Then that collaborate with the AJAX component and covert the formal advertisement described in Description Logics back to graphical figures that display on integrated user interface. Alternative suggestions are also displayed according to the ranking similarity.

Customers are able to decide which advertisement satisfies their needs and ask the system to execute related Web Services behind that. All revelent Web Services will be packages as the composition Service, which is described by a BPEL4WS document. Finally, the Service Composer send that document to the Execution Engine.

In addition, through collaboration with Community Engine, the Service Composer provide the user a open community environment to discuss their requirement and share their experience about using the system . When the system cannot find a suiting advertisement for the customer, the customer can publish their unanswered requirements on the community that suggest the administrator to update related advertisements in ontologies. Furthermore, administrator could release part of ontology on the community. Let every user participate in ontology maintenance, decreasing maintenance time and making concept in ontologies not obsolete.

- **The Inference Engine**

The Inference Engine is responsible for interacting with the Service Composer and matching services through subsumption reasoning based on Description Logics(DLs) inference. It provides approximately matching services which correspond to the requirements and helps ensure the dependencies and constraints between individual requirements. In a word, the Inference Engine has an inference capability to find suitable services and return them to the customer.

- **The Execution Engine**

When the Customers decides the service they want, the Service Composer produces a related process document in terms of service execution and pass it to the Execution Engine to invoke the composition service. The Execution Engine is responsible for invoking the composition services correctly and controls the status of the involved Web Services. According the work flows in the process file, which is written in process language like BPEL4WS, the invocation of the Web Services executes sequently. If any Web Services fail, the Execution Engine will charge the rollback of all the executions and also inform the Service Composer a execution failure message. If all the Web Services execute successfully and finish, the Execution Engine will return a successful message to the Service Composer and send it back to the customer afterwards.

- **The AJAX Component**

Asynchronous JavaScript and XML, as known as AJAX, is one of the technical pillar of Web 2.0 technology. AJAX is a group of inter-related web development

techniques used for creating interactive web applications .It increases the responsiveness and the interactivity of web pages by exchanging small amounts of data with the server "behind the scenes." In addition, it improves the user-experience in browser-based applications. The AJAX Component adopts AJAX technique so that entire Semantic Web application do not have to be reloaded each time there is a need to fetch data from the server. It is intended to increase the system's interactivity, speed, functionality and usability.

Therefore, we want to bring that advantage of AJAX into Semantic Web applications to lower the entrance barriers. The AJAX Component is responsible for hiding the formal Semantic information and generating a interactive GUI on the Service Composer. In other word, the AJAX Component encapsulates the complex information which are not proper for the customer and convert it into the nature language or graphics . While displaying the matching result to the customer, it also covertes the suiting advertisement defined in Description Logics back to human-readable format, such as pictures, videos.

- **The Community Component**

The Community Component is another great idea we borrow from web 2.0 technology. Web 2.0 approaches empower the individual to take part in community activities by lowering the barriers: informal, lightweight, easy-to-use, and easy-to-understand. The community we've seen in web 2.0 pages allows contributors to collaborate and share information easily. For example, Wikipedia is a community, there are hundred and thousands volunteer people collaboratively create and maintain the knowledge on the Wiki site. In [41], they propose a new collaborative approach for Ontology Engineering. The Community Component is responsible for creating a open Wiki-based environment where the user can share their requirements and match results. Besides, the customer participates in maintenance a light-weight ontology though Wiki-based community.

Figure 4.1 shows the Web Services Composition Architecture and how the components interact with each other.

4.2 Service Composer

4.2.1 Design of the Service Composer

In Figure 4.1, the Service Composer plays the central role in the Semantic Web Services composition architecture. It is also a bridge which connects the user and the system. Because complicated Semantic languages, such as Description Logics, impose high entrance barriers for casual users, we decrease information complexity by removing unnecessary modules from the Service Composer.

Overall, it helps the customer define their complicated requirements, and achieve their goals by matching advertisements and executing the compound services. In fact, the Service Composer cannot finish those tasks by itself. A completed procedure of those task involves many components. As a pivot in the system. The Service Composer collaborates other component in the architecture.

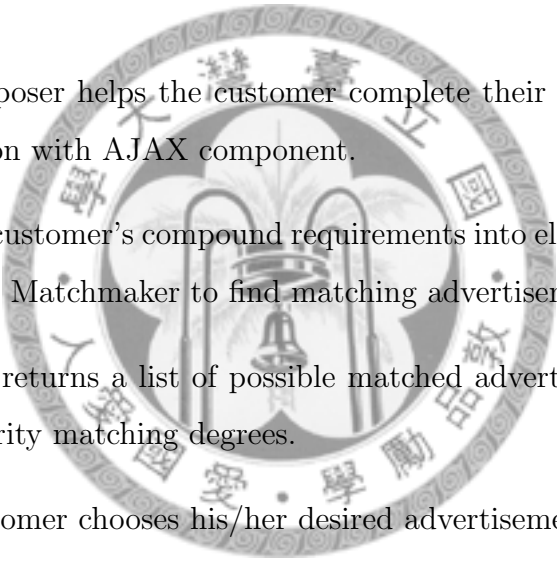
When customer manipulates the system, the Service Composer communicates with AJAX component to provide the customer a responsive user interfaces. While the customer inputting their requirements, the Service Composer translates the customer's ambiguous inputs into detailed and formal semantic descriptions. Before sending requirements to Inference Engine, the Service Composer attempts to decompose the customer requirements (a compound task) into many sub-requirements(subtask). A simple sub-requirement may be satisfied by many different sub-advertisements. And each sub-advertisement corresponds to certain Web service described by WSDL document. The task of service composition mechanism and service decomposition are implemented in the Service Composer component in order increase the accuracy of matching.

After decomposing the customer's requirement, the Matchmaker in the Service Composer Component calls the Inference Engine to find suitable services by approximate matching. The subsumption reasoning relationship between description of the requirements and advertisements will be checked at matching stage. A list of matched services with corresponding degrees, which represent the matching similarity between the requirements and advertisements, will be returned to the customer. The customer can choose one set of the matched services and ask the Service Composer to invoke the related Web Service. Meanwhile, the Execution Module in the Service Composer obtains the WSDL

files of the selected services, assembles them into the BPEL file according to the work flows of the services, compiles the related setting files about the BPEL engine , deploys them in the Execution Engine, and invokes the BPEL services.

In addition, Service Composer provide the user a open community to discuss their requirement and share their experience about using the system on their personal pages. When the system cannot find a suit advertisement for the customer, the customer can publish their requirement on the community to suggest the administrator to add related advertisements in ontologies. Furthermore, administrator can release part of ontology on the community. Let every user participate in ontology maintenance, decreasing the maintenance time and making concept in ontologies not obsolete.

The whole process and the interaction between components can be described as follows:

- 
1. The Service Composer helps the customer complete their requirements through a of series interaction with AJAX component.
 2. It transforms the customer's compound requirements into element requirements and sends them to the Matchmaker to find matching advertisements.
 3. The Matchmaker returns a list of possible matched advertisements sorted by corresponding similarity matching degrees.
 4. After the the customer chooses his/her desired advertisement package, the Service Composer will ask Service Execution Module to execute related Web Service.
 5. The Service Execution Module transform the abstract descriptions of the selected advertisements into the executable descriptions, i.e., BPEL4WS files. The BPEL4WS files are deployed to the Execution Engine and wait to be invoked.
 6. When the advertisements are executed successfully, the system sends a message to the user. In addition, the Service Execution Module has a mechanism to handle execution failures.
 7. The customer can publish the requirement and the advertisement he/she chose on Wiki-based Community Component, sharing his/her opinions about this system.

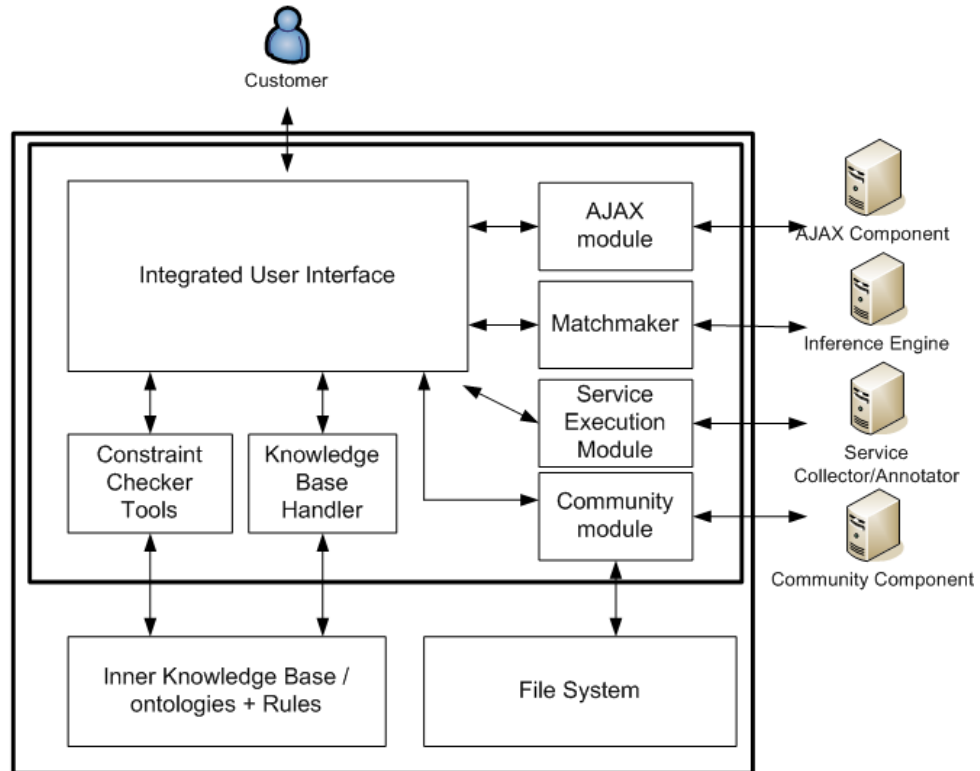


Figure 4.2: Architecture of the Service Composer

Meanwhile, the customer's experience may help the administrator improve the inadequate ontologies.

4.2.2 Architecture of the Service Composer

Figure 4.2 shows the architecture of the Service Composer. The bold rectangle represents the scope of the Service Composer. Outside of the bold rectangle are other components that cooperate with the Service Composer. The components from up to down are the customer, AJAX Component, the Inference Engine, the Execution Engine, and the Community Component in the Semantic Web Services composition architecture. We use arrows to represent the interaction between the components. Detailed definition of the components inside the bold rectangle are given below.

- **Inner Knowledge Base / Ontologies and Rules**

Our Knowledge base consists of Ontologies and rules. As we mentioned at previous

chapter, ontologies play an essential role in sharing and exchanging knowledge in the architecture of the Semantic Web Services composition. Each component in the architecture shares the set of ontologies and communicates with other components by accessing ontologies. Ontologies are kind of formal representations of knowledge. Rules used to describe the complicated relationships between roles and we can use rules to capture the role composition in the ontologies. We use Inner Knowledge Base for certain functions:(1) data storage: Requirements, advertisements, and related information are described in OWL which based on DL. (2) With deduction power of the inference engine like RACER and, reasoning allows us to infer implicitly represented knowledge from the ontologies.

Because all data flows within the Service Composer are read from the knowledge base and are written back, the component in the Service Composer can communicate with the inner knowledge base directly or indirectly. (See the arrows in Figure 4.2) When the user input their requirement, they are stored in the inner knowledge base so that the data can be extracted later when needed.

- **Integrated User Interface**

The Integrated User Interface mashing up with AJAX component provide the customer responsive user interfaces and aid them input their requirements step by step. It is also responsible for displaying the matched results after service matching. Integrated User Interface encapsulates the complex and formal information which are not proper for the customer. Integrated User Interface provides error detections if the customer inputs inappropriate data. Besides, it will invoke the constraint checker tools to check the consistency of the constraints after the user complete their requirements. The whole process can be simplified as follows. First, the customer sets their requirements by interacting with the AJAX component. Second, he/she can review and re-edit their requirements. Third, he/she submit the requirements to the Matchmaker to search for suitable advertisements and get the ranking scores of the advertisements. Finally, the customer can invoke the desired advertisements from the result list.

The Integrated User Interface plays an important role because it directly interacts

with the customer. To lower the entrance barrier of Semantic Web application, we should put the design of the friendly UI as the first priority.

- **AJAX Module**

Through cooperating with the AJAX Component and the Integrated User Interface, AJAX module is responsible for passing the parameters which involve in displaying AJAX-based interfaces. While the Integrated User Interface is displaying the responsive user interface, for instance, a interactive map, AJAX module retrieves necessary information from the ontology, such as location name, descriptions of locations. Meanwhile, it continuously listen the interaction events triggered by the customer' behavior, such as mouse clicking, mouse trajectory. Through clicking mouse on the map, the customer input their requirement easily.

- **Community Module**

Community Module connects the Service Composer and the Community Component. After the customer input their requirements and select their desire advertisements, community module publish those requirements and corresponding advertisements to the Community Component. Besides, if the customer input a requirement that contain some information not in the ontologies, the Community Module will automatically report to the Community Component.

- **Matchmaker**

The Matchmaker is a matching module that invokes the Inference Engine to start reasoning. It acts as a bridge between the Service Composer and the Inference Engine. We implement the matching algorithm and the approach of computing similarity degree in this component. The service matching algorithms are tightly dependent with the domain ontology and are designed for a specific domain or a specific system. After finding matched services through the reasoning of the Inference Engine, the Matchmaker returns the desired advertisement lists according to the similarity degree.

- **Service Execution Module**

The Service Execution Module provides tools to generate executable processes from

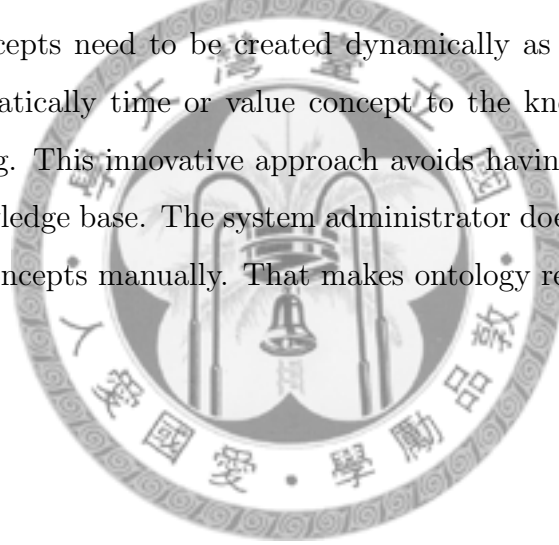
abstract service descriptions and templates, and invokes the Execution Engine. It returns the results from the Execution Engine to the the customer. A more detailed description of the service execution stage is given in Section

- **Knowledge Base Handler**

The Knowledge Base Handler handles the all the access of the ontology, including write-in and read-out. Through the Knowledge Base Handler, every module in the Service Composer is able to retrieve the data from ontology.

- **Dynamic Concept Component**

To handle the subsumption between requirements and advertisements, we have to overcome the subsumption between the numeric concepts. Since the particular concepts like Time and Value Partition are related to unlimited concepts in the real-world, those concepts need to be created dynamically as a programming method that adds automatically time or value concept to the knowledge base while performing reasoning. This innovative approach avoids having large specific concepts in the inner knowledge base. The system administrator does not have to define and maintain large concepts manually. That makes ontology reasoning more efficient.



4.3 Knowledge Base Management System

4.3.1 Design of the Knowledge Base Management System

The Knowledge Base Management mainly provide the administrator an integrated management environment to edit and maintain their ontologies. Different from the Service Composer is dedicated to the customer, the Knowledge Base Management is dedicated to the administrator. With the Knowledge Base Management, the administrator is able to maintain ontologies, inference implicit knowledge, an check the correctness of the ontologies. Recent years, there are several outstanding ontology management system, such as Protégé, OntoEdit, OilEd. Our architecture adopts Protégé and related plug-ins, which are used to extend its management ability, as the Knowledge Base Management.

Protégé Axiom Language (PAL) tab-widget provides constraint checkers to examine different kinds of constraints. according to the needs of the application, the ontology constraints are defined by Protégé Axiom Language. We will introduce constraints handling in detail in section 4.5. In addition, Protégé SWRL tab-widget supports SWRL rules reasoning which allows implicit knowledge to be inferred by asserting certain rules.

4.3.2 Architecture of the Knowledge Base System

Figure 4.3 shows the architecture of the Knowledge Base System. The bold rectangle represents the scope of the Knowledge Base System. Outside of the bold rectangle are other components that cooperate with the Service Composer. We use arrows to represent the interaction between the components. Detailed definition of the components inside the bold rectangle are given below.

- **Inner Knowledge Base / Ontologies and Rules**

The Inner Knowledge Base we define in the Knowledge Base Management System is the same as the Service Composer's one. They share the same set of ontologies and rules. The most different point is the Knowledge Base Management System manipulates the ontologies by reading and also writing. But the Service Composer focus on reading rather than writing.

Because all data flows within the Service Composer are read from the knowledge base, the Knowledge Base Management System has to check the validity of the

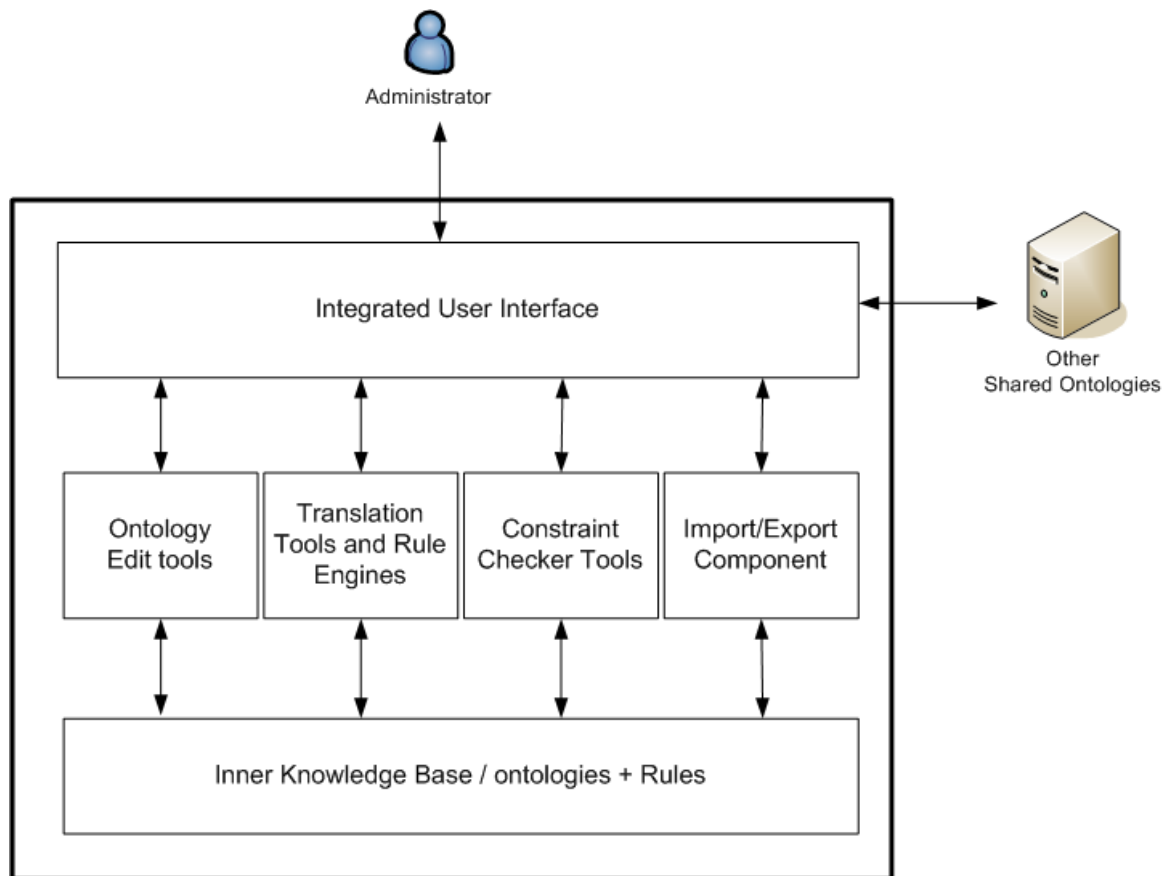


Figure 4.3: Architecture of the Knowledge Base System

ontologies to ensure their consistency by examining the constraints. With the validation mechanism, case of GIGO(garbage in, garbage out) can be avoided. There are several related constraint checker tools in the Knowledge Base Management System to keep the consistency of the knowledge base.

- **Integrated User Interface**

the Knowledge Base Management System provides an Integrated User Interface that makes the administrator maintain ontologies in a comfortable and convenient way. The Integrated User Interface includes all kinds of functions for maintain including ontologies editing, SWRL rules editing, ontologies inferring, constraints defining, validation checking, ontologies import/export tools.

- **Import / Export Component**

The goals of ontologies are defined as common use, sharing, and exchanging in Semantic environments. The Import/Export component allows different component not in our architecture, like other application, to exchange ontologies through file transportation. Outside ontologies can be imported as a plug-in into the inner knowledge base, and ontologies in the knowledge base can also be exported. The ontologies are exchanged and combined via the Import/Export Component without notifying the customer. It is a seamless and flexible mechanism in the service-oriented approach.

- **Translation Tools and Rule Engines**

Rules are defined to increase the expressive powers and complement the limited expressiveness of the ontology language. There are many ongoing efforts to design rule languages for the Semantic Web. These rules can be defined by the administrator through the user interface of the Knowledge Base Management System. They can infer implicit knowledge from the defined rules through a translation tool that transforms the rules into the specific ontology language according to the rule engines that enable reasoning. For example, the SWRL rules can be defined with a SWRL-Tab editor as a Protégé plug-in. We adopts the Jess rule engine as the Rule Engines that allows implicit composition roles to be inferred by SWRL rules and stored the new knowledge in the ontologies.

- **Ontology Edit Tool**

The Ontology Edit Tool provide the administrator basic functions to edit their ontology. The functionality of editing includes addition, remove, and modification for concepts, roles, and individuals.

- **Constraint Check Tools**

To ensure the global consistency of the inner knowledge base, the constraint check tools are designed for examining the correctness of the constraints. The tools handle different kinds of constraints like quantitative constraints and non-quantitative constraints, we will introduce in detail in section 4.5. Constrain Check tools are designed to solve the domain problems based on domain ontologies with different reasoning tools and engines.



4.4 Ontology Modeling

Our system is based on Semantic Web technology. Consequently, ontologies play an important part in the architecture of the Semantic Web Services composition. They characterize the non-functional properties of Web Services and Web Services profiles. To communicate with different components, each component exchanges information by sharing the same set of ontologies in the architecture. In this section, we attempt to illustrate the approach of ontology modeling.

TBOX Modeling and ABOX Modeling

An ontology is a formalization of concepts in a specific domain. They are essential for knowledge reusing, exchanging, and sharing. An ontology contains two parts. The first part, called TBox, is to define terminologies with concepts (or classes) and the terminology taxonomy using concepts (or classes) and subconcepts (or subclasses) relationship. Besides, it defines properties (or roles) to describe the relationship between concepts. The second part, called ABox, is to assert individuals (or instances) corresponding to the previously defined concepts in the first part.

Because ontology languages are based on Description Logics (DLs), we can facilitate subsumption reasoning based on the inference theorems of DLs. Therefore, we adopt concepts expression to represent the requirements and advertisements.

Ontology language modeling involves TBox modeling and ABox modeling. The concepts and roles refer to TBox modeling, which defines a concept hierarchy and also the relationships between concepts, and the individuals refer to ABox modeling.

TBox conceptual modeling supports subsumption reasoning so that the similarity degree between requirements and advertisements can be inferred. During the concept modeling phase, concepts and roles are constructed to describe the requirements and advertisements. Using inference engines, such as Racer, to match the requirements with advertisements.

System-Specific and Common Ontologies

In Semantic-based Web Services composition architecture, ontologies are divided into two types. The first one is system-specific ontologies, which are for specific domains.

For example, in the implementation of our system, the Traveller, we took tourism as our specific domain. System-specific ontologies supports the definition of the requirements and the advertisements and also supports the operations about matching scheme. We will attempt to illustrate the Design of the Ontologies of the requirements and the advertisements in 4.6). System-specific ontologies have to concern about handling common constraints and checking global constraints. And also it collaborates with SWRL rules for capturing the complicated relationships between properties in the ontologies.

The other type of ontologies, common ontologies, contain domain-independent knowledge which can support system-specific ontologies. For example, in the tourism domain, when a new trip requirement is created, it will be stored at system-specific ontologies. But a trip requirement contain other related concepts about the trip such as time, value, locations, those domain-independent information is stored at common ontologies. Because of the independent characteristic of common ontologies, they can be imported from other ontologies providers or share them with other applications.

There are several ontology languages that have a trade-off between expressive powers and computing complexities. The choice of what ontology need to be implemented depends on the needs of the application types and the computing ability of inferencing.

Design of Ontology - Requirement and Advertisement

The ontology design for the Service Composer, which are used to store the user's requirements and the service providers' advertisements in the Semantic-based service composition architecture is the most important part. It is also closely related to the service matching approach.

Based on [27], each requirement is defined as a concept (or a class). We use the same concepts scheme to model advertisements so that requirements and advertisements can be matched easily for their subsumption relationships. In ontology modeling, the user's requirements and service providers' advertisements are regarded as concepts. Because of the inference limitation of SWRL rules for capturing the implicit relationships, we define an individual as a realized instance to represent the corresponding concept.

A composite service, which combines many desired services in a specific order, needs to be represented in the ontology. As we state previously, the Service Composer can

decompose the user's requirement into many sub-requirements. We regard those sub-requirements as many sub-concepts in the concept hierarchy of system-specific ontologies. Figure 4.4 shows the TBox approach modeling.

Requirement - Class and property definitions:

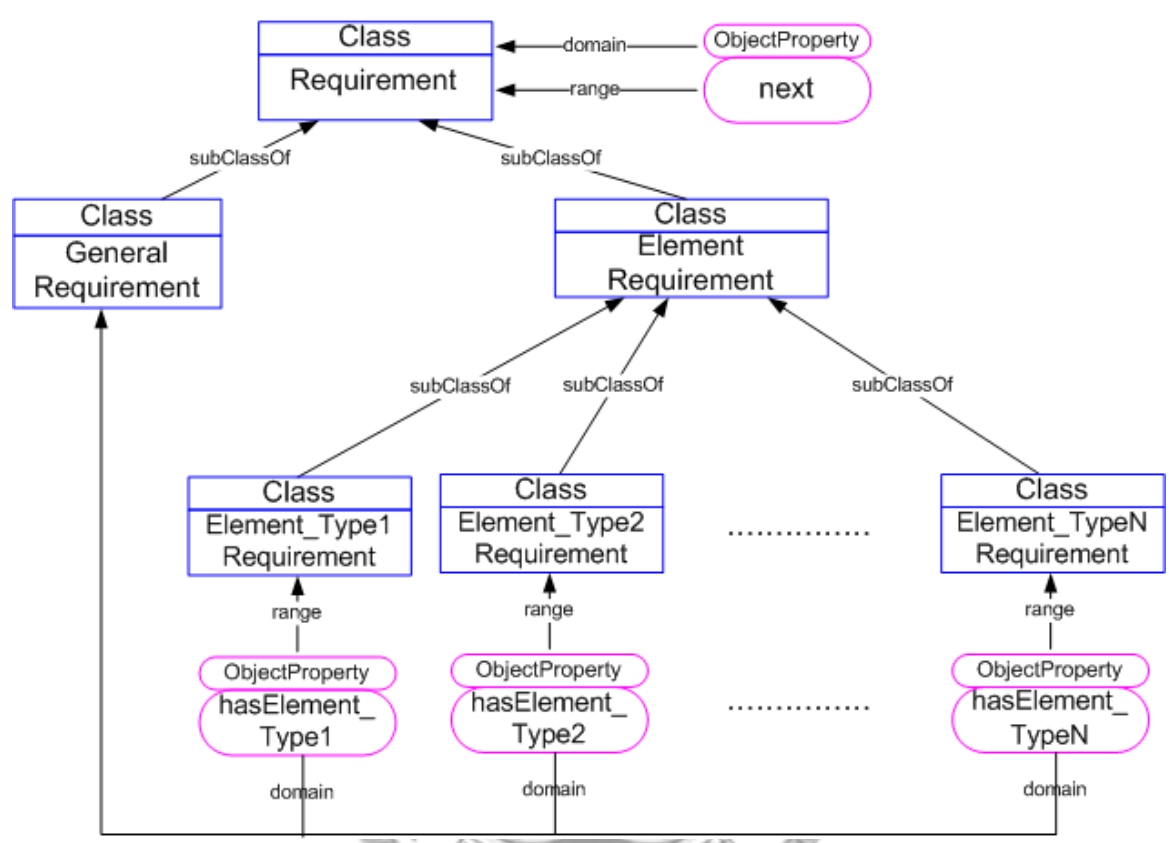


Figure 4.4: **Requirement** Modelling - The TBox Approach

In Figure 4.4, the root class **Requirement** has two subclasses: **General Requirement**, which defines the customer's requirements, and **Element Requirement**, which defines element requirements of **General Requirement**. The **Element Requirement** has several subclasses that represent different types of element requirements, such as **Element_Type1 Requirement** and **Element_Type2 Requirement**. The **General Requirement** has properties, such as **hasElement_Type1**, and the range of each object-type property depends on its type class. Figure 4.5 shows an example of a complete requirement, which is composed of two general requirements **MyReq_1** and **MyReq_2**.

A Requirement example:

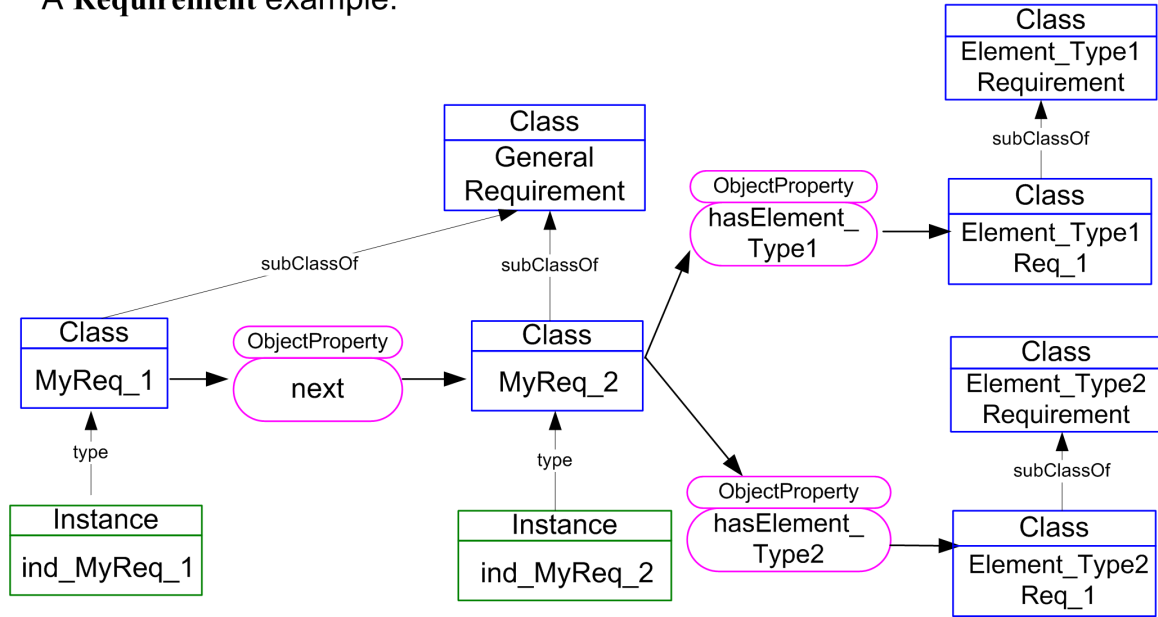


Figure 4.5: Requirement Example - The TBox Approach

MyReq_1 and **MyReq_2** are subclasses of **General Requirement** connected by the object-type property **next** inherited from super class **Requirement**, which **next** represents the order of the element requirements. **MyReq_2** contains two element requirements: **Element_Type1Req_1** and **Element_Type2Req_1**. Each element requirement is a subclass of the respective type element requirements. Besides, **MyReq_2** correspond to a individual, **ind_MyReq_2**, to represent the class for SWRL rule reasoning. **Advertisement**, which defines providers' advertisements of Web Services, is designed by the same approach as **Requirement**. We can find the same class hierarchy of **Requirement** and **Advertisement** defined in the system-specific ontology, shown in Figure 4.6. The figure also shows the common ontology including **Location**, **Time**, and **ValuePartition**, which describes the domain-independent information in concepts to support the tourism domain ontology. For example, **Location** defines tourism location in concepts, **Time** defines the measurement unit of time in concepts and **ValuePartition** defines those values or numbers in concept such as, budget. Section 4.5 contains more detailed descriptions of design **Time** and **ValuePartition**. Figure 4.7 presents an example of the subsumption reasoning and the result.

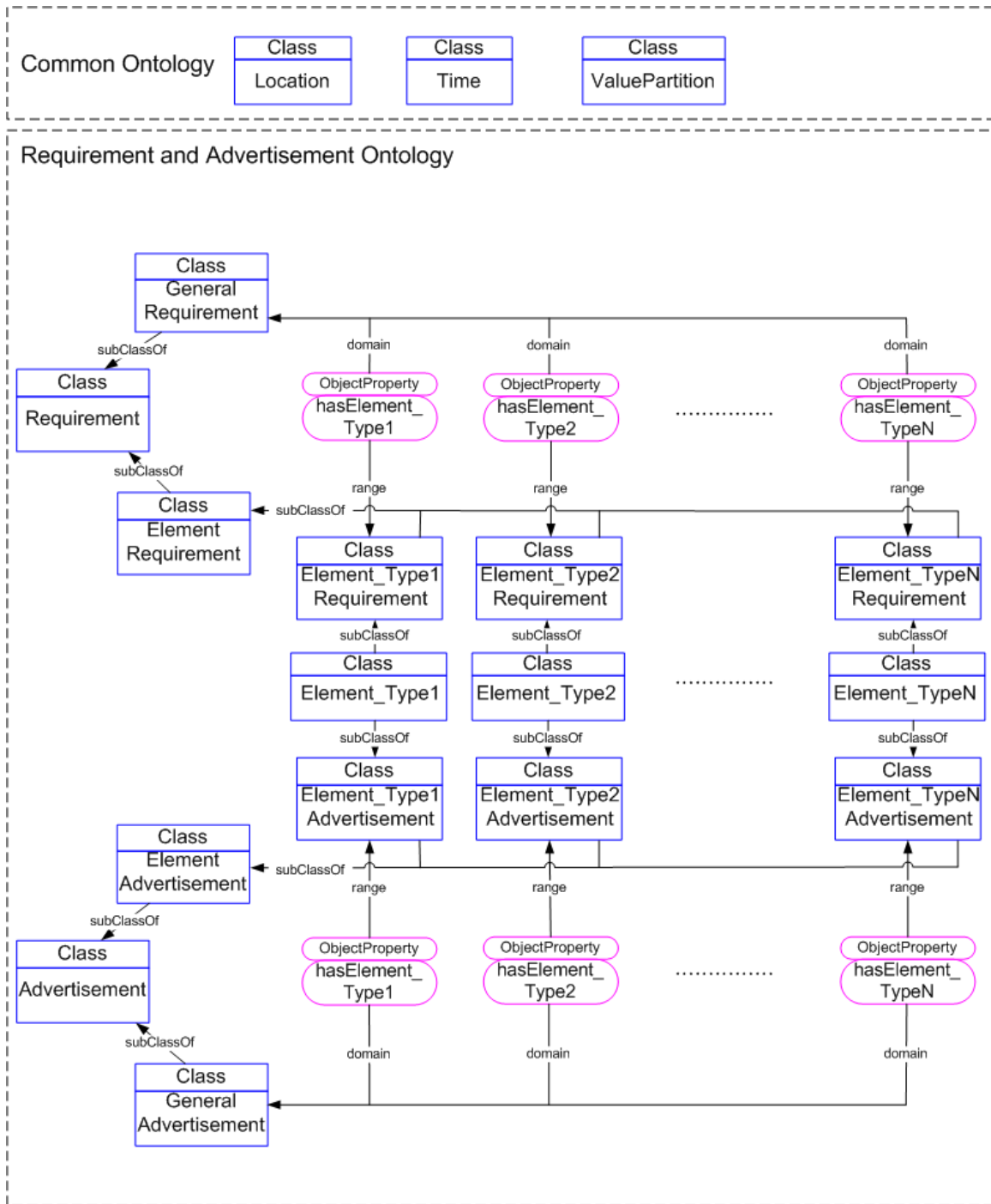


Figure 4.6: Ontology Design of the Architecture - **Requirement, Advertisement** and Common Ontology

Subsumption Reasoning (Advertisement and Requirement):

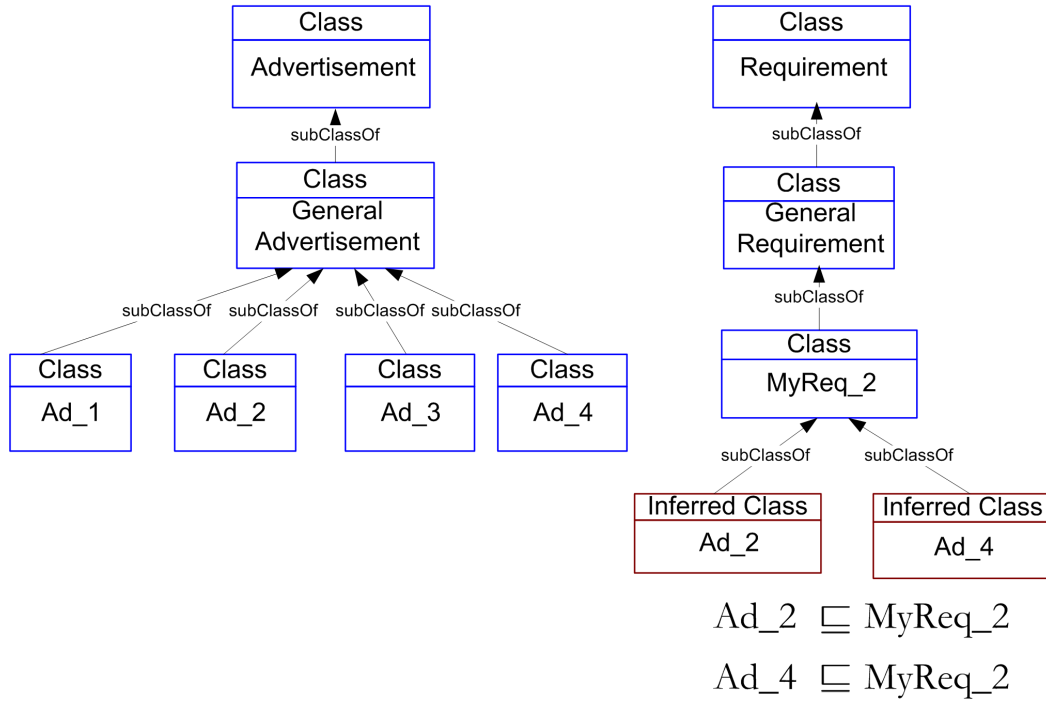


Figure 4.7: Example of Subsumption Reasoning

4.4.1 Service Composition Mechanism

Since Web Services technologies become maturing and convenient, we start to compose many services together to complete a complicated task. Therefore, we need to concern about the execution order of the services. Different orders may lead to different consequences.

Based on the Inference rule of Hoare Logic [Hoare 1969], the Sequence rule is the essential aspect of the composition. However, few studies express the rule in detail.

$$\frac{\{P\}S_1\{Q\} \quad \{Q\}S_2\{R\}}{\{P\}S_1; S_2\{R\}} \quad (\text{Sequence})$$

In [9], DL supports the composition of relationships to define concepts and roles.

If these assembled services are independent without involving interaction each other, it

is very easy to handle them. Just executing them in any order. However, if there are several interactions between these composed services, then the composability of the services should be considered [43]. [43] proposes a composability model to check whether component services are composable [10]. The composability model for Web Services consists of six parts: *binding composability*, which compares the binding protocols of interacting services; *operation mode composability*, which compares operation modes including notification, one-way, solicit-response, and request-response; *messages composability*, which compares the numbers of message parameters, data types, business roles, and units; *operation semantics composability*, which compares the semantics of service operations; *qualitative composability*, compares the qualitative properties of Web services; *composition soundness*, which checks if the combination of Web Services in a specific way is worthwhile.

According our previous research [57], the Semantic-based service composition architecture focuses on service composition and emphasizes the data/control flows of synthesized services from one service to another in a particular order. When the desired Web Services are composed to a service, the composability of the services should be considered. We pick four principles from the composability model. The *binding composability* can be taken as checking **binding** element in WSDL of two composed services. The *operation mode composability* can be seen as checking the message dependency of **portType** in WSDL of the two composed services. The *message composability* can be examined by **message** types in WSDL of the two composed services. The *operation semantics composability* of the two composed services should be checked using the service description in OWL.

Besides, in the Semantic Web Services composition architecture, services are composed by checking the composability and extending the aspects of the Sequence. A requirement is composed of many element requirements. An advertisement is also composed of many element advertisements. When the Service Composer wants to compose the element requirements, it has to check the composability of the related element requirements first. The **next** object property in the ontology model (Figure 4.6) represents the composition and also expresses the order of the involved requirements.

4.4.2 Service Execution Based on Semantic Technology

In the Semantic-based service composition architecture, services are invoked in the same way as in the Web Services environment. Those Web Services annotated with semantic can be invoked through SOAP messages or other appropriate protocols. However, we take advantage of invoking more accurate services because these services are matched and selected according to the customer's requirements and preferences. The invocation sequence of the services is also important in the architecture. The composite services are assembled as BPEL according to the business policy and the execution order. The service execution based on Semantic technologies in the architecture has to implement the fault tolerance. For example, alternative services will be selected to substitute the failure service in the service matching stage [21].

4.5 Constraint Handling

In the Semantic-based service composition architecture, constraints are used to represent the respective conditions of requirements and advertisements. For example, we use constraints to restrict some requirements' and advertisements' attributes, such as time and cost. Constraints are also applied to Web Services selection in the QoS architecture and can be divided into two types: global constraints and local constraints [6]. In [62], the Semantic and dynamic service selection framework contains a *Constraint Analyzer* to analyze the characteristics of the constraints and handle them via appropriate approaches.

Checking constraints ensures the consistency of the knowledge base including the common ontologies and the domain-specific ontologies. Different types of constraints are checked by different approaches. In addition, specifying a number constraint in a specific range is a frequent task. We adopt value partition approach to subsumption checking for the comparison between two numbers. For instance, we want a certain number in ontology lower than a threshold. Our system adopts this kind of range constraint to restrict common ontologies like time and value partition.

Sometimes, those constraints with subsumption cannot be a effective restriction because of concept hierarchies design. In this case, we have to implement additional val-

idation in programs. However, using programs is not a good solution because it is lack of flexibility. That means those constraints will be bundled with a specific domain, of a specific system.

Next, we summarize the classification of constraints and the solutions for handling them. The common ontologies of time and value partition are detailed in subsequent sections.

4.5.1 Constraints

According to the framework in [62], constraints are basically divided into two types: **quantitative constraints** and **non-quantitative constraints**. Quantitative constraints are constraints that can manipulate the four fundamental arithmetic operations. For example, the total price of an trip order must be the sum of all trip prices. We can say total price has quantitative constraints that it must be equivalent to sum of all trips. In contrast, non-quantitative constraints use to restrict those features which do not support arithmetic operations. For example, if a trip consists of three-day itineraries, one of the feature of the trip, the first day of the trip must be the same as the start date of first itinerary.

Exception for the categories of quantitative or non-quantitative, constraints can be particularly divided into binary relationships and global relationships. A binary relationship means that the constraint relationship only involves two different features, such as checking if start date of a trip is before the end date of a trip. Constraints of binary relation compares two features to decide whether the concepts are consistent in the ontology. We often adopts subsumption approach in binary relation comparison. That's why we need to dynamically construct the concept hierarchy for time and value. Different from binary relationship, global relationships involve more than two features, such as checking whether the total budgets of a trip is equivalent to the total sum of all itineraries. As far as we know, it is impossible to express the subsumption relationship between more than three concepts in OWL ontology modeling. Therefore, we must find another approach rather than subsumption approach. When handling global constraints, we define SWRL rules to solve non-quantitative constraints or use a program to solve quantitative constraints.

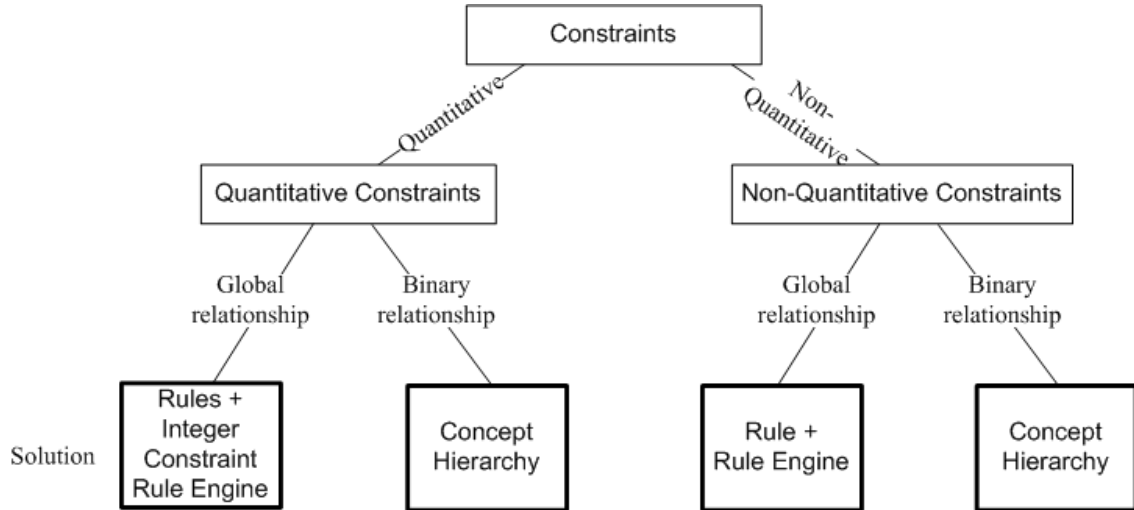


Figure 4.8: The Classification of Constraints

Based on the quantitative types and the constraint relationships, we classify constraints and propose a solution about constraint handling beyond the approaches in [62, 39, 6]. Figure 4.8 shows the classification of constraints.

4.5.2 Time Constraint

It is controversial to judge that Time belongs to a quantitative constraint or a non-quantitative constraint. According to the time temporal concepts defined in [1], the Time duration has quantitative attributes that can be taken as quantitative constraints. For example, if it takes one hour and fifteen minutes from Taipei City to Hsinchu City by train, the value of the time unit, hour, can accumulate with another time duration, minute.

In the Semantic Web Services architecture, we simplify the unit of the Time constraint and we only consider one kind of time unit, date. A date consists of months, days, and years. We adopt the approach proposed by [39] to handle time and value partition that dynamically construct value concept in the value concept hierarchy for subsumption reasoning. There are **before** and **after** relationships between the dates of Time concepts in the Time ontology. Through checking the subsumption relationships between different Time concepts we can decide the sequence of the time and apply that in our domain

application.

During dynamical concepts constructing, the needed time concepts and also the necessary relationships are added to the knowledge base. However, this approach [39] for handling time wastes space in the knowledge base and reduces the inference efficiency as time goes on.

4.5.3 Value Partition

In Figure 4.8, if we want to handle the binary relationship between two quantitative features, we define the concept hierarchy of the quantitative constraints. To make subsumption checking, those quantitative concepts are constructed dynamically as object-type concepts. In the Semantic-based service composition architecture, we apply the Value Partition ontology proposed in [39]. The Value Partition approach uses subsumption relationship to express the comparison of amounts. By checking subsumption, system can inference which quantitative value is greater or less than the other.

Figure 4.9 shows the Value Partition Ontology.

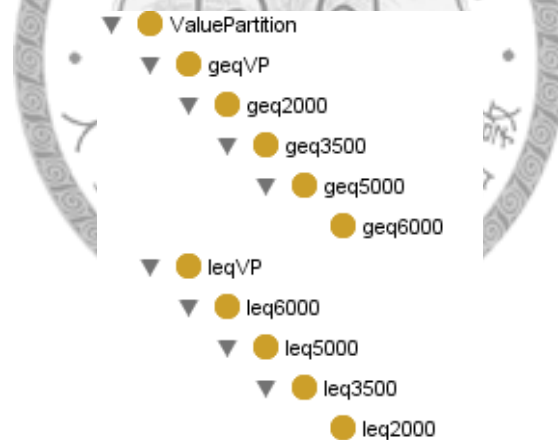


Figure 4.9: The ValuePartition Ontology

4.6 Ontology Maintenance

Ontology maintenance is an important issue in Semantic Web Applications. To correctly represent and reflect the real world, concepts in Ontologies need constant updates and maintenances. But it is a time-consuming job for those ontologies administrators. Because most existing editing tools like Protégé are stand alone desktop applications which lacks of collaborative work. Besides, involving ontology engineering specialist is very expensive, most of ontology maintenance in current Semantic Web Applications are controlled by a small group of people. Through this traditional maintenance approach, a small group constructs the ontology for a bigger group has several drawbacks [41]. First, the addition or update for new concepts can be time-consuming and lack of completion. For example, missing concepts cannot be added by any user who reveals the need for a new concept, but has to be added by the small group of creators. Second, the ontology creators read the concept in the different manner from how the potential user does. Despite a formal language like Description Logics can precisely grasp the meaning of concepts, ontologies creator may misunderstand the meaning of needed concept that described in natural language by potential users. It leads to the problem that the creator add the unnecessary concepts to the ontologies. Sometimes concepts are becoming obsolete by the time they enter the ontologies.

Therefore, the managements of ontology need to be more efficient and more organized so that machines can use those plentiful and correct ontologies in reasoning tasks to deduce the right results. In the long run, ontology maintenance cannot rely on small groups. It should be collaborative task that everyone who use the system can participate in ontology maintenance.

4.6.1 Wiki-supported Ontology Engineering

Ontology Engineering is usually not supposed to be a one-time activity of an expert committee, but rather a sustainable process of continuous evolution [53]. That means the traditional maintenance approach, controlled by small group, are not flexible and agile enough for current Semantic Web applications.

[41] provides another approach to ontology maintenance. They borrow the idea of

community from Web 2.0 era. Using Wiki as a platform where everyone can share information and collaboratively maintain ontologies. Different from the traditional maintenance approach, missing concepts can be added by any user who reveals the need for a new concept, it decrease the duration of addition and make maintenance process more efficient.

Wiki is a popular knowledge management tool widely adopted on the Internet. The basic idea is to use a Wiki as a mechanism to:

- **Concepts Creation**

Any user can create an URI for any needed concept.

- **Concepts Annotation**

Users can describe the concept using natural language and probably multimedia elements such as, pictures, videos, rather than the formal and complexity Logic language. Even Potential users can understand the meaning of concepts.

- **Concepts Refinement**

Wiki technology provides comprehensive version control and edit tools. Refining and modifying the definition of concepts cab be convenient for the user.

- **Collaborative maintenance**

Ontologies administrators can release a part of ontologies to Wiki. Give the user more power to modify and refine ontologies.

In [41]], they show that standard Wiki technology can be easily used as an ontology development environment for named classes. It supports the user's participation in the creation and maintenance of lightweight ontologies. And also they prove that the URIs of Wikipedia entries are surprisingly reliable identifiers for ontology concepts. In Semantic-based Web Services composition architecture, we add the a new component based on Wiki, called Community Component. In addition, we proposed a complete Ontology Maintenance Procedure to support ontology maintenance. We believe that will increase the efficient of maintenance in a long run.

4.6.2 The Model of Ontology Maturing

In [41] and [52], they have made some observations about how new ideas develop in the contexts of knowledge management. In [41], this development process was divided into five abstract phases as the so-called knowledge maturing process. This process is viewed as a macro model for interconnected individual learning processes. Detailed definition of the process are list sequentially below.

- **Emergence of Ideas**

Emergence of Ideas is the first step in the knowledge maturing process. In this initial transition, new concept ideas are introduced which are informal and not well-defined. Most of the time, they are personal expression which are informally communicated and typically represented by tags. Accordingly, we introduce a new tag or correct the existing one without further reflecting.

- **Consolidation in Communities**

The second part of knowledge maturing process is called Consolidation in Communities. Through reuse and adaption of concept symbols, a shared vocabulary emerges within a community. When comparing currently envisioned concepts with previously used ones, we discover similarities and differences that allow for creating concepts or accepting existing concepts. In this stage, the cognition from people in communities will gradually consolidate the new concepts idea or just refuse them. But these preliminary concepts are still without formal semantics.

- **Formalization**

Within the third phase, new concepts are organized into hierarchical construction or other taxonomies. For instance, we need the hierarchy for subsumption inference. Subsumption inference we applied in our system is based on the subconcept relations. In formalization stage, we have to decide that the new concepts, such as new location, a new tourism spot, or a new advertisement, belongs to which super-concept(categories) in ontologies.

- **Axiomatization**

The last phase of knowledge maturing process captures more domain semantics

by adding background knowledge for improving inferencing processes. This step requires a high level of competence in logical formalism, such as Description Logics. Therefore, this can usually only be carried out by domain experts or system administrators.

According to the model of knowledge maturing process, we believe Wiki-based community efficiently support the first two phases of the knowledge maturing process. So that we add Wiki community component to our system. Through collaboratively editing feather of Wiki, every user is able to publish a new concepts ideas. Everyone in the community will notice the emergence of those new ideas. They can discuss and compare new ideas with others. Gradually, the new concepts are consolidated by those people who involve in the community. In addition, Wiki-based community give the system administrator an effective suggestion that what concepts should be update in the ontologies. That makes the maintenance of the ontologies faster and flexible. Up-to-date concepts can be add to the ontologies immediately. We will introduce the ontology maintain procedure in detail at next section.

4.6.3 Wiki Community Component and Ontology Maintain Procedure

Wiki Community Component is one of the new components in Semantic-based Web Services composition architecture. We borrow this idea from web 2.0 technology to empower the individual to take part in community activities by lowering the barriers. It is responsible for creating a open Wiki-based environment which allow the customer to contribute their requirements and corresponding match results and to tap the collective intelligence of a community. Most importantly, Customers are able to participate in light-weight ontology maintenance through Wiki-based editing tools. Our Wiki Community Component indirectly help the administrator to maintain ontologies. In other word, it summary the administrator a guidance from the customer' preliminary concept creation.

To collaboratively maintain the ontologies, we proposed an Ontology maintenance Procedure. That maintenance procedure was divided into five phases. Detailed definition of the procedures are list sequentially below.

- **Ontology Meta Model Definition**

At this first phase, the administrator has to define an ontology meta model suitable for a large audience. Our principle for meta model is less complicated is better. Many ontologies have a subsumption hierarchy that allows to infer implicit class membership, but this is not mandatory to show the whole ontologies to the customer, especially to those users who do not have any logical formalism background. With simplifying the existing ontology to light-weight ontology, the customer can easily understand the structure of ontology and participate actively.

- **Light-weight Ontology Release**

Within the second phase, the administrator releases the simplified ontology to Wiki-based Community Component. The scheme of simplified ontology are consistent with the meta model defined at previous phase. Light-weight ontologies released on the Wiki can be annotated with nature language, pictures, videos, or other multimedia elements. For a large participators, these easy-to-understand, media-rich, Wiki-based user interface lower entrance barriers of collaborative ontology engineering.

- **Preliminary Concepts Creation and Modification**

During the third phase, through Wiki platform, the customer freely modifies existing concept or creates new concepts according with the released ontology meta model.

- **Preliminary Concepts Consolidation**

The forth part of ontology maintain procedure is a evolution process that needs participators to accept or reject the preliminary concept idea through continuously reuse and adaption of concept symbols. If they are tend to have a common cognition toward the new concepts, those new concept ideas will be gradually consolidated and be well formed. For instance, those stable concepts in Wiki are rarely modified by others because participators accepted the current definition. In contrast, the controversial concepts are always have many different version in history log.

- **Ontology Update**

The last phase, Ontology Update, which has to involve the administrator to update the ontology directly through manipulating KBMS, such as Protégé. The Wiki Community Component indirectly provide the administrator a guidance that suggest what concepts should be updated or created. With Wiki Community Component and the user's participation, ontology maintenance becomes more effective.



4.7 Service Execution

4.7.1 Design of the Service Execution Module

The Service Execution Module in the Service Composer is activated when the customer chooses matched services to execute them. It handles the execution part of the architecture and is the bridge between the Integrated User Interface and the Execution Engine. After the user choose one of the desired services and ask the Service Composer to invoke it, the related information about the services is sent to the Service Execution Module. That contain several component: the WS-BPEL File Handler, the WS-BPEL Template Pool, the WS-BPEL Generator, and the Execution Engine Invoker. At first, the WS-BPEL File Handler starts to yield a WS-BPEL file using the WS-BPEL Generator. The WSDL Parser in the WS-BPEL Generator obtains related WSDL files from the Internet and extracts the necessary information and parameters. The WS-BPEL Generator obtains the template (Abstract Process) from the WS-BPEL Template Pool and combine with WSDL files into the BPEL file (Executable Process). Finally, the WS-BPEL File Handler then deploys the BPEL file to Execution Engine.

The Service Execution Module interacts with the Execution Engine and returns messages about the execution status to the Service Composer. The messages, which will be displayed in the Integrated User Interface, should be clear and detailed. If any incidents occur, the Service Execution Module is responsible for handling them.

4.7.2 Architecture of the Service Execution Module

- **WS-BPEL File Handler**

The WS-BPEL File Handler is responsible for communicating with the Integrated User Interface, handling the BPEL related files, and deploying the BPEL files to the Service Engine. It receives the desired service lists from the user and collects respective service descriptions from the Internet. After it asks the WS-BPEL Generator to generate the executable processes of WS-BPEL, it sets the related configurations of the Execution Engine and deploys these files (WS-BPEL files and the Execution Engine setting) to the Execution Engine. If there were errors in the BPEL File Handling stage, the WS-BPEL File Handler sends corresponding messages to the

Integrated User Interface to notify the user.

- **WS-BPEL Template Pool**

The WS-BPEL Template Pool stores the BPEL templates, which are designed by the system administrator. According to different processes and interaction of services, different BPEL templates are pre-defined using the existing application for creating BPEL processes. These templates are stored as abstract processes and can be transformed to executable processes by adding related parameters. They can be reused to meet the defined composition processes as needed by the invoked services..

- **WS-BPEL Generator**

The WS-BPEL Generator is responsible for generating WS-BPEL files according to the selected templates and related WSDL files. It produces the executable processes from the BPEL template (abstract processes) by adding related parameters extracted from the WSDL files. It uses the WSDL Parser to parse the WSDL files and get the parameters needed for the BPEL files. After it finishes generating the WS-BPEL files, it returns them to the WS-BPEL File Handler.

- **Execution Engine Invoker**

The Execution Engine Invoker acts as a bridge between the Service Composer and the Execution Engine when the user want to invoke services. After the WS-BPEL files are deployed by the WS-BPEL File Handler, the Execution Engine Invoker is responsible for invoking the desired services. If the services are invoked successfully, it sends messages to the user. However, if any Web Services fail, the Execution Engine will rollback all the executions it has done so far and send the Execution Engine Invoker an execution failure message.

Otherwise, the Execution Engine must be bundled into a Web application server. It can provide execution logs via the Web pages so that the system administrator and the user can trace the execution status of the services according to the processes of the WS-BPEL.

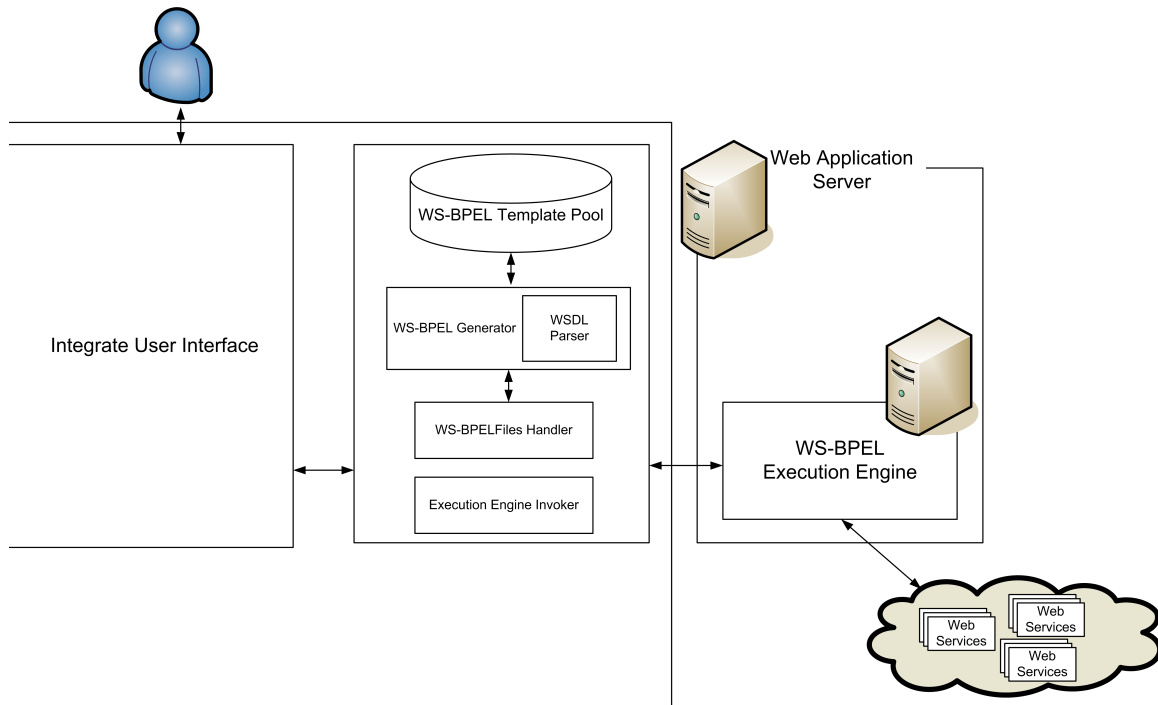


Figure 4.10: Architecture of the Service Execution Module

4.7.3 Development of the Business Process Execution Language

In the service execution stage, BPEL files are generated by the Service Execution Module in the Service Composer. We adopt the methodology for development of Web Service-based Business Processes proposed in [31], and simplify the process to meet the essential needs of the Semantic-based Web Services composition architecture. In Figure 4.11, the service execution involves two stages: the **Build Time** and the **Run Time**. The Build Time refers to the procedure of defining the WS-workflows with the desired characteristics and related settings in the Execution Engine. It involves in three phases: **Preprocessing**, **WS-BPEL Process Generation**, and **WS-BPEL Deployment**. The Run Time refers to the actual **Service Execution** and **Post-run Time**, which monitors the status of the Execution Engine. The detailed functions are as follows:

- **Preprocessing**

In the Preprocessing stage, the agents or the system administrators should participate in and define BPEL templates according to their needs or business processes.

Templates can represent collections of activities that implement composition patterns and activities with specific features. The system administrators can use existing BPEL related tools like Active-BPEL Designer to define the processes. They make an abstract process definition and store it in the template pool.

- **WS-BPEL Process Generation**

In the architecture of the Service Execution Module, when the customer selects a suggested advertisement package to invoke, the Service Composer sends a request to the WS-BPEL Generator to process BPEL4WS files. In the WS-BPEL Process Generation stage, the process definition program representation is generated. After collecting the related WSDL files of the advertisements from the Internet, the selected template (abstract process) is combined with the related parameters and definitions, and a BPEL file (executable process) is generated. Besides, the related files about the Execution Engine are made in this stage.

- **WS-BPEL Deployment**

When all the BPEL files and the setting files about the Execution Engine are completed, they are packaged and deployed to the Execution Engine by the WS-BPEL Deployment. The deployed BPEL files become a service process in the Execution Engine and ready to be called.

- **Service Execution**

Service Execution executes the deployed processes according to the execution order scheduled by the process control flow. During executing, data is exchanged between processes and the invoked Web services.

- **Post-run Time**

It would be useful to gather the status information of the execution during run time. In this stage, the system administrators monitor the execution, analyze the process logic, and configure it by checking the status information.

The BPEL life-cycle provides a semi-automatic development process for BPEL. Humans only have to participate in the first stage, Preprocessing, because business logics and process flows should be defined by the user. The rest of stages in BPEL life-cycle can

be implemented automatically. The process model can shorten the development time of processes and also hide the complexity from the developers because humans do not need to join every stage. In addition, the model provides flexibility by postponing the choice of a language for the definition and by deferring the binding to the specific Web Services to the latest possible time.

Service composition correlates highly with service execution because the order of composed services is considered as the same as the order of service execution. The information of the service composition details what services participate in the process and the correct execution order. In [63], the authors note that there are many types of the service composition, such as sequential service composition, sequential alternative composition, parallel with results synchronization, and parallel alternative composition. Sequential service composition means that the execution of a constituent service is dependant on its preceding service. Sequential alternative composition means that alternative services could be part of the composition. Every alternative will be attempted until anyone service succeeds. The parallel with results synchronization means that the constituent services can run concurrently. However, the results of their execution need to be combined. The parallel alternative composition means that alternative services are pursued in parallel until one service is chosen. Each of them is classified by the flows of involved services.

Based on the Semantic technologies, the service composition will be accurately composed by matching with the customer's requirements. Furthermore, Semantic technologies support Web Services with providing accurate suggestions to dynamically and automatically invoke the services. Besides, different orders of service execution are designed according to the customer's needs and business process logic.

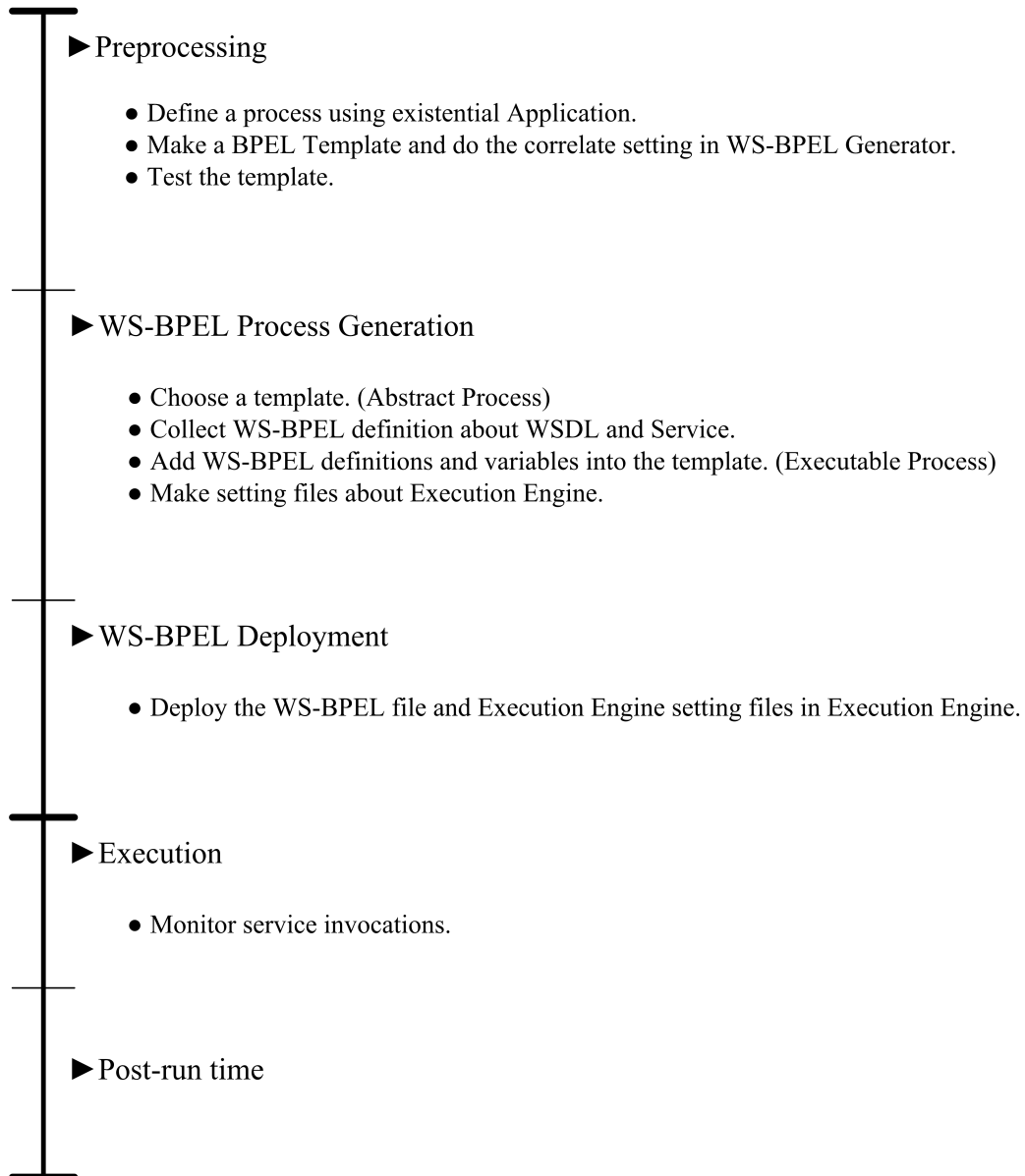


Figure 4.11: Life-cycle of the Business Process Execution Language

Chapter 5

Implementation - The Traveller

5.1 The System Design

To validate the Semantic-based service composition architecture described in the previous section 4.1.2, we have implemented a prototype system for the tourism domain, called **The Traveller**, as a web application based on existing Web Services, semantic web, and Web 2.0 technologies. The Traveller obeyed the service composition architecture that contains each essential components we introduced in previous chapter. The reason why we chose the tourism industry domain to be our analytic target is that the relationship between the customer's trip requirements and the providers' trip advertisements is conspicuous for observation.

In general, the Traveller provides an integrated tourism service that includes the customer's requirement defining, approximate matching, service invocation. Through mouse clicking in the browsers, the customer can define the requirements and match for suitable Web Services.

The Traveller was originally designed in [27], which was implemented as a plug-in application in the Protégé, a well-developed ontology management system developed by Stanford Medical Informatics at the Stanford University School of Medicine¹. However, as a Protégé plug-in there is a big problems that is those ontology tools and ontology languages impose high entrance barriers for potential users. While using previous system, the customer often had information-overload problem. So that we decide to divide the system view into customers' perspective and administrators' perspective. At this time,

¹<http://Protege.stanford.edu/>

we have a lot of improvement on providing a friendly system interface and encapsulating complex semantic language into figures and pictures which are easy to be understood by the customer. To achieve that, we spent a lot of time on transplanting our system from Protégé to an open J2EE web application. Being a web application, the customer does not have to install Protégé or related plug-in modules, such as SWRL tab-widget, Protégé Axiom Language (PAL) tab-widget, and Racer inference engine. All they need is an accessible Internet and a browser. With combining different plug-ins, Protégé plays an important role of being a useful management tool for administrators. That contains 1) SWRL tab-widget with the JESS rule engine, which is used to define SWRL rules and examine role relationships between individuals in the ontologies; and 2) the Protégé Axiom Language (PAL) tab-widget, which helps the global constraint checker analyze the integrity constraints.

To demonstrate our system, we take a scenario in the tourism domain as an assumptive example. That is a customer wants to plan a trip for two days from Taipei to Nantou in Taiwan. We assume that he/she does not know what exactly places to visit or which hotel to accommodate. As a Semantic Web application, the Traveller can aid the customer plan their trips. The Traveller adopts Google Maps API as the AJAX interface, that helps the customer indicate their starting point and destination by putting the flag on the Google Maps. With stating point and destination, a basic trip requirement is initialized. Also the customer could input detailed information more than locations, such as the customer's budgets, date, numbers of people, and spots which they want to visit. The Traveller guides the customer to fill-in necessary information step-by-step. If input data are inconsistent with what we expect, such as data type mismatch, a warning window will jump out that reminds the customer to correct their inputs. After completing requirements, the Traveller translates the requirements into formal logic language at the back-end. Subsequently, it starts making exactly or approximately matching services according to the customer's preferences. Finally, the matched result will be returned, the customer can make a decision to invoke the Web Services on the result list such as flight booking services and hotel reservations.

Based on the above scenario, there are the four stages in the Semantic Web Services composition architecture, which we describe belows:

- The system collects the advertisements of potential services from UDDI registry or the related web sites.
- Users define their requirements and complete the service description with the help of the system.
- The system matches the appropriate advertisements of the services according to the user's requirements.
- The user selects the desired package of advertisements from the matched advertisements and asks the system to invoke them.

In the following section, we introduce the implementation system and explain the service descriptions of requirements and advertisements, ontologies, constraint checking, and rules in the system.

5.2 Service Description

To accommodate to the matching scheme, the descriptions of the customer's requirements and providers' advertisements are defined in the same form, Description Logics. In the Semantic Web Services composition architecture, we adopt concepts to express service description which contain constraints (features/attributes) about the trip service, like trip price, starting point, destination, and date. We use the \sqcap constructor to connect these constraints together. With concepts expression, we can specify a service requirement or advertisement which hold all constraints at the same time.

In the matching stage, a suitable matched service must satisfy all or partial specified constraints. From customers' perspective, a requirement description represents a customer's demand and expectation about the service. From a service provider's perspective, a advertisement description characterize the functionality service provides. Note that the service description we mention here is an "abstract description" which detailed the functional descriptions other than execution details in the service profile of WSDL document. Travel agents can publish their advertisements using OWL-DL based service descriptions or traditional existing database schema. In the back-end, those service profiles published on the Internet are automatically discovered and stored by the Service

Collector/Annotator, one of the main component in our system. If the travel agent adopts different kinds of service descriptions to those of the ontologies in our system, the Service Collector/Annotator is responsible for translating the heterogenous descriptions into the same format. Those abstract descriptions will be mapped into OWL-DL concepts so that service properties are restricted with specific constraints. In front-end, the Service Composer accept and store the customer's requirements using the shared set of ontologies for matching with abstract service descriptions. Following the scenario mentioned above, we explain the service descriptions in the Traveller system.

5.2.1 Trip Requirement Description

According to the design approach of the ontologies discussed in Section 4.4, in the scenario, the Traveller accepting a requirement, a two-day trip from Taipei to Nantou from July 1st to July 3rd, 2007, from a customer. The customer is seeking for a trip for two people and trip budget around NT.10,000. The customer plan that the first day, July 2nd, 2007, start from Taipei to Hsinchu, and second day, July 3rd, 2007, start from Hsinchu to Nantou County by bus. We defined the general trip as the concept, **MyTrip**, which shows below. **MyTrip** concept can be segmented into two days represented by **MyRequirement-1** and **MyRequirement-2**. The bus requirement can be defined as a transportation requirement called **TransReq1**. Besides, the customer has stated preference about the tourism spot named Ching Jing Farm², and its budget is about NT.1,000. The tourism spot requirement in July 3rd. is described by the concept, **SpotReq1**.

$$\begin{aligned}
 \text{MyTrip} &\equiv \exists \text{tripStartDate.2007-07-01} \\
 &\sqcap \exists \text{tripEndDate.2007-07-03} \\
 &\sqcap \exists \text{startsFrom.Taipei} \\
 &\sqcap \exists \text{endsAt.NantouCounty} \\
 &\sqcap \exists \text{hasNumberOfPeople.\{2\}} \\
 &\sqcap \exists \text{hasTripBudget.leq10000} \\
 &\sqcap \exists \text{hasTripElement.MyRequirement-1} \\
 &\sqcap \exists \text{hasTripElement.MyRequirement-2}
 \end{aligned}$$

²Ching Jing Farm is a famous farm in Nantou County, Taiwan.

MyRequirement-1	≡	∃ tripStartDate.2007-07-01
	⊔	∃ tripEndDate.2007-07-02
	⊔	∃ startsFrom.Taipei
	⊔	∃ endsAt.Hsinchu
	⊔	∃ hasNumberOfPeople.{2}
	⊔	∃ next.MyRequirement-2
MyRequirement-2	≡	∃ tripStartDate.2007-07-02
	⊔	∃ tripEndDate.2007-07-03
	⊔	∃ startsFrom.Hsinchu
	⊔	∃ endsAt.NantouCounty
	⊔	∃ hasNumberOfPeople.{2}
	⊔	∃ hasBudget.leq6000
	⊔	∃ hasTransTE.TransReq-1
	⊔	∃ hasSpotTE.SpotReq-1
TransReq-1	≡	∃ hasTransDepartDate.2007-07-02
	⊔	∃ hasTrans.Bus_AoWanTa_ChingjingFarm
	⊔	∃ hasNumberOfPeopleTrans.{2}
	⊔	∃ hasTransBudget.leq2000
SpotReq-1	≡	∃ hasSpotScheduledDate.2007-07-03
	⊔	∃ hasSpot.CingjingFarm
	⊔	∃ hasNumberOfPeopleSpot.{2}
	⊔	∃ hasSpotBudget.leq1000

5.2.2 Service Advertisement Description

Travel agents publish their advertisements using OWL-DL based service descriptions or traditional existing database schema. If the travel agent adopts different kinds of service

descriptions from the ontologies we use, the Service Collector/Annotator is responsible for translating the heterogenous descriptions into the same format. The concept listed below represents a two-day trip advertisement for two persons from Hsinchu to Nantao for the period July 2nd to July 3rd, 2007, which advertised price of the trip is NT.5,000. In addition, it also include a spot itinerary, called **SpotAd0**. The spot itinerary is for a ChingjingFarm spot scheduled on July 3rd.

AdvertisementNantou-0	≡	∃ tripStartDate.2007-07-02
	⊃	∃ tripEndDate.2007-07-03
	⊃	∃ startsFrom.Hsinchu
	⊃	∃ endsAt.NantouCounty
	⊃	∃ hasNumberOfPeople.{2}
	⊃	∃ hasBudget.leq5000
	⊃	∃ hasSpotTE.SpotAd-0
SpotAd-0	≡	∃ hasSpotScheduledDate.2007-07-03
	⊃	∃ hasSpot.CingjingFarm
	⊃	∃ hasNumberOfPeopleSpot.{2}
	⊃	∃ hasSpotBudget.leq500

5.3 Implementation of the Traveller

In this section, we will explain the components of the Traveller in detail. The components include the Service Composer, the AJAX Component, the Community Component, the Inference Engine, the Execution Engine, the Service Collectors/Annotator, the Knowledge Base Management System, Ontologies. We have mentioned the function of each component in section 4.1.2. Somehow, the Service Composer and KBMS play the most two important roles in the architecture. The former interacts with the user in the front-end by providing an user-friendly interface and collaborates with the other components in the back-end by accessing the set of the ontologies. The later one monitor and manage the correction of the ontologies with SWRL rule and Protégé Axiom Language by manipulating Protégé and its plug-ins. Therefore, we focus more implementation detail

on those two components. We adopt JSP/Servlet technology, which is a Java-based web application, as the Service Composer in the Traveller. To lower the entrance barrier of the semantic application, we have modified the functionality of the Service Composer which is simpler than the previous version.

Based on the Semantic technology environment, we use the Web Ontology Language (OWL-DL), a family of knowledge representation languages for authoring ontologies, adored by the World Wide Web Consortium, to describe the service descriptions and related information and use the Semantic Web Rule Language (SWRL) to increase the inference power for solving the role composition. To implement the Inference Engine, we adopt the Racer DL reasoner as the back-end Inference Engine to perform concept subsumption reasoning. Otherwise, we adopt the Active-BPEL Engine³ as the Execution Engine, which attaches to the Web Application Server of Tomcat 5.0⁴. For the Service Collector/Annotator, we apply the approach proposed in [14]. Here, we detail the implementation of each component in the Semantic-based service composition architecture. Table 5.1 illustrates the implementation tools and the corresponding components.

Table 5.1: Implementation of the components of the Traveller

Components	Implementation
The Service Composer	JSP/Servlet
The AJAX Components	Google Maps API
The Community Components	JSPWiki
The Inference Engine	Racer
The Execution Engine	Active-BPEL Engine
The Service Collector	Java-application
The Knowledge Base Management System	Protégé 3.11
Ontology Standard	OWL-DL
File System	MySQL 5.0
Application Server	Tomcat 5.0

³<http://www.active-endpoints.com/active-bpel-engine-overview.htm>

⁴<http://tomcat.apache.org/>

5.3.1 Implementation of the Service Composer

- **Inner Knowledge Base**

Knowledge Base is represented by Web Ontology Language(OWL), a language for defining Web ontologies. We adopt Protégé API to access the OWL file which contains numbers of concepts and relative rules. Regarding the trade-off between the expressive power and the computational complexity, we adopt OWL-DL as the specification for our system's ontology. The requirement that the customer input would be translated into OWL format and stored at ontologies. To encapsulation the complex semantic information, we did not authorize the customer any power to modify the ontologies. All related management tools are removed from the Service Composer.

- **Integrated User Interface**

As we state previously, the Traveller is web application based on Semantic Web technologies. To provide the interface on pages, the Integrated User Interface is implemented by Java Server Pages(JSP), a script language widely adopted on the Internet. It provides a user-friendly interface that helps the user complete their trip requirements step-by-step, and return the suitable service on pages. Most importantly, the Integrated User Interface mashing up with Google Maps API that provide a interactive AJAX GUI to the customer. In addition, it connects other component such as the Wiki-based community through hyper-links.

- **AJAX Module**

With cooperate with the AJAX Component, AJAX module is responsible for building a geographic interface, for instance, a Earth map, displayed on the Integrated User Interface. While displaying the geographic interface, AJAX module retrieve necessary information from the ontology, such as location name, descriptions of locations. Meanwhile, it continuously listen the interaction events between the customer and the system, such as mouse click, mouse trajectory. Those events are used to trigger certain functions. For example, Through clicking mouse on the map, the customer inputs their requirement easily.

- **Community Module**

Community Module is implemented as a Java program, that dynamically generate the document for JSPWiki. The document contains the customer information, their requirements, and corresponding advertisements. Also if the customer input a requirement that contain some information not in the ontologies, it generate a report document to the administrator. The documents generated by the Community Module are used to be deployed on JSPWiki. So that JSPWiki can display the needed information on the collaboratively editing environment.

- **Matchmaker**

The Matchmaker is a matching module that invokes the Inference Engine to start reasoning. It acts as a bridge between the Service Composer and the Inference Engine To provide an approximate service matching between requirements and advertisements, we developed a Java program as a matchmaker module to communicate with the Inference Engine, Racer inference engine. This Java Program is responsible for computing similarity degree. Also numbers of domain rules for decision are implemented in the code, such as the similarity between transportation lines, location concepts substitution rule.

- **Dynamic Concept Component**

To solve the problem of quantitative concepts, we use a set of Java programs to dynamically create quantitative concepts like Time and ValuePartition. When requirements related values and time, this Java programs will adds automatically time or value concepts to the knowledge base. That implies relationship between concepts and the concept hierarchy are created automatically and dynamically. to avoids having to define a large knowledge base of quantitative concepts in advance,

- **Service Execution Module**

We develop this Service Execution Module combining the following components:WS-BPEL File Handler, WS-BPEL Template Pool, WS-BPEL Generator, and Execution Engine Invoker. It provides tools to generate executable processes from abstract service descriptions and templates, and invokes the Execution Engine. It also returns the results from the Execution Engine to the Integrated User Interface.

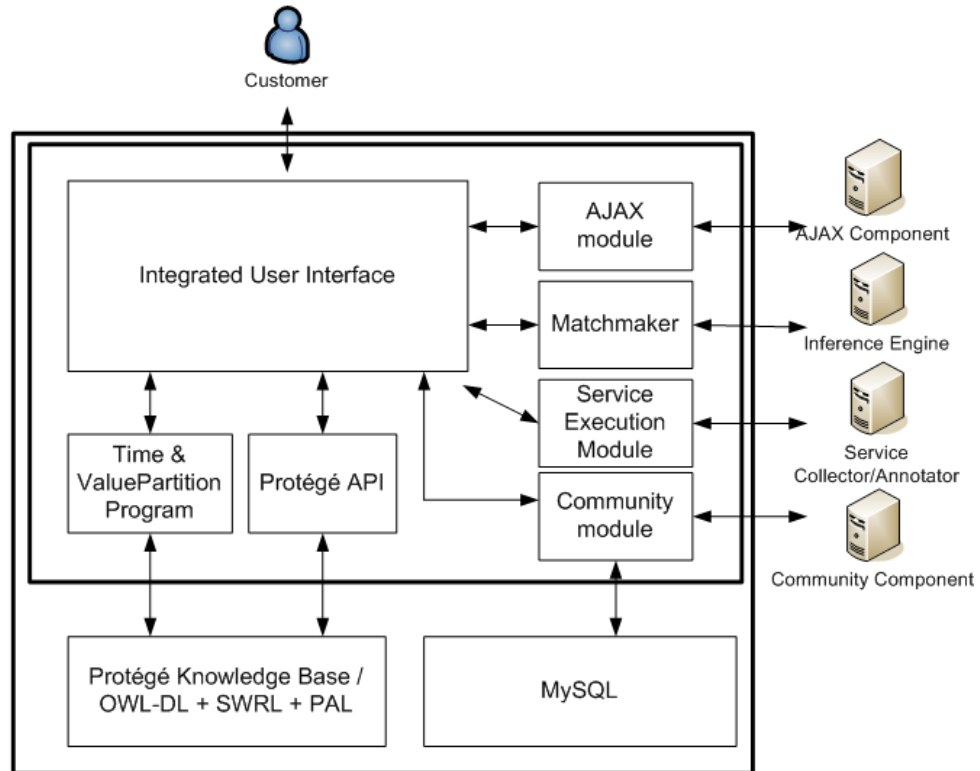


Figure 5.1: The Implementation of the Service Composer

Figure 5.1 illustrates the implementation of the Service Composer in the Traveller.

5.3.2 Implementation of the Knowledge Base Management System

- **Inner Knowledge Base**

Inner Knowledge Base we mentioned here is exactly the same ontology as the System Composer's one. It is represented by Web Ontology Language (OWL). We use Protégé OWL editor to define and maintain the tourism domain ontology. Only administrators can modify and update the ontology at back-end. To increase the ontology management capability, we also adopt Semantic Web Rule Language (SWRL) and Protégé Axiom Language (PAL) rules to support maintenance of ontology.

- **Integrated User Interface**

Protégé ontology editor provides an Integrated User Interface for the administrator.

With that, creating or modifying a class, property, instance is very convenient. In addition to OWL editing, Integrated User Interface integrate other Protégé plug-in tools such as, The Protégé Axiom Language tab-widget plug-in and the Protégé SWRLJess Tab-widget plug-in. SWRL rule and PAL also can be defined in Integrated User Interface.

- **Translation Tools and Rule Engines**

We adopt Protégé SWRLJess Tab plug-in for editing and reasoning rules. SWRL rules are used to strengthen the expressive power of OWL-DL in our system, and to capture implicit composition relationships. The SWRLJess plug-in translates SWRL rules to JESS rules, which can infer new knowledge through the Jess rule engine. It returns the inferred knowledge and stored it in OWL-DL format in the knowledge base.

- **Constraint Checker Tools**

To check the consistency of the ontology, we use the Racer DL reasoner to handle the binary relationship between two constraints by subsumption inference(classification). Besides, we use the Jess Rule Engine and SWRL rules to solve the global relationship among more than two non-quantitative constraints. Otherwise, we adopt the PAL rule engine provided by Protégé Axiom Rule plug-in to check the global relationship among quantitative constraints. These Constraint Checker Tools are used to check constraints like trip budgets and time dependencies between the requirement and its element requirements.

- **Import/ Export Component**

We use the Protégé OWL plug-in to exchange knowledge base with other Semantic Web applications . Ontologies will be stored as OWL files through the Protégé OWL editor plug-in. With OWL standard, Semantic Web applications are able to import their ontology into our knowledge base. In contrast, we can export our ontology in OWL format to others.

Figure 5.2 illustrates the implementation of the Knowledge Base Management System in the Traveller.

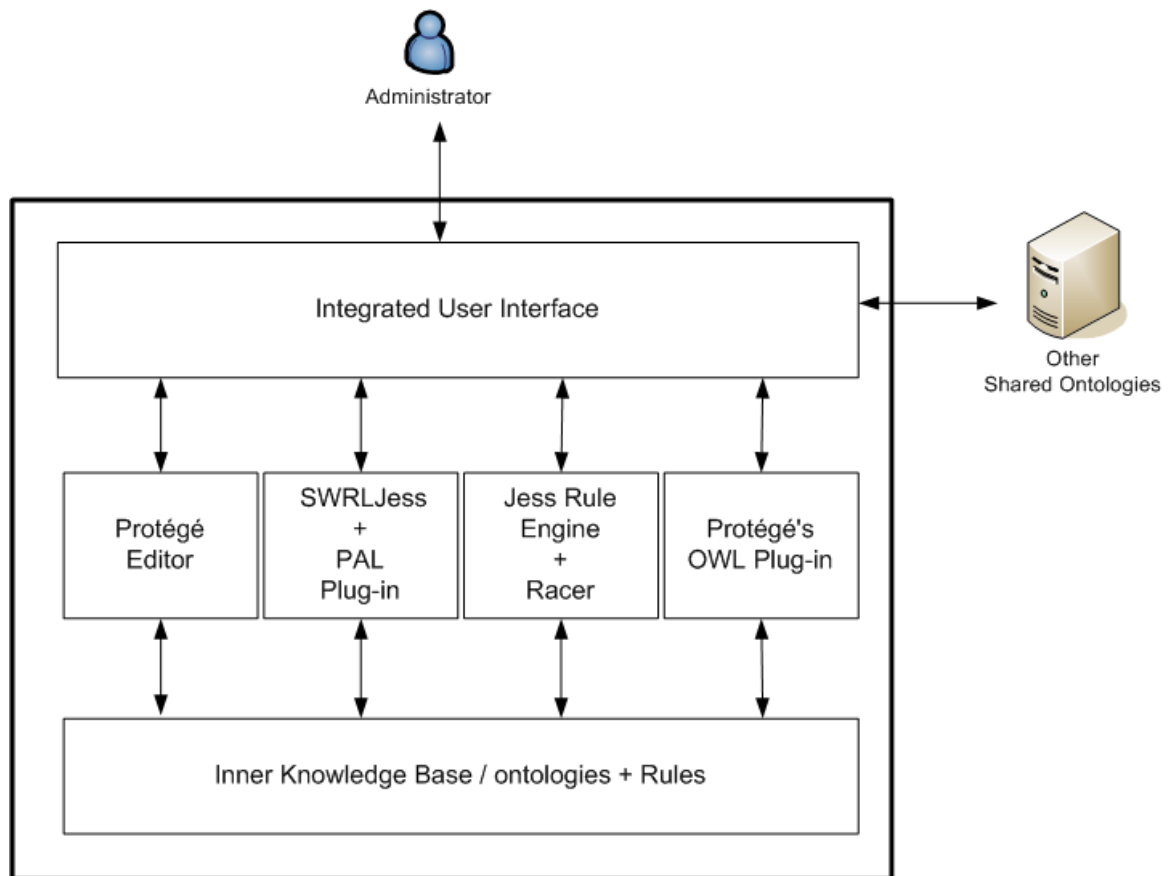


Figure 5.2: The Implementation of the Knowledge Base Management System

5.4 Ontology Design

5.4.1 The Tourism Domain Ontologies

According to the architecture of the service combination approach mentioned in the 4.4, ontologies play an essential role for different components in the architecture to communicate with each other. They make computers able to exchange information with each other in the semantic-level understanding instead of only in the syntactic-level consistency. In the following sections, we introduce the ontologies design in implementation system, the Traveller, and explain the consideration and purposes. Following the architecture of the ontology in [27], we attempt to distinguish two different kinds of ontologies in our system. First, there are upper common ontologies, which can be reused in different domain, such as time, and Value Partition, to describe the constraints like time and budget. Second, there are domain-related ontologies for describing requirements, advertisements, accommodation, transportation, event, spot, and location. Figure 5.3 shows an overall tourism ontology design. We will specify these ontology design details in the following sections.

5.4.2 The Spot Ontology

As we mentioned above, domain-related ontologies are those description for requirements, advertisements, accommodation, transportation, event, spot, and location. In this section, we introduce the spot ontology which ontology hierarchy are similar with event, accommodation, and transportation. Therefore, after introducing the spot ontology, readers also understand the ontology design of event, accommodation, and transportation. We assume there is a Web Service, a spot ticket booking service, contains functionality descriptions, such as input and output message definitions, and non-functionality descriptions, such as service provider information. In the service composition architecture, we use a concept(class) to define the spot service. The non-functional service descriptions are represented as properties of the concepts. Figure 5.4 illustrates the **Spot** ontology. In the figure, a class called **Spot** is used to represent a root class of a spot ticket booking Web Service. It contains numbers of properties to represent the descriptions of spot services. For example, a spot class has its name, its Web site URL, e-mail address, telephone number, fax number, address, corresponding WSDL description, and its location. We

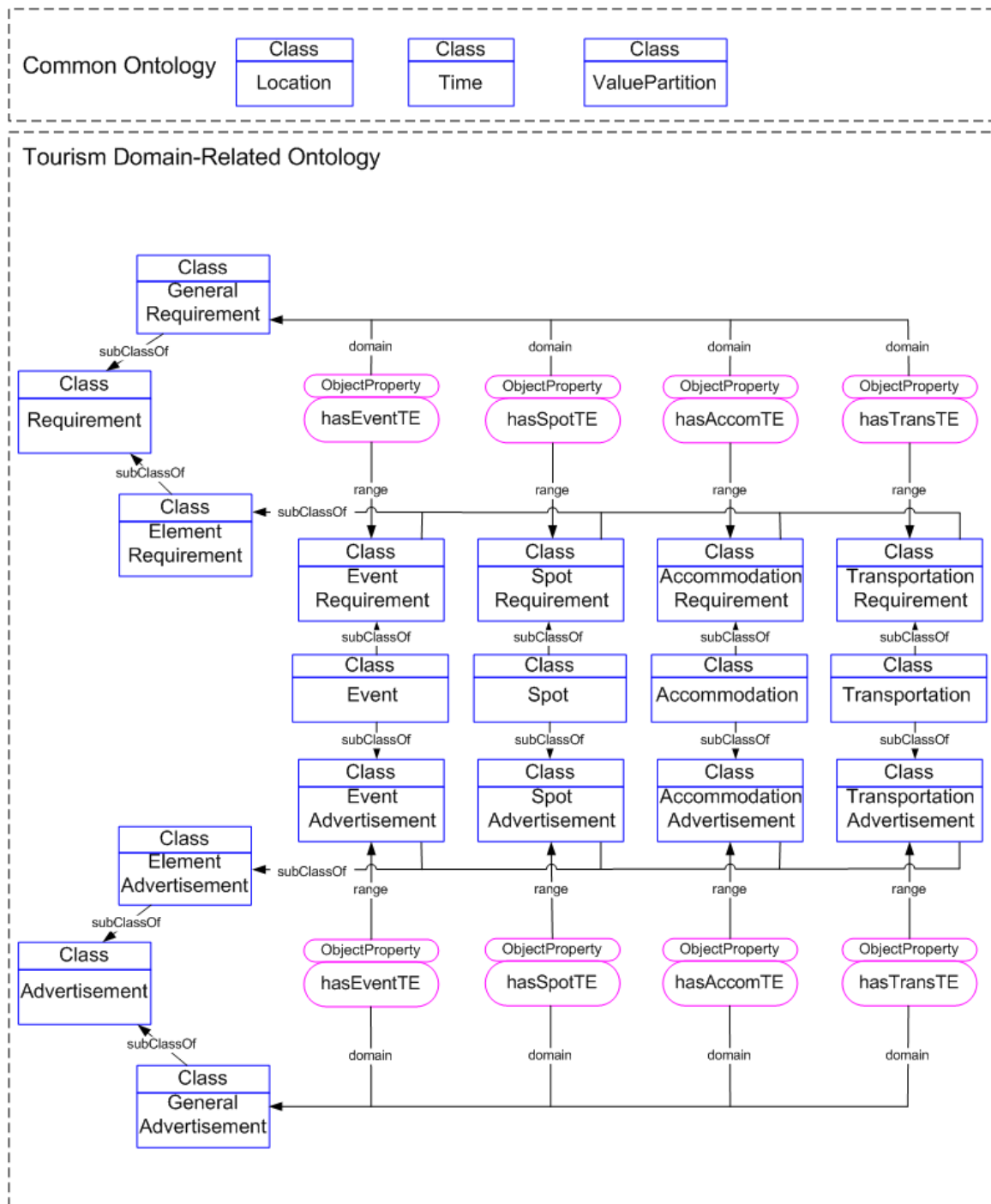


Figure 5.3: The Tourism Ontology Design

divide spots into four categories: cultural spots, general spots, natural spots, and temple spots. The classifications CulturalSpot class, GeneralSpot class, NaturalSpot class, and TempleSpot class respectively denote the types of spots.

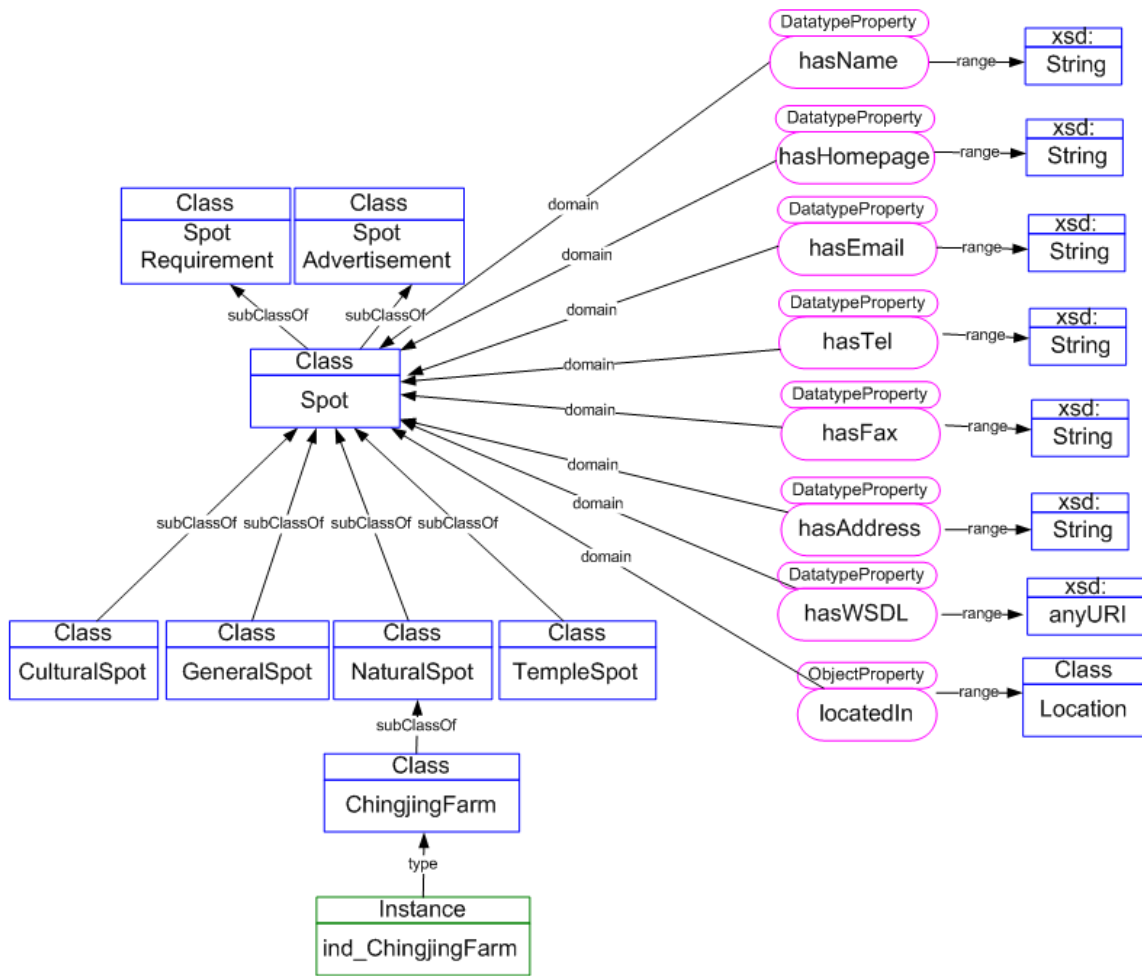


Figure 5.4: Design of the Spot Ontology (Part)

Take Ching Jing Farm, one of the famous spots in Taiwan, as an example. We construct a **ChingJingFarm** concept to represent that spot. We classify the **ChingJingFarm** as a natural spot, so it should be a subclass of the **NaturalSpot** class. Also we build an individual **ind_ChingJingFarm** to represent **ChingJingFarm** concept in ABox level. As far as we know, there is no role composition inference in TBOX level. That's why we have to build an additional individual to capture the composition relationship between different concepts through SWRL rules. **Ind_ChingJingFarm** contains the corresponding information according to the range of property defined in the spot ontologies. For example, if **hasName** property's range is String, then a String type information

need to be filled in hasName property. **Event**, **Accommodation**, and **Transportation** ontology are modeled in the same approach.

In Figure 5.4, the **Spot** is the subclass of **SpotRequirement** and **SpotAdvertisement**, so we can facilitate the subsumption relationship in the TBOX level reasoning between the requirements and the advertisements of the spots. We can also capture the relationship by individuals of the Spot concept in ABOX level reasoning in Figure 5.5.

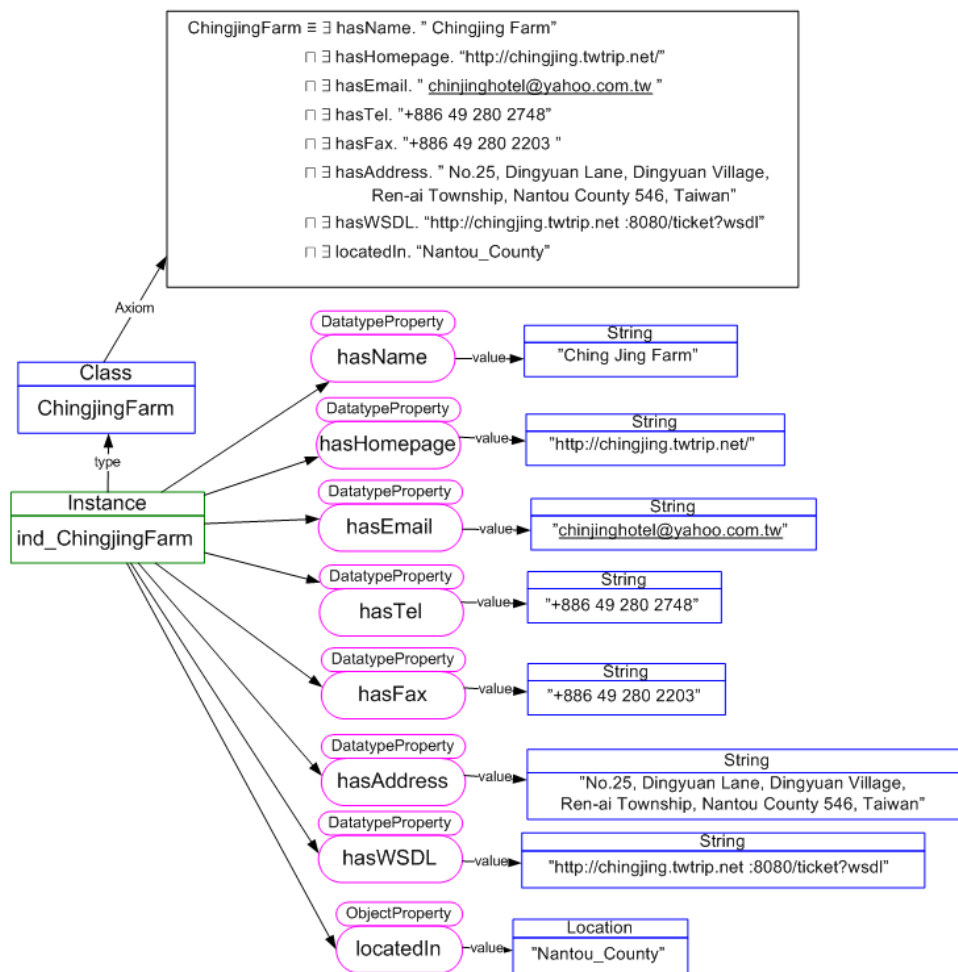


Figure 5.5: The Ching Jing Farm Service Profile

5.4.3 The Requirement Ontology

In the service composition architecture, a requirement description stands for a customer’s needs. A requirement description can be considered as a abstract description from Web

Services perspective. The **Requirement** ontology is used to represent such requirement descriptions. In contrast, the **Advertisement** ontology represents the descriptions of the services offered by service providers. In the **requirement** ontology, we define a **Requirement** class to express a general requirement. A general requirement consists of many element, for instance, Spot Requirement is a typical trip element. So that we can decompose a compound requirement represented in ontology.

According to the ontology design described in Section 4.4, we have implemented the **Requirement** ontology in our system, as shown in Figure 5.6. The root class **Requirement** represents a set of compound requirements. There are two subclasses of **Requirement** class. First one, the **GeneralRequirement** class, is used to represent a composed requirement which is connecting Event, Spot, Accommodation, and Transportation Requirement through properties. Each general requirement stands for an itinerary. And numbers of general requirements aggregate a complete requirement, called **Trip Requirement**. Second one, the **TERequirement** class, can be divided into four types of trip element requirements as subclasses in our prototype system. They are **AccomRequirement**, **EventRequirement**, **SpotRequirement**, and **TransRequirement**. Accordingly, a **GeneralRequirement** class has four types of properties called **hasAccomTE**, **hasEventTE**, **hasSpotTE**, and **hasTransTE**, and the respective object property classes **AccomRequirement**, **EventRequirement**, **SpotRequirement**, and **TransRequirement** represent the different types of trip element requirements in the tourism domain.

As a trip requirement contains, it contains necessary information like the trip's start date, end date, departure location, arrival location, trip budget, and so on. The trip information we mentioned above are annotated as properties, such as **tripStartDate**, **tripEndDate**, **startsFrom**, **endsAt**, **hasTripBudget**, of the **Requirement** class.

The **TripRequirement** class is used to describe a trip package which has a **hasTripElement** property to connecting GeneralRequirement requirements. With defining **Requirement** as the range of the **hasTripElement**, the **TripRequirement** class is used to describe a trip package connecting GeneralRequirement requirements. These general requirements can be considered as several daily plans included in the trip package. Figure 5.7 shows an example of the Requirement scenario in the Traveller. We assume

Requirement in the Traveller - Class and property definitions:

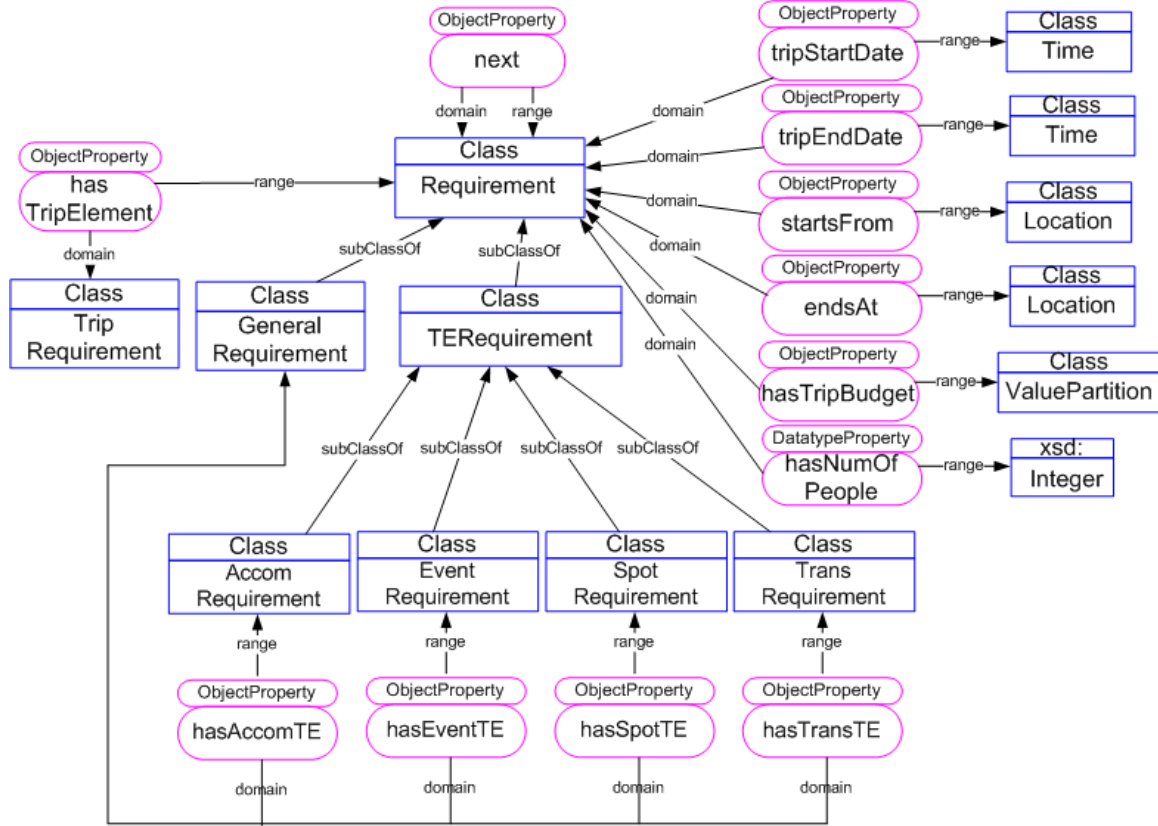


Figure 5.6: The **Requirement** Ontology in the Traveller

that system got a two-day trip requirement. We respectively create two classes called **MyReq_1** and **MyReq_2** as requirements. for the first day and the second day itinerary. The **next** property represents the dependencies of the two requirements. **MyReq_2** has two trip element requirements. The **next** property represents the dependencies of the two requirements. In our scenario, the customer would like to take *bus* to *Ching Jing Farm* at the second day. So we add a spot requirement, called **SpotReq_1** and also add a transportation requirement, **TransReq_1**. These two trip element requirements are connect with **MyReq_2** through properties, **hasSpotTE** and **hasTransTE**. Figure 5.8 shows the concept definitions of **MyReq_2** and its trip element requirements, **SpotReq_1** and **TransReq_2** classes. Figure 5.9 illustrates the implementation of the scenario in the previous sections.

A Requirement example in the Traveller:

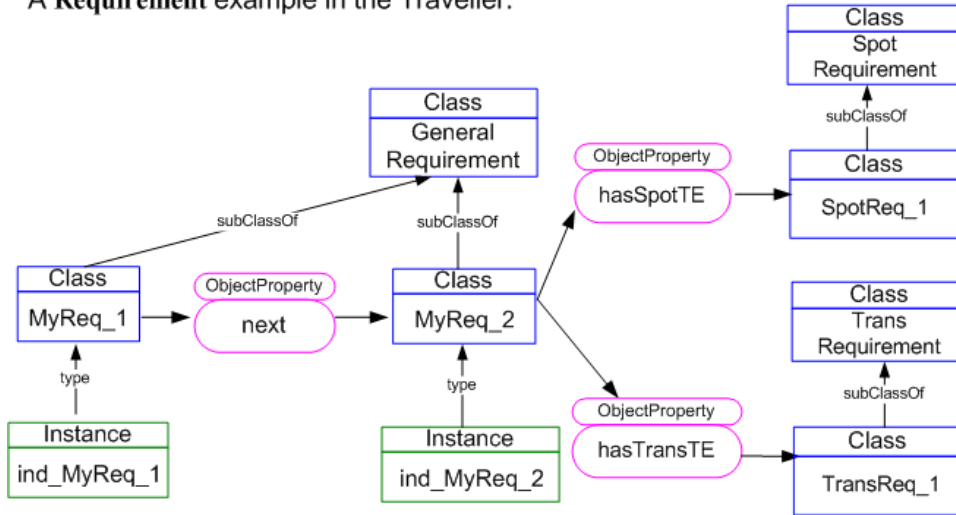


Figure 5.7: The Requirement Example in the Traveller

5.4.4 The Advertisement Ontology

In the service composition architecture, a advertisement description stands for a provider's service. The architecture of the advertisement ontology is identical to the requirement ontology. It is used for representing advertisement descriptions in concept expression. In the **Advertisement** ontology, we define a **Advertisement** class to express a general advertisement. Like the general requirements we mentioned in above section, a general advertisement consists of many trip element, such like Spot advertisement. Therefore, we can decompose a compound advertisement into many small pieces of trip as we did to a compound requirement. According to the architecture described in Section 4.4, we have implemented the **Advertisement** ontology in our system, as shown in Figure 5.10. The root class, **Advertisement**, represents a set of compound advertisements. Similar to the requirements ontology, it also has two subclasses. The **GeneralAdvertisement** class is used to represent composed advertisements. The **TEAdvertisement** class has four types of trip element advertisements as subclasses. They are **AccomAdvertisement**, **EventAdvertisement**, **SpotAdvertisement**, and **TransAdvertisement**. Accordingly, a **GeneralAdvertisement** class has four types of properties called **hasAccomTE**, **hasEventTE**, **hasSpotTE**, and **hasTransTE**. The respective object property classes **AccomAdvertisement**, **EventAdvertisement**, **SpotAdvertisement**, and **TransAd-**

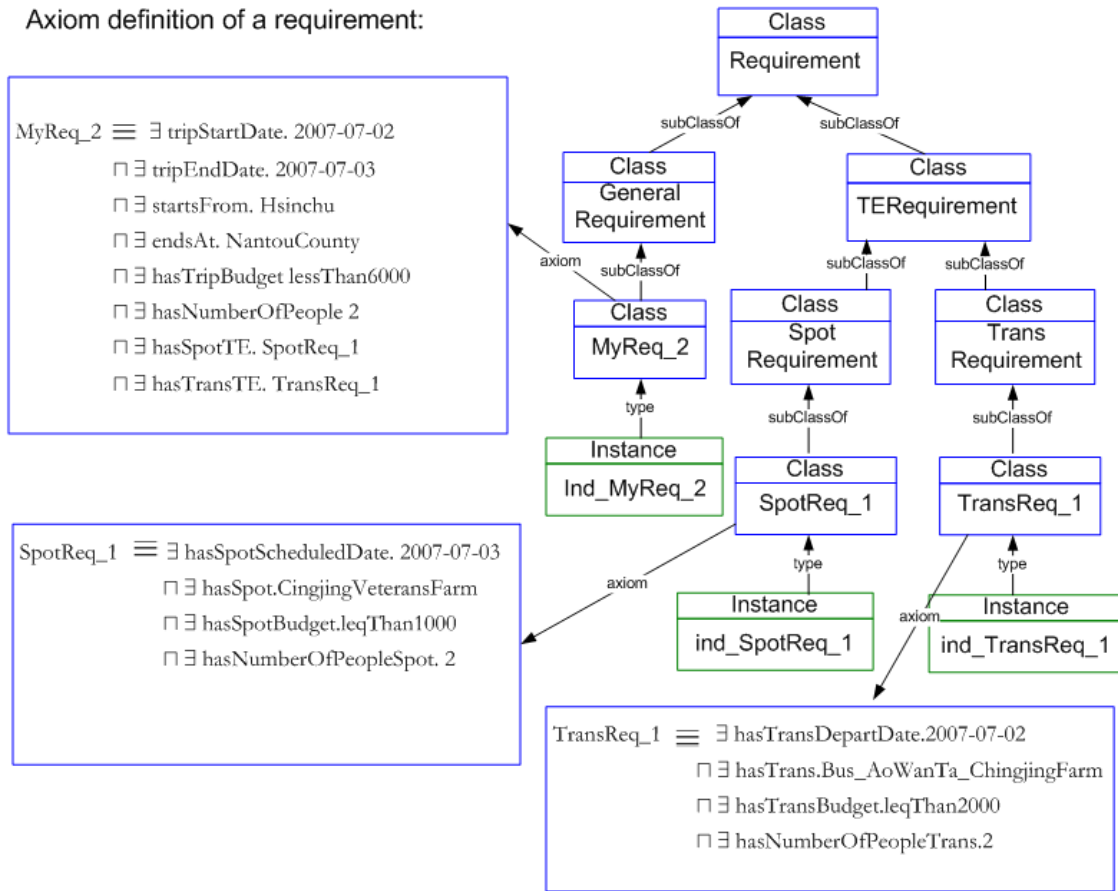


Figure 5.8: The **Requirement** Concept Definition in the Traveller

vertisement represent the different types of trip element advertisements in the tourism domain.

A trip advertisement also contains necessary information, such as the trip start date, end date, departure location, arrival location, trip budget, and so on. Those trip information we mentioned above are annotated as properties, such as **tripStartDate**, **tripEndDate**, **startsFrom**, **endsAt**, **hasTripBudget**, of the **Requirement** class.

The **Package** class is used to describe a trip package. The **hasTripElement** property connects the **Package** and the **Advertisement**. Through the **hasTripElement** property, these **Advertisement** can be seen as several stops included in the trip **package**. In the Traveller, we use **GeneralAdvertisement** as a trip's element advertisement

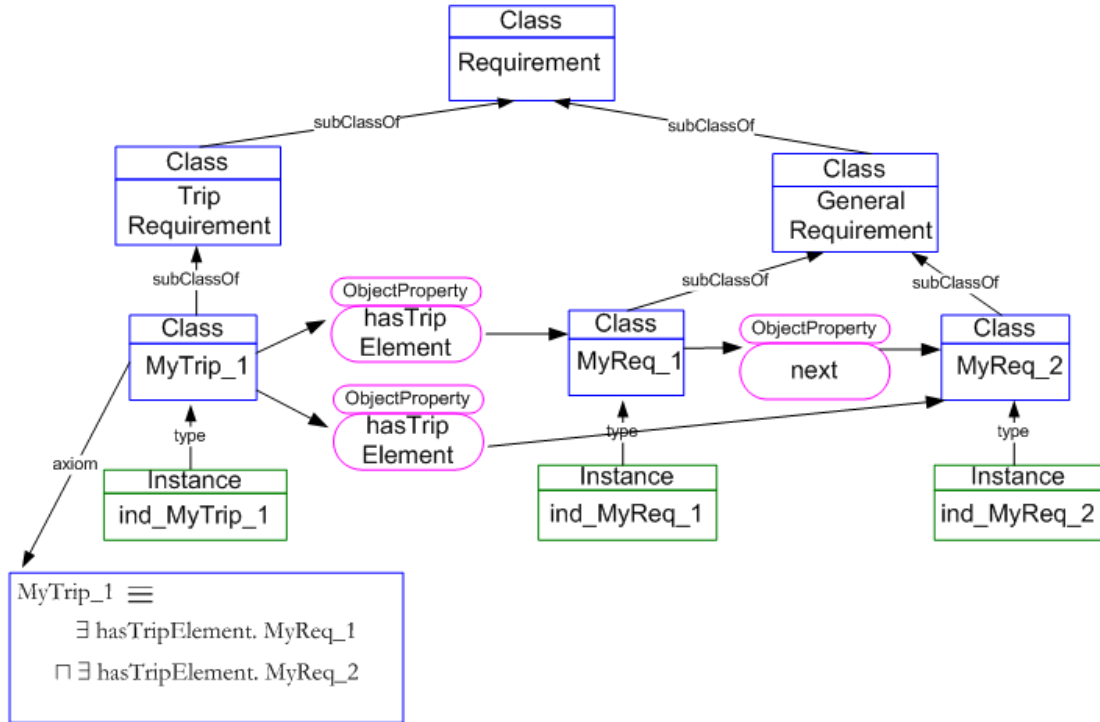


Figure 5.9: The Trip Requirement of the scenario in the Traveller

that can be divided by the date of the itinerary.

Figure 5.11 is example of an advertisement in the Traveller. According to the providers' Web Services profile defined in 5.2.2, we create two classes, called **TripAd_1** and **TripAd_2**, as advertisements that respectively represent the first day and the second day trip advertisement. The **next** property represents the dependencies of the two advertisements. **TripAd_2** has two trip element advertisements, **SpotAd_1** and **TransAd_1**, stand for the spot advertisement and the transportation advertisement. These advertisements are connected to **TripAd_2** by **hasSpotTE** and **hasTransTE** properties. Figure 5.12 shows the concept definitions of **TripAd_2** and its trip element advertisements, **SpotAd_1** and **TransAd_2** classes. Figure 5.13 illustrates the implementation of the scenario in the previous sections.

Advertisement in the Traveller - Class and property definitions:

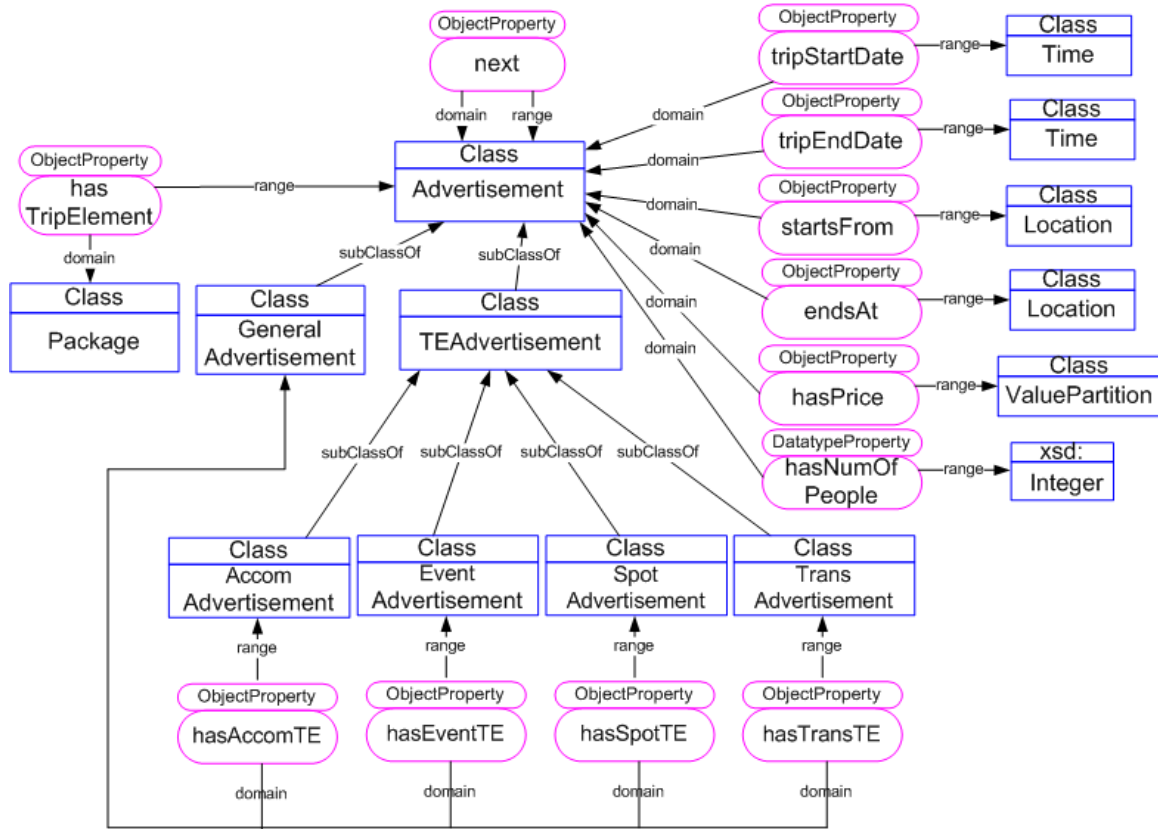


Figure 5.10: The **Advertisement** Ontology in the Traveller

5.5 Constraint Checking

In Section 4.5, we discussed the approach for handling different kinds of constraints. Those constraints are used to check the consistency of the ontology. In the Traveller, we simplify the constraint problem to three main topics: *Time constraints*, *Budget constraints*, and *Location constraints*. When we handle the binary relationships of constraints, we construct a concept hierarchy that implies relationships of subsumption reasoning, so that the binary relationships of the constraints can be checked easily by concept subsumption reasoning. Even if we want to handle the binary relationship of two quantitative constraints, we model the concept hierarchy of the quantitative constraints like the ValuePartition ontology 4.5.3. Through design of the hierarchy, ValuePartition ontology transforms a problem of comparisons of value to a problem of concept subsumption. By

An Advertisement example in the Traveller:

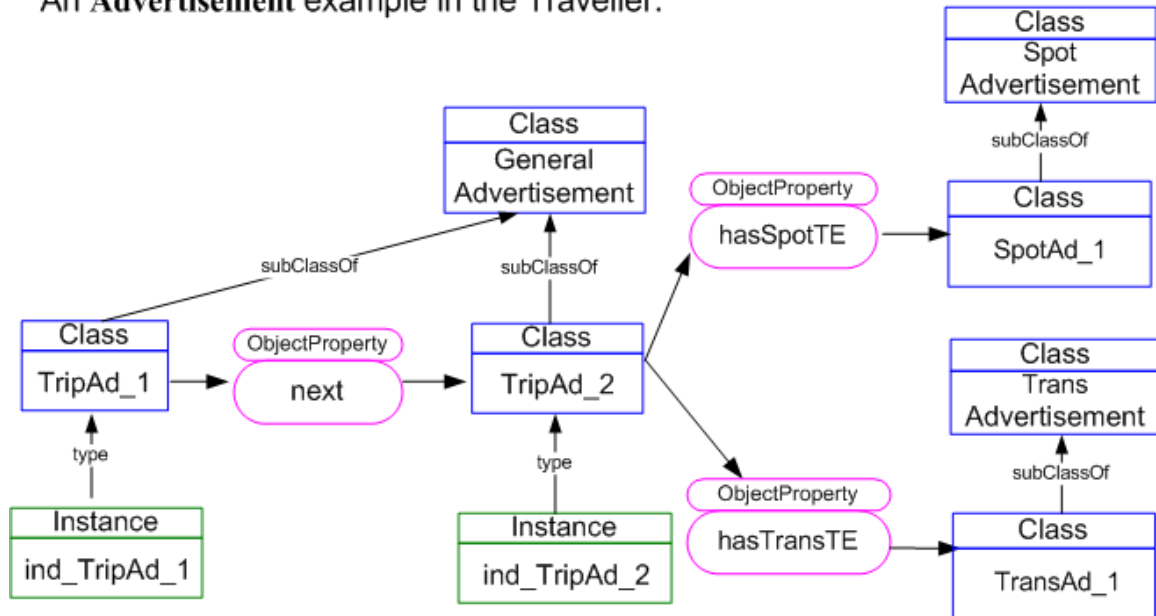


Figure 5.11: The **Advertisement** Example in the Traveller

concept subsumption reasoning, we not only can compare a requirement with an advertisement for service matching, also we can compare the quantitative relationship between constraints of two element requirements in a composed requirement.

In Figure 5.14 shows a new picture for constraint handling in the Traveller, we adopt the Semantic Web Rule Language (SWRL) and JESS rule engine as the solutions in the non-quantitative constraints of the global relationship. The SWRL rule specify the relationships of the constraints and check the consistency of the knowledge base. It is used to infer new knowledge that implies role compositions among complicated relationships of non-quantitative constraints. We also adopt the Protégé Axiom Language(PAL), a tab-widget plug-in for Protégé, to define PAL rules that help us check the relationships among quantitative constraints. For administrators, SWRL rule and the Protégé Axiom Language provides a useful back-end tools that make management ontology more efficient. In Section 5.6, we focus on the implementation of PAL rules and SWRL rules.

In the following sections, we explain the Time constraints and Budget constraints in the Traveller. We also discuss the binary relationships and global relationships between these constraints.

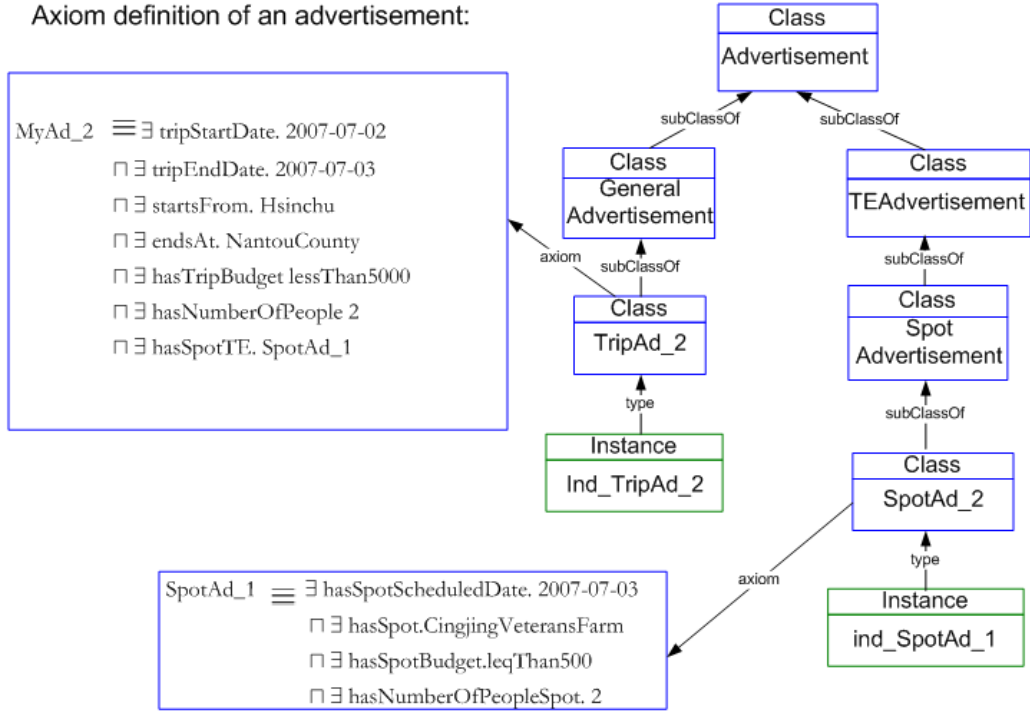


Figure 5.12: The **Advertisement** Concept Definition in the Traveller

5.5.1 Time Constraints

In the descriptions of the requirements and advertisements, Time constraint specify the time condition of the trip activity, such as start date and end date of the trip, which are represented by the **tripStartDate** and the **tripEndDate** properties. According to the approach [39], we facilitate *before* and *after* relationships between the **tripStartDate** and the **tripEndDate** properties. Through [39], we can check the time dependencies to decide the order of two dates or if a date within a date range. In the scenario described in the previous section, the customer want to start a trip from July 2nd to July 3rd, 2007. The Traveller dynamically add Y2007M07D02 and Y2007M07D02 to the Time Ontology. The time definition of Y2007M07D02 concept is presented as follows:

$$\begin{aligned} \text{Y2007M07D02} \sqsubseteq & \exists \text{ends_before.Y2007M07D03} \\ & \sqcap \exists \text{begins_after.Y2007M07D01} \\ & \sqcap \exists \text{begins_after.Y2007M06} \end{aligned}$$

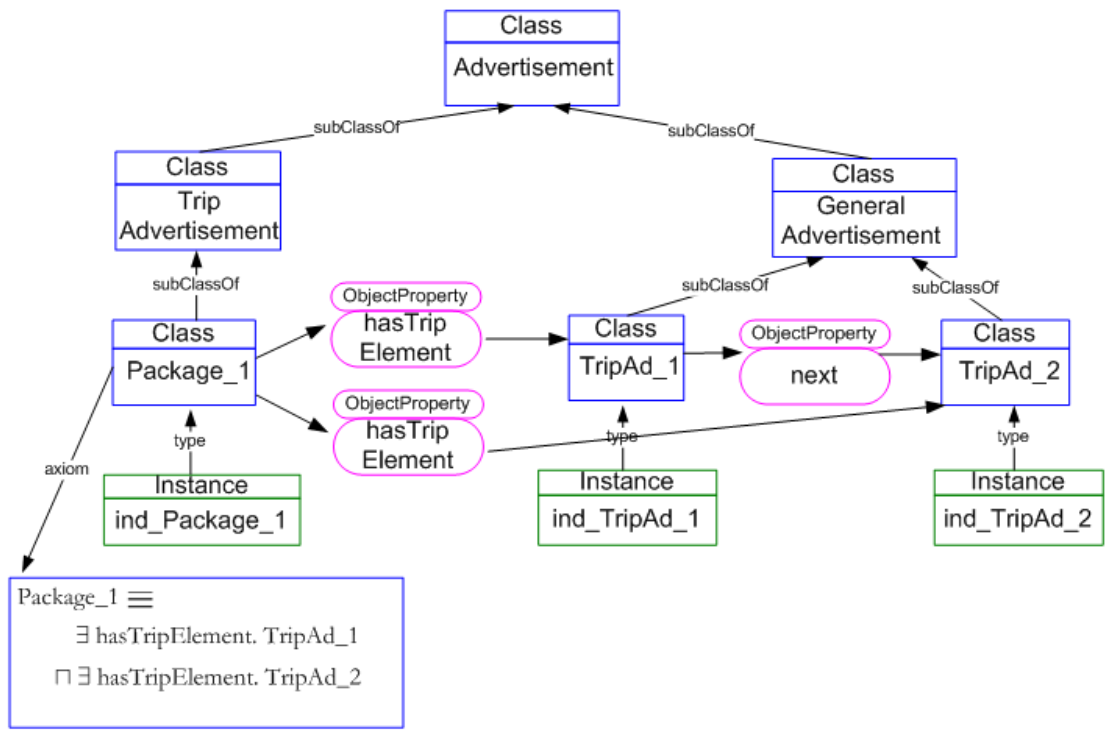


Figure 5.13: The Trip Package Example

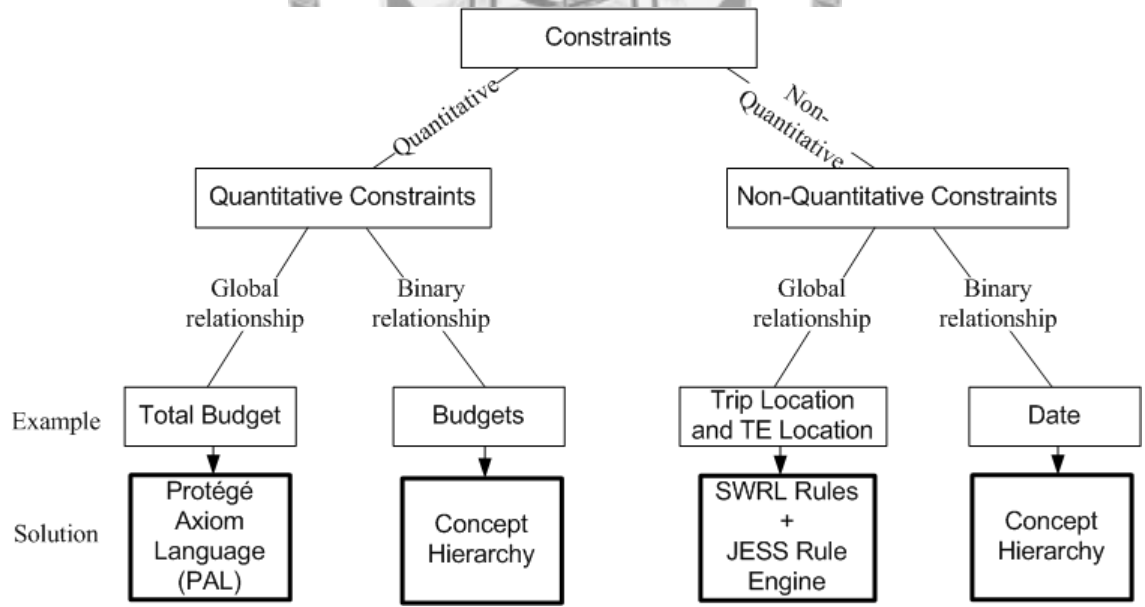


Figure 5.14: Constraint Handling and Checking in the Implementation System

□ Y2007M07

Subsequently, we intuitively use Y2007M07D01 concept as the restriction of the **tripStartDate** property in Requirement concept. Similarly, Y2007M07D03 concept will be the restriction of the **tripEndDate** property in Requirement concept. Finally, the system automatically infers relationships about the Y2007M07D2 and Y2007M07D3 concept. Therefore, the relationship between the tripStartDate and tripEndDate can be easily examined, such as checking whether the tripStartDate is before the tripEndDate by checking concept subsumption listed below:

$$\text{TripStartDateConcept} \sqsubseteq \exists \text{ ends_before.TripEndDateConcept}$$

In the scenario described in the previous section, the customer plans to go Ching Jing Farm on July 3rd, 2007. The spot element requirement has a **hasSpotScheduledDate** property and a concept of Time ontology as its restriction. The spot scheduled date is a concept of Y2007M07D03 presented in the Time ontology. We can check whether the spot scheduled date, Y2007M07D03, is within the range of the start and the end dates of the trip by checking the subsumption listed below:

$$\text{SpotScheduledDateConcept} \sqsubseteq \exists \text{ begins_after.tripStartDate} \sqcap \exists \text{ ends_before.tripEndDate}$$

The examples we mention above belongs to non-quantitative binary relationship defined at section 4.5.1. we can handle the constraints of tripStartDate, tripEndDate, and hasSpotScheduledDate by the existing roles, *ends_before* and *begins_after*, of the Time concept; instead of using SWRL rules.

5.5.2 Budget Constraints

The budget constraints usually restrict the number of budget greater or less than a specific number. In practical, the budget constraints check if each *Budget* of element requirement is within the *Total Budget* of the trip requirement. They also check that if the summation of budgets of element requirements is less than total budget of the trip requirement. According to the **ValuePartition** ontology proposed in [39], this kind of the quantitative constraint can be transformed to a concept subsumption problem as Time

constraint. That means the system can solve quantitative constraint by checking concept subsumption. In OWL-DL, it is hard to make quantitative handling, such as budget constraint. **ValuePartition** complements the quantitative weakness in OWL-DL.

With dynamic concept program, the system automatically create the subsumption relationships between the quantitative concept in the **ValuePartition** ontology as figure 5.15.

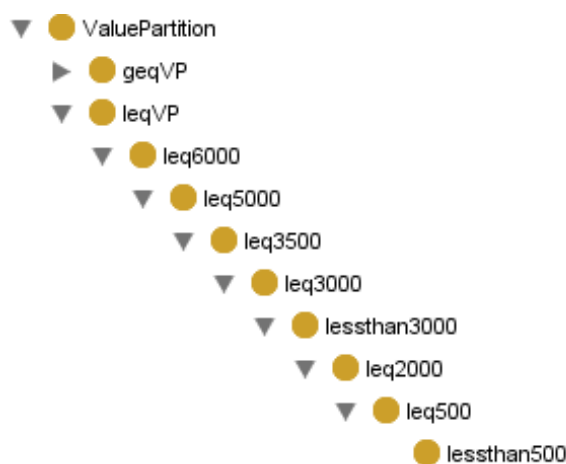


Figure 5.15: Relationship between leq5000 and leq500 in the ValuePartition Ontology

Taking the scenario as the example, we facilitate subsumption reasoning to check if the budget of the *spot advertisement* is less than the advertisement budget in our scenario. Trip advertisement 5.2.2, AdvertisementNantou-0, has a property **hasBudget** which connects with concepts, leq5000, which describes the cost of advertisement is less than Nt.5000. . Similarly, Spot advertisement, SpotAd-0, has a property **hasSpot-Budget**, and its budget is Nt.500. Before adding the budget of advertisement and spot advertisement, the system automatically create **leq5000** and **leq500** concepts with subsumption relationships in advance.. When adding the **leq500** concept that expresses the spot budget, the implied relationship is also defined. Figure 5.15 shows the relationship between **leq5000** and **leq500**, where **leq500** \sqsubseteq **leq5000** can be reasoned through the Racer DL reasoner. The problem of checking the budget 500 is less than the budget 5000 can be solved by checking subsumption relationship as follows:

$$\text{LessThanTripElementBudgetConcept} \sqsubseteq \text{LessThanTripBudgetConcept}$$

Not only comparing **hasBudget** and **hasSpotBudget**, we adopt the same approach to the relationship checking between **hasBudget** and **hasEventBudget**, **hasAccommodationBudget**, or **hasTransBudget** properties which describe the budgets for events, accommodation, and transportation, respectively.

However, the complicated comparison problem involving more than two budgets, i.e., problem of checking whether the summation of the budgets of the element requirements is less than the total budget of the trip requirement or not. For this kind of problem, concept subsumption relationship between ValuePartition concepts does not work. In Section 5.6.1, we will illustrate how to use PAL rules to solve the problem of more than two budgets.

5.6 Constraint Rules

In 4.5 and 5.5, we realize that the subsumption induction can not solve the global relationship constraint. Accordingly, we adopt additional rules language, PAL and SWRL rules, to complement the lack of OWL-DL expression power as shown in the figure 5.14. In the following sections, we explain the mechanism of these rules and give examples of implementing our system.

5.6.1 PAL Rules

The Protégé Axiom Language tab-widget plug-in(PAL) is a useful tool for Protégé that check integrity constraints in ontologies. We adopt the PAL as the engine to validate the global quantitative constraint. While constructing the rules, we have to define the range and write the constraint statements. Here, we take the Total Budget Constraint as the example. First, we define the rule for ensuring the summation of all trip element budgets must be equal to or less than the trip budget. However, PAL does not support recursive rule definitions. Therefore, we need to manually define many rules with different numbers of trip elements explicitly. For example, we have a PAL statement that checks the summation of the budget of trip element(1) and trip element(2) for two-element trips. But a trip may contain three or more element, that's why we have to define many

statement for those trips which contain at least three elements. We represent the PAL rule for checking 2-element trips as follows:

- The range definitions:

```
(defrange ?trip :FRAME Trip)
(defrange ?tripElement :FRAME GeneralRequirement hasTripElement)
```

The two statements define the ranges of two variables used in the constraint statements. The first `?trip` stands for instances of the **Trip** class (frame). `:FRAME` means a class in OWL. The second `?tripElement` stands for instances of the **GeneralRequirement** class (frame) and appears in the range of a `hasTripElement` property (slot).

- The constraint statement:

```
(forall ?trip (forall ?tripElement
  (=>(and (hasTripElement ?trip ?tripElement)
    (= (number-of-slot-values hasTripElement ?trip) 2)
    (own-slot-not-null next ?tripElement))
  (or (> (hasBudget ?trip)
    (+ (hasBudget ?tripElement)
      (hasBudget (next ?tripElement))))
    (= (hasBudgetD ?trip)
      (+ (hasBudget ?tripElement)
        (hasBudget (next ?tripElement)))))))
```

5.6.2 SWRL Rules

As PAL rules, Semantic Web Rule Language (SWRL) rules enhances the limited expressive power of OWL-DL, SWRL rules are defined to capture the complicated relationships of role/property compositions over ontologies. We adopt the Protégé SWRLJess Tabwidget plug-in as a rule editor for defining rules. Accordingly, we adopt SWRL rules and a rule engine to handle non-quantitative constraints as the Figure shown in 5.14 Here, we

demonstrate a SWRL rule example applied in the Traveller. If we want to infer the implicit relationship, **next** property, between two trip elements, we check the condition: if the first trip's arrival location is the location of another trip's departure location. If it matched with the condition, then system inference that they are connected by the **next** property used to describe their order. The following rule expresses the **next** property of concepts that is inferred by asserting trip individuals.

$$\text{hasTripElement}(\text{?a}, \text{?x}) \wedge \text{hasTripElement}(\text{?a}, \text{?z}) \wedge \\ \text{endsAt}(\text{?x}, \text{?y}) \wedge \text{startsFrom}(\text{?z}, \text{?y}) \longrightarrow \text{next}(\text{?x}, \text{?z})$$

The SWRL rules are facilitated by individuals in ABOX modelling. The variables with initial ? indicate individuals. Because SWRL rules operating at ABOX, we respectively build an trip element individual for representing the trip element concept. Reference 5.4.4 for detailed information.

The second example of SWRL rule is used to induce if a location of spot is accessible for transportation line. In the ontology, we have properties: **locatedIn** describes a spot location, **isPassedBy** describes the location is accessed by what kind of transportation, such as bus, train, or air plane. With these two properties, we can inference an implicit property, **isReachable**, indicate the spot is accessed by what kind of transportation. The rule is illustrated by the following SWRL rule:

$$\text{locatedIn}(\text{?spot}, \text{?location}) \wedge \text{isPassedBy}(\text{?location}, \text{?transportation}) \longrightarrow \\ \text{isReachable}(\text{?spot}, \text{?transportation})$$

The above SWRL rule is suitable for ?spot and also for ?accommodation. It represents that whenever any transportation passes through the location where the accommodation locates, the accommodation can be reached by that transportation.

Though the above-mentioned examples of SWRL rules look trivial, they are still efficient in ontology maintenance. In the Traveller, we adopt TBOX modeling to represent the descriptions of requirements and advertisements because of the mechanism of subsumption reasoning. However, SWRL rules can only apply to individuals in ABOX. The Ontologies in the Traveller has to be inferred by SWRL rules periodically to discovery

the complicated relationships between properties.

5.7 The Traveller Demonstration

As we mentioned in the previous chapter 5.3, we have implemented a new prototype system, the Traveller, as a Web application based on our ontology-based architecture and related methodologies. The Traveller can be considered as a Web application providing integrated tourism planning service on the Internet. From requirements defining, service matching, service execution, to ontology maintenance, the whole procedure can be manipulated remotely at the user's browsers. In this chapter we will illustrate the system integrated user interface and demonstrate the tourism matching service provided by the Traveller step by step.

5.7.1 Matching Service Process

The graphical user interface of the Traveller system is shown in Figure 5.16. The user interface consists of three main panels - requirement input panel, matching result panel on the left side of the interface. And AJAX-based Google Maps on the right side. The matching process comprised of several steps listed in the following:

1. To input trip requirement, according to the customer preferences, they are able to choose to operate the AJAX-based Google Maps on the right side of the Traveller web page or the traditional combo-box input style on the left side. Through mouse clicking on the map, the customer decides the starting location and the destination of the daily trip requirement. Furthermore, the customer could infill the additional condition such as start date, end date, budget, transportation, accommodation, number of people, and tourism spot. Customers could compose many daily trip requirement as their complete trip requirement.
2. After finish inputting trip requirements, click the "Send" button on the top-left corner of requirement input panel to perform our matching scheme. In the process of matching, approximate requirements with similarity value will be generated and stored in the ontology. Then, Racer Inference Engine will be triggered to

perform the "Classification". Classification help us find subsumption relations between approximate requirements and advertisements. A advertisement subsumed by approximate requirements represents that it is a suiting service for the customer. Detailed information of matching scheme will be introduced in [28].

3. According to similarity value, all matching results will be listed in the result combo-box on the bottom of panel. To get detailed information about suiting trip package, selecting the package listed in combo-box. Then the route of the package you select will be displayed graphically on the map, also related information will be shown in the text area below.
4. After reviewing the packages, the customer can decide to execute which package through checking the check-box corresponding that package. Once "Execute" button is clicked, the Traveller will perform the service execution procedure, also related information about the selected package will be published to Wiki-based community as the customer's trip history log.



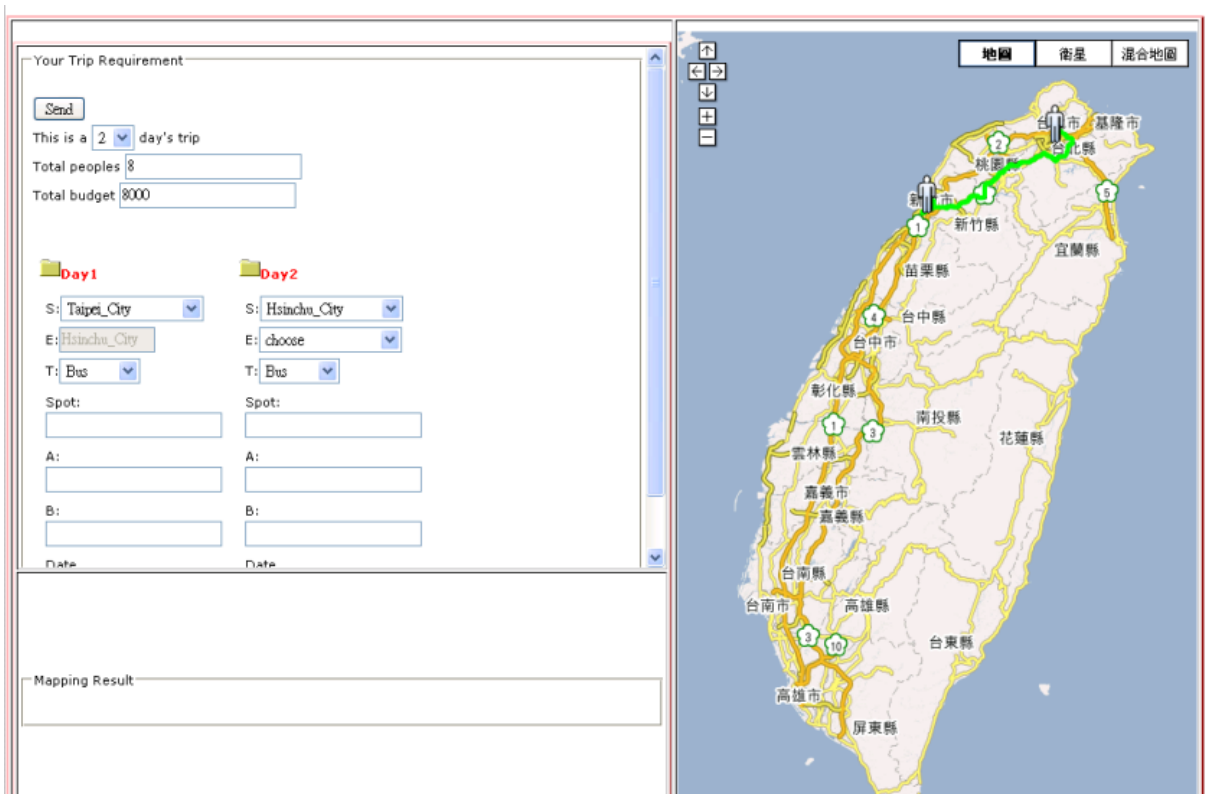


Figure 5.16: The User Interface of Matchmaker

Chapter 6

Conclusion

Although Semantic Web technology enables Web Services to be matched precisely according to their semantic descriptions, it also impose high entrance barriers of Semantic Web applications. However, Web 2.0 technology booming recently has demonstrated a lot of remarkable feathers that make the user closer to Web application. With borrowing brilliant ideas from Web 2.0, we create a new architecture combining the strength of Semantic Web and Web 2.0 technology. That makes Semantic-based application more friendly and easier for maintenance.

In this thesis, we proposed a new semantic-based service composition architecture in the distributed environment. Exception for combining Web 2.0 and Semantic Web technologies together, we also create a community platform in our architecture that allows user to share their information and to participate in ontology maintenance cooperatively. Through ontology maintenance, the system is able to utilize the descriptive capability of OWL adequately for matching service. Web Services which fit the customer's needs can be discovered and executed automatically. In this chapter, we summarize contributions of this thesis and the future works.

6.1 Contributions

- **A new semantic-based service composition architecture** A complete architecture providing service matchmaking and service invocation was proposed. Different from the previous version, we enhance the interaction between the user and the system. Also the user has more power to participate in the ontology maintenance through Community Component. By integrating advantage of Semantic Web and

Web 2.0 technology, a Semantic application based on this architecture would be even more friendly and efficient.

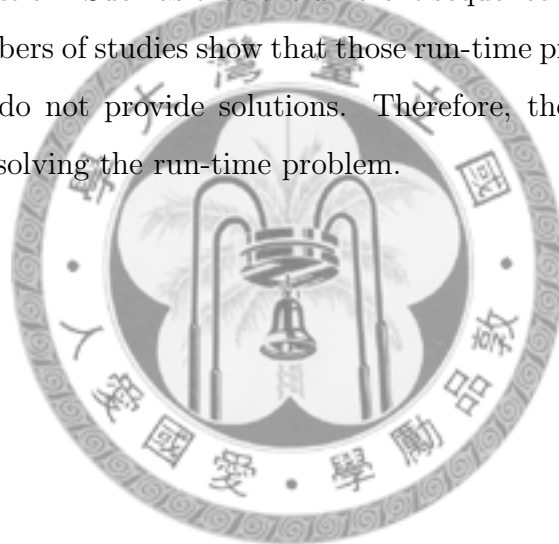
- **differentiation between the customer and the administrator** The customer and the administrator's view of this system are divided. All complex Semantic language are encapsulated by the system in the front-end, the customer manipulates the system without having any Description Logics background. In contrast, the administrator has authority to log on Knowledge Base Management System dedicated on maintaining ontologies in the back-end.
- **Ontology Maintenance Procedure** We proposed an Ontology Maintenance Procedure that involves the customer in ontology engineer through Wiki-based community. Time-consuming Ontology Maintenance is no longer controlled by a small group. Because of the customer's participation, Ontology Maintenance become even more agile and quick
- **The Traveller Prototype System** We implement a new Semantic application system called Traveller based on our ontology-based architecture and related methodologies. From requirements defining, service matching, service execution, to ontology maintenance, the whole procedure can be manipulated remotely at the user's browsers. The Traveller can be considered as a Web application providing integrated tourism service on the Internet.

6.2 Future Work

- **Integration between Semantic Web and Web 2.0 technology** Although we have proposed and implemented a architecture combing Semantic Web and Web 2.0 technology, but there are a lot of experiments needed to be validated for the system efficiency.
- **Ontology design methodologies** We do have our approach to model ontology in the travel domain, but we are not sure whether the design methodologies is good enough. Our ontology design methodologies are still waiting for being testify by experts in related domain. Also we need to concern about the trade-off between

expressiveness and realization. It is a depends-on question according to the application's domain and the computing ability of inferencing. To improve practicability of the Semantic application, these issues should be estimated carefully.

- **System efficiency** The system efficiency is a critical factor to decide if a system is usable. it includes the service reasoning time, the service execution time, loading time, and so on. As things stand, our service reasoning time is a little bit slow for the user. More Semantic application issues that increase system efficiency should be discussed. Such as Ontology segmentations.
- **Problem of Service Execution** There are many run-time situations and problems in terms of Service execution in real-world. Complicated factors are getting involved with service execution. Such as that the different sequence may lead to the different come out. A numbers of studies show that those run-time problems are very difficult challenges, they do not provide solutions. Therefore, there is a lot of space for improvement on solving the run-time problem.



Bibliography

- [1] *Time Ontology in OWL*, <http://www.w3.org/TR/owl-time/>.
- [2] *Universal Description, Discovery, Integration*, <http://www.uddi.org/>.
- [3] *Web Services Description Language*, <http://www.w3.org/TR/wsdl>.
- [4] E-services: Current technology and open issues. volume 2193/2001. Springer Berlin / Heidelberg, Sep. 2001.
- [5] Rohit Aggarwal, Kunal Verma, John Miller, and William Milnor. Constraint driven web service composition in meteor-s. *Services Computing, 2004.(SCC 2004). Proceedings. 2004 IEEE International Conference on*, pages 23–30, 2004.
- [6] Danilo Ardagna and Barbara Pernici. Global and local QoS constraints guarantee in Web service selection. *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, page 806, 2005.
- [7] Assaf Arkin, Sid Askary, Ben Bloch, Francisco Curbera, Yaron Goland, Neelakantan Kartha, Canyang Kevin Liu, Satish Thatte, Prasad Yendluri, and Alex Yiu. Web services business process execution language version 2.0. WSBPEL-specification-draft-01. *OASIS (2005)*.
- [8] Assaf Arkin, Sid Askary, et al. WS-BPEL: Web services business process execution language version 2.0, 2004.
- [9] Franz Baader, Diego Calvanese, McGuinness Deborah, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

- [10] Tim Berners-Lee. Services and Semantics Web architecture. *white paper, World Wide Web Consortium*, 2001.
- [11] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik F. Nielsen, Satish Thatte, and Dave Winer. Simple Object Access Protocol (SOAP) 1.2. Technical report, www.w3c.org, 2003.
- [12] François Bry, Frank-André Ries, and Stephanie Spranger. CaTTS: calendar types and constraints for Web applications. *Proceedings of the 14th international conference on World Wide Web*, pages 702–711, 2005.
- [13] Anis Charfi and Mira Mezini. Aspect-oriented web service composition with ao4bpel. *Proceeding ECOWS*, 2004.
- [14] Yi-Shan Cheng. An approach to mapping relational databases to ontologies. 2007.
- [15] Roberto Chinnici, Martin Gudgin, Jean J. Moreau, and Sanjiva Weerawarana. Web Services Description Language (WSDL) Version 1.2. Technical report, www.w3c.org, 2002.
- [16] The DAML Services Coalition. DAML-S: Semantic Markup for Web Services. Technical report, www.daml.org, 2002.
- [17] Luiz A. G da Costa, Paulo F. Pires, and Marta Mattoso. Automatic composition of web services with contingency plans. In *IEEE International Conference on Web Services (ICWS'04)*, 2004.
- [18] Mike Dean, Dan Connolly, Frank V. Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn A. Stein. Web Ontology Language (OWL) reference version 1.0. Technical report, www.w3c.org, 2002.
- [19] T. Di Noia, E. Di Sciascio, and F.M. Donini. Extending semantic-based matchmaking via concept abduction and contraction. *EKAW 2004*, pages 307–320, 2004.
- [20] T. Di Noia, T. Di Sciascio, F.M. Donini, and M. Mongiello. Semantic matchmaking in a P-2-P electronic marketplace. In *In Proceedings of the Eighteenth Annual*

ACM (SIGAPP) Symposium on Applied Computing, Special Track on E-commerce technologies, pages 532–536, March 2003.

- [21] Glen Dobson. Using WS-BPEL to implement software fault tolerance for web services. *Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 126–133, 2006.
- [22] A. Dogac, Y. Kabak, G. Laleci, S. Sinir, A. Yildiz, and A. Tumer. Satine project: Exploiting web services in the travel industry. In *eChallenges 2004 (e-2004)*, 2004.
- [23] Thomas R. Gruber. A translation approach to portable ontology specifications. Technical report, Knowledge Systems Laboratory, Computer Science Department, Stanford University, 1993.
- [24] Jerry R. Hobbs and Feng Pan. An ontology of time for the Semantic Web. In *ACM Transactions on Asian Language Information Processing*, volume 3, pages 66–85, 2004.
- [25] Ian Horrocks, Frank V. Harmelen, Peter Patel-Schneider, Tim Berners-Lee, Dan Brickley, Dan Connolly, Mike Dean, Stefan Decker, Dieter Fensel, Richard Fikes, Pat Hayes, Jeff Heflin, Jim Hendler, Ora Lassila, Deb McGuinness, and Lynn A. Stein. DAML+OIL. Technical report, www.daml.org, 2001.
- [26] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosf, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Technical report, www.daml.org, 2004.
- [27] Chia-Tzu Hsieh. The Traveller: A Service Combination System Based on Semantic Web Technology. Master’s thesis, 2006.
- [28] Chung-Hao Hsieh. Approximate Matching and Ranking of Web Services Using Ontologies and Rules. Master’s thesis, 2007.
- [29] Chen-Feng Huang. A Semantic-Based Framework for Web Services Composition, Master’s thesis, 2005.
- [30] IBM. Business Process Execution Language for Web Services, 2002.

- [31] D. Karastoyanova. A Methodology for Development of Web Service-based Business Processes. *Proceedings of AWESOS*, 2004.
- [32] Markus Keidl, Stefan Seltzsam, and Alfons Kemper. Reliable web service execution and deployment in dynamic environments. *Proc. of the Intl. Workshop on Technologies for E-Services (TES)*, 2819:104–118.
- [33] Rania Khalaf, Nirmal Mukhi, and Sanjiva Weerawarana. Service-oriented composition in BPEL4WS. In *Proceedings of the Twelfth International Conference on World Wide Web (WWW)*, page 768, 2003.
- [34] R. Lara, D. Roman, A. Polleres, and D. Fensel. A conceptual comparison of WSMO and OWL-S. *Proceedings of the European Conference on Web Services (ECOWS 2004)*, 2004.
- [35] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Technical report, www.w3c.org, 1999.
- [36] Frank Leymann, Dieter Roller, and Satish Thatte. Goals of the BPEL4WS Specification. *xml.coverpages.org (2003)*, August. <http://xml.coverpages.org/BPEL4WS-DesignGoals.pdf>.
- [37] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. *International Journal of Electronic Commerce*, 8(4):39–60, 2004.
- [38] Jun-Hong Liu. Mapping Relational Databases to Ontologies: An Approach Using Cluster Analysis, Master’s thesis, 2008.
- [39] Wei-Lun Lu. Approximate Matching of Service Descriptions Using Ontologies and Rules, Master’s thesis, 2006.
- [40] Daniel J. Mandell and Sheila A. McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation.
- [41] Daniel Bachlechner Martin Hepp and Katharina Siorpaes. Harvesting wiki consensus - using wikipedia entries as ontology elements. In *Proceeding of the First Workshop on Semantic Wikis - From Wiki to Semantic[SemWiki2006]*, 2006.

- [42] D.L. McGuinness, F. van Harmelen, et al. OWL Web Ontology Language Overview. *W3C Recommendation*, 10:2004-03, 2004.
- [43] Brahim Medjahed, Athman Bouguettaya, and Ahmed K. Elmagarmid. Composing Web Services on the Semantic Web. *The VLDB Journal The International Journal on Very Large Data Bases*, 12(4):333–351, 2003.
- [44] Alexander Mikroyannidis. Toward a social Semantic Web. *Computer*, November 2007, 2007.
- [45] T.D. Noia, E.D. Sciascio, F.M. Donini, and M. Mongiello. A system for principled matchmaking in an electronic marketplace. *International Journal of Electronic Commerce*, 8(4):9–37, 2004.
- [46] Tim O’Reilly. What is Web 2.0: Design patterns and business models for the next generation of software. In <http://www.oreillynet.com/>, 2005.
- [47] J.Z. Pan and I. Horrocks. OWL-E: Extending OWL with expressive datatype expressions. Technical report, IMG Technical Report, Victoria University of Manchester, 2004.
- [48] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Semantic Matching of Web Services Capabilities. In *Proceedings of the First International Semantic Web Conference (ISWC)*, volume 2342 of *Lecture Notes in Computer Science*, pages 333–347. Springer-Verlag, 2002.
- [49] Marco Pistore, Paolo Traverso, Piergiorgio Bertoli, and Annapaola Marconi. Automated synthesis of composite BPEL4WS web services. *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, pages 293–301, 2005.
- [50] A. Rector. Representing Specified Values in OWL: value partitions and value sets. *W3C Working Group Note*, 17, 2005.
- [51] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rube’n Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Christoph Bussler, and Dieter Fensel. Web Service Modeling Ontology (WSMO). *Applied Ontology 1(2005)*, 2005.

- [52] Andreas Schmidt. Knowledge maturing and the continuity of context as a unifying concept for knowledge management and e-learning. 2005.
- [53] Andreas Schmidt Simone Braun and Andreas Walter. Ontology maturing: a collaborative web 2.0 approach to ontology engineering. In *Proceedings of the Workshop on Collaborative Construction of Structured Knowledge at the 16th International World Wide Web Conference, 2007*.
- [54] Biplav Srivastava and Jana Koehler. Web service composition: Current solutions and open problems. In *Workshop on Planning for Web Services(ICAPS)*, pages 28–35, 2003.
- [55] N. Stojanovic, R. Studer, and L. Stojanovic. An approach for the ranking of query results in the Semantic Web. In *Proceedings of the Second International Semantic Web Conference (ISWC)*, volume 2870 of *Lecture Notes in Computer Science*, pages 500–516. Springer-Verlag, 2003.
- [56] Katia Sycara, Seth Widoff, Matthias Klusch, and Jianguo Lu. LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5(2):173–203, 2002.
- [57] Hsin-Ying Tai. Automated Web Service Composition and Execution Based on Semantic Web Technology. Master’s thesis, National Taiwan University, July 2007.
- [58] Paolo Traverso and Macro Pistore. Automated composition of semantic web services into executable processes. *Proceeding ISWC 04*, 2004.
- [59] Paolo Traverso and Macro Pistore. Automated composition of Semantic Web services into executable processes. 2004.
- [60] Yih-Kuen Tsay, Po-Chun Chen, Chih-Hsiung Liu, and Jyun-Yang Syu. Ontology-Based Automation of Web Services Composition and Brokering. Unpublished Manuscript, 2004.
- [61] Chih-Hua Tu. A Semi-Automatic Approach for Mapping Structured Web Pages to Ontologies, Master’s thesis, 2008.

- [62] Kunal Verma, Karthik Gomadam, Amit P. Sheth, John A. Miller, and Zixin Wu. The METEOR-S Approach for Configuring and Executing Dynamic Web Processes. *LSDIS METEOR-S project. Date*, pages 6–24.
- [63] Jian Yang and Mike. P. Papazoglou. Web components: A substrate for web service reuse and composition. In *Proceedings of the Fourteenth International Conference on Advanced Information Systems Engineering (CAiSE'02)*.

