# SEMANTIC PROCESSES FOR CONSTRUCTING COMPOSITE WEB SERVICES

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

BY

KARANİ KARDAŞ

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

AUGUST 2007

Approval of the thesis:

**SEMANTIC PROCESSES FOR CONSTRUCTING COMPOSITE WEB SERVICES**

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

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

Prof. Dr. Volkan Atalay
Head of Department, **Computer Engineering** ____________________

Asst. Prof. Dr. Pınar Şenkul
Supervisor, **Computer Engineering Dept., METU** ____________________

**Examining Committee Members:**

Prof. Dr. İsmail Hakkı Toroslu
Computer Engineering Dept., METU ____________________

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

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

Asst. Prof. Dr. Hürevren Kılıç
Computer Engineering Dept., Atılım Uni. ____________________

M.Sc. Arslan Arslan
Manager, LOGO ____________________

**Date:** 27.08.2007

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

Name, Last name: Karani Kardaş

Signature :

## ABSTRACT

SEMANTIC PROCESSES FOR CONSTRUCTING COMPOSITE WEB SERVICES

Kardaş, Karani
M.Sc., Department of Computer Engineering
Supervisor : Asst. Prof. Dr. Pinar Senkul

August 2007, 106 pages

In Web service composition, service discovery and combining suitable services through determination of interoperability among different services are important operations. Utilizing semantics improves the quality and facilitates automation of these operations. There are several previous approaches for semantic service discovery and service matching. In this work, we exploit and extend these semantic approaches in order to make Web service composition process more facilitated, less error prone and more automated. This work includes a service discovery and service interoperability checking technique which extends the previous semantic matching approaches. In addition to this, as a guidance system for the user, a new semantic domain model is proposed that captures semantic relations between concepts in various ontologies.

Keywords: Semantic Web Services, Semantic Web Service Composition, Semantic Matching, Semantic Mapping, Semantic Inference

# ÖZ

BİRLEŞİK WEB SERVİSLERİNİN OLUŞTURULABİLMESİ İÇİN ANLAMSAL İŞLEMLER

Kardaş, Karani
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Yrd. Doç. Dr. Pınar Şenkul

Ağustos 2007, 106 sayfa

Web servislerinin birleşiminde, servis arama ve farklı servisler arasında birlikte çalışabilirlik belirlenerek uygun servislerin birleştirilmesi önemli işlerdir. Anlamsallık birleşik servis kalitesini arttırır ve bu işlemlerin otomasyonunu kolaylaştırır. Anlamsal servis arama ve servis eşleme için çeşitli yaklaşımlar vardır. Bu çalışma, ihtiyaçları ve kısıtları karşılamak amacıyla, bir çok farklı web servisinin bir araya getirilerek birleşik Web servislerinin anlamlı bir şekilde oluşturulabilmesi için yapılan çalışmaları anlatmaktadır. Bu çalışmada; Web servis birleşimi sürecini daha kolay, daha az hataya yatkın ve daha otomatik hale getirmek için önceki anlamsal yaklaşımları kullandık ve genişlettik. Bu çalışma önceki anlamsal eşleme yöntemlerini genişleten servis arama ve servis birlikte çalışabilirlik yöntemlerini içerir. Bunlara ek olarak, kullanıcıya yol gösterme amacıyla çeşitli ontolojilerde yer alan kavramlar arasındaki anlamsal ilişkileri kapsayan yeni bir anlamsal alan modeli önerilmiştir.

Anahtar Kelimeler: Anlamsal Web Servisleri, Anlamsal Web Servisi Birleşimi, Anlamsal Eşleme, Anlamsal Çıkarsama

To My Wife and Family

## ACKNOWLEDGMENTS

I would like to thank my wife and family for their great support and also I would like to thank my thesis advisor Assoc. Prof. Pınar Şenkul for her guidance and help throughout my research.

# TABLE OF CONTENTS

# LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

Today there is large number of software requirements for enterprises. These requirements are various and complex. To become more competitive, enterprises generally prefer to concentrate on only specific subjects, therefore there is a tendency to use outsourcing approaches. Service oriented architectures become more popular in order to implement outsourcing technique in information systems.

Services that are responsible for specific operations can not cover today's complex requirements. Therefore services come together to form composite services. A composite service is not transparent to the user; it acts like a simple service and covers complex business requirements.

In service composition process, finding related services and combining suitable services are important required operations. Without semantic information, service composition requires more human intervention and becomes a harder and error prone process.

Main objective of this thesis study is to increase the service composition quality and to automate service composition process by using semantic techniques. For these purposes, within the scope of this thesis, a semantic wrapper layer structure around the service repository, a set of semantic matching algorithms and a semantic domain model are proposed.

Semantic wrapper layer on the service repository is used in service discovery. It helps to discover services that are not only syntactically but also semantically compatible with the service requirements.

Semantic matching algorithms that are proposed in this work include a concept matching algorithm, a service matching algorithm and a service composability checking algorithm. Proposed concept matching algorithm, which is used to find similarities between concepts, extends the previous basic semantic matching algorithms with *intersection* and *semantic mapping* techniques. Proposed service matching algorithm is used in order to find similarities between semantic Web services. Proposed service composability checking algorithm is used in order to check the interoperability between semantic Web services.

Proposed semantic domain model is developed in order to extract more semantic information from different ontology models. It is also used for guiding the user when modeling the composition.

All of these proposed semantic processes are used in a Web service composition framework named Composite Web Service Framework (CWSF) [7, 8]. CWSF is a semi-automated Web service composition framework which gains semantic capabilities by using the proposed processes.

CWSF is an environment that is used for modeling and construction of service compositions under user's constraints. It provides modeling the Web services and resource allocation constraints in order to find executable compositions fulfilling these constraints. CWSF lets users to define compositions in a workflow manner according to user needs and constraints.

The proposed semantic processes in the thesis are used in CWSF for composition modeling, service discovery and service interoperability checking operations. In CWSF, user defines service templates by using the proposed semantic domain model as a guide. The user creates composition flow with defining service templates. When user selects a domain, actions which can be done in the domain are presented to the user. The user selects some actions as service templates. Required concepts of the selected action are listed to the user by the proposed semantic domain model for definition of user constraints. The user defines constraints for some of the attributes of required concepts. According to the constraints given by the user, candidate services are obtained. Proposed semantic wrapper layer and proposed matching algorithms are used to find related semantic Web services. Constraint checking is done by using constraint programming techniques. Once appropriate concrete services are selected, compositions are constructed by considering composabilities of candidate services by using the proposed semantic service composability checking algorithm.

Proposed semantic processes are used for finding semantic relations to make composition processes more automated. They cover all semantic requirements of CWSF. Detailed information about CWSF application is given in [7] and [8].

The structure of the thesis is as follows: In chapter two, background information and current problems of Web service composition that the thesis aims to solve, are presented. In addition, related studies about the subject are considered in chapter two. In chapter three, proposed methods and main contributions are shown in detail. Experiments and efficiency issues of the proposed algorithms are discussed in fourth chapter. Lastly in chapter five, conclusion and future work are given.

# CHAPTER 2

# PROBLEM ANALYSIS

In this chapter, basic background information about the research subject is given. Deficiencies of today’s applications about the subject are explained and problems are stated. Firstly, in section 2.1 and 2.2, basic information about Web Services and Web Service Compositions are given. In section 2.3, current problems are stated. Basic information about Semantic Web, Semantic Web Services and Semantic Web Service Composition are given in sections 2.4, 2.5 and 2.6 respectively. Finally, related works are discussed in section 2.7.

## 2.1 Web Service

Web has become an inevitable part of our daily life. It contains a large amount of data. In addition to this, it has become a medium where users seek services and applications are provided to the users. Therefore, Web services and service oriented architectures are gaining popularity.

Web services can be described as standard, modular, self-contained and self-describing applications across Web. Figure 2.1 shows the basic Web service structure [23]. As it is shown in the figure, stakeholders in Web service usage process are service provider, service requester and service broker. Service provider provides a general purpose operation. Service requesters are service users. Service broker is service registry which holds services and helps service requesters to find related services.  A sample scenario is as follows. A service provider publishes their service advertisements to the service broker. A service requester queries service broker to find a service that the service requester needs. The service broker sends service provider’s address to the service requester. Service requester can access the service provider and can begin to communicate with the service provider by using this address.

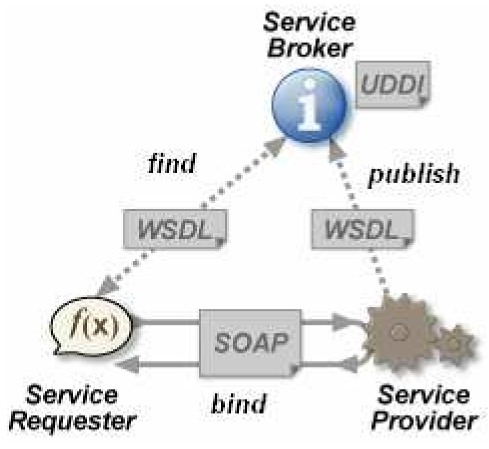


**Figure 2.1** Structure of Web Service [23]

Web services standardize ways of Web based applications using simple but effective technology infrastructure. Basic technology of Web services includes XML (eXtensible Markup Language) [25], UDDI (Universal Description Discovery and Integration) [21], WSDL (Web Service Description Language) [24] and SOAP (Simple Object Access Protocol) [20]. As shown in Figure 2.1, these core Web service structures allow businesses to communicate with each other without knowing detailed structure of each other's systems.

XML is simple, flexible text format which is used for data exchanging. Its primary purpose is to facilitate the sharing of data across different information systems in a human and machine readable format. All communication methods of Web service are in XML format. Therefore different Web services from different sources can communicate with each other easily. Web services are platform independent; they are independent from any operating system or programming language. For example, Web service applications which are created with using Java programming language [6] can communicate with Web service applications which are created with using C# programming language. Also Web service applications which are running on Windows operating system can communicate with Web service applications running on UNIX operating system.

UDDI is a platform independent, XML based industry standard Web service registry. Business organisations describe their operations across the Web and find one another on the Web with using UDDI. Companies can register and/or search Web services according to their businesses with using UDDI. UDDI defines a protocol for publishing and discovering Web services. It can be considered as a directory for storing Web service information. It provides a keyword based or category based search for Web services. Operation based discovery is not provided. UDDI describes businesses by their physical attributes such as name and address and the services that they provide. With the help of the UDDI registries, Web services can be found and then used. The role of the registry includes both storing the advertisements of capabilities and performing a match between the request and the advertisements.

WSDL and SOAP are also XML based. WSDL describes Web service in a structured way. The signature of the service operation and binding methods of service are defined in WSDL. WSDL files are easy to use and maintain. SOAP is an extensible message format in which parameter transferring protocol for Web services are defined. SOAP is a simple protocol which lets Web service applications exchange information. SOAP defines information exchange between Web service applications in a platform-independent manner.

## 2.2 Web Service Composition

The Web is moving from being a collection of pages toward a collection of services that interoperate through the Internet [18]. In today's world, users have complex requirements which may not be solved by simple atomic applications. Different applications from different vendors need to execute together to implement complex tasks for handling complex requirements. Organisations may have a capability of specific concerns. They may have a set of Web services to serve their implemented businesses. These Web services which handle a specific task, compose complex services called composite services.

A composite service is a set of services (simple and/or composite) working together to perform a goal. For example "Car Broker Composite Service" and "Traveling Composite Service" can be considered as composite services. "Car Broker Composite Service" is complete "car sale" operation which includes car dealer, financing and insurance simple (or composite) services. "Traveling Composite Service" includes plane reservation, hotel booking and shuttle reservation operations. Figure 2.2 shows the template of Traveling Composite Service. It includes Airlines Service, Hotel Service and Shuttle Service in an ordered way. It acts as a simple service but delegation to the actual service is done when needed.

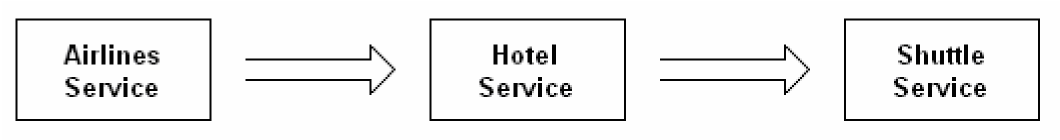


**Figure 2.2** Traveling Composite Service

Web service composition operation requires interoperability of Web services which takes part in the composition. According to the position in the composition, each service sould be interoperable between the previous and the next service. Parameter, business, constraint compatibilities can be given as examples of interoperability.

The composition of Web services requires finding Web services based on their capabilities and the recognition of these services that can be matched together to create a composition. In order to perform automated Web service composition, a reasoning system must order, combine and execute Web services that collectively achieve the user's objective. Main goal of this process is dynamic binding of Web services to existing business processes. Web service composition operation includes automatic selection, interoperation, composition and execution of Web services processes. Selection operation involves service discovery process. In this operation, appropriate services are found in a set of candidate Web services for each task according to user requirements and constraints. In interoperation process, interoperability of candidate Web services for each task is considered according to the position of the services in composition. Composition process is constructing a runnable complex Web service in such a way that interoperable set of services are selected and ordered. Execution process is run time behavior of composition. Web services are executed according to their orders in the composition.

## 2.3 Current Problems

Web service composition process involves discovery of the services that meets the requirements of the composition and determination of the interoperability among the services. It is a difficult and error prone process which generally requires human intervention. The difficulty in composition generally stems from a basic set of problems. In this part, some of the well-known problems in composition process are discussed.

As it is mentioned in “Web Service Composition” section (section 2.2), while composite service is being created, many simple or composite services are selected for each composition task. Since there may be many candidate Web services for each task, it is difficult for the user to select suitable ones according to his requirements and constraints. In addition to this, user may not have the full information about the composite service that he requires. Without detailed domain information, the user can not find proper services easily. A guiding mechanism may be needed to handle this deficiency.

Each service of a composite service may be from different service providers that use different information system structures such as processes and models. This makes the interoperability of services an important problem. In order to resolve this problem, a common metadata in machine understandable form may be used. However, in some cases it is hard to use common meta model in different organizations since each organization has different processes and its own legacy systems. In order to solve this kind of meta model distinction, mappings are needed to define between different meta models.

It is also difficult to make Web service composition process automatic by using the current Web service technologies. Basic Web service technologies which are UDDI, WSDL and SOAP are based on XML which is only machine readable and hence the semantics of a business model can not be fully expressed. Therefore, Web service composition process requires the use of semantic languages like OWL [14] and OWL-S [15]. In addition to this, currently UDDI does not provide means for describing metadata and hence it allows only a keyword-based search. In addition, UDDI is not suitable to define relationships among services, so it is hard to identify complementary services. It is also hard to form a relationship between the service and required properties to discover services according to product instance information like "I want a car but its model should be Scoda Fabia". Also it is hard to define properties such as second hand car for the related services Therefore, it needs to be augmented with additional properties in order to facilitate the composition process.

Similar to UDDI, WSDL has also deficiencies for an automated composition process. In WSDL, only the signature of the service operation can be defined. Service meaning and complex parameters meanings can not be defined in WSDL. There is not any structure in WSDL to define service metadata. Therefore, in order to improve WSDL, new service definition methods with better machine understandability must be added.

As described above, the basic reason of all these problems is the deficiency of semantics for services. Lack of semantics leads to more human intervention to composition process. By incorporating semantic information into composition process, computers can understand services, their parameters, user requirements and constraints so that compositions can be constructed automatically or semi-automatically and human intervention is decreased.

## 2.4 Semantic Web

Current Web is machine readable, not machine understandable. As a solution for this deficiency of current Web, semantic Web is considered as second generation

Web. Semantic Web is an extension of the current Web in which information is given in a well defined meaning form, for this reason, it is called as second (or next) generation Internet. This enables computers and human to work in cooperation.

Semantic Web is based on a vision of Tim Berners-Lee, the inventor of the World Wide Web. The effort behind the Semantic Web is to enrich the Web with machine understandable information by adding semantic annotation to Web documents. With the help of semantics, Web becomes machine understandable and software agents, sophisticated search engines and Web services use the Web more easily. Semantic Web provides conversion of the Web from its unorganized and human-readable form into a machine-understandable form. Search engines and other programs can understand the content of Web pages and site with the help of semantic. Since semantic Web makes information more structural, information searching and information extracting becomes easier and meaningful. Semantic Web transforms the Web into a medium through which data can be shared, understood, and processed by automated tools.

The ultimate goal of semantic Web is full automation. Semantic Web performs this automation by giving meaning to each Web resources. Content of the Semantic Web is represented by ontologies. With using ontologies, all concepts in the Web are defined in a computer understandable format.

Ontology is shared conceptualization of domains. Meaning of data is given through ontology which is used for semantic representation. Ontology is a schema for a domain. Domain concepts and relations between concepts are defined in ontology documents. In short, it is used to represent metadata of domain. Ontologies represent a shared agreement on the meaning on the terms. They provide more automated reasoning power. If applications use common ontologies, they can exchange semantic information. A common ontology defines the vocabulary among agents.

Ontology documents define and relate concepts. An ontology is generally composed of classes, properties of classes and relations between classes. Classes are formal descriptions of concepts in a domain. Properties are features and attributes of concepts. Relations are taxonomies (inheritance, disjoint relations) between concepts. An ontology provides a vocabulary that describes a domain of interest and a specification of the meaning of terms used in the vocabulary. Notion of ontology encompasses several data/conceptual models, for example, classifications, database schemas, or fully axiomatized theories. Ontologies can be used in every field in computer applications such as information integration, electronic commerce, semantic Web services, social networks, and so on.

In order to define ontology documents ontology languages are used. OWL (Web Ontology Language) [14] can be cited as the mostly used ontology language. It is based on XML, however it extends XML to describe not only the structure of the data with elements and attributes, but also to describe the data itself.

Content of a sample OWL file is shown below. In this file “Price” concept is defined as it has a string attribute and it has a relation to the concept “Unit” which is defined in another OWL file.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/unnamed.owl#"
  xml:base="http://www.owl-ontologies.com/unnamed.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Price"/>
  <owl:DatatypeProperty rdf:ID="value">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Price"/>
  </owl:DatatypeProperty>

   <owl:ObjectProperty rdf:ID="unit">
    <rdfs:domain rdf:resource="#Price"/>
    <rdfs:range
rdf:resource="file:///E://calisma/apiDeneme/sampleOwl-
SDocuments/actualServices/Unit.owl#Unit"/>
   </owl:ObjectProperty>

</rdf:RDF>
```

Domains, concepts and relations between the concepts are defined in OWL documents, as shown above.

## 2.5 Semantic Web Services

A Web service that has a semantic model is called a *semantic Web service*. Semantic Web services are defined in semantic models so that they can be searched and found more correctly according to user needs. This facilitates the interoperability of the services, as well. Therefore, automated composition of semantic Web services that includes automatic selection, composition, interoperation and execution of Web services, becomes easier and more reliable.

If Web services publish their metadata by using common ontology documents, Web services can be compared more easily, their description becomes more machine understanable and software agents can easily use these services. Ontologies can be used to describe services so that agents can advertise and discover services according to a semantic specification of functionality. To achieve more automation, standardization of ontologies, message content and message protocols will be necessary.

Defining domain specific ontologies facilitates integration of Web services. Therefore Web services are modeled and defined in ontology documents. OWL-S (Semantic Markup for Web services) [15] can be given as sample service ontology language. OWL-S expresses Web service entities. Restrictions and constraints on service descriptions can be expressed in OWL-S. It also provides the shared semantics needed to achieve interoperability. OWL-S descriptions are amenable to automated reasoning. It provides appropriate support for data types. Flexibility is provided by support for loosely structured descriptions. Software system (agent) can read and utilize the OWL-S interface without human assistance, because OWL-S provides semantic representation. It is suitable for the automatic discovery and composition of services.

Main parts of OWL-S file are service profile, service model and service grounding. Service profile describes what the service does. It is used for advertising and discovering services. It represents the functionalities of Web services. Service model describes how the service works. It is the detailed description of a service's operation. Service grounding specifies the details of how an application can access a service. Communication protocol is described in this part. It provides details on how to interoperate with a service. The structure of OWL-S is shown in Figure 2.3 [15].

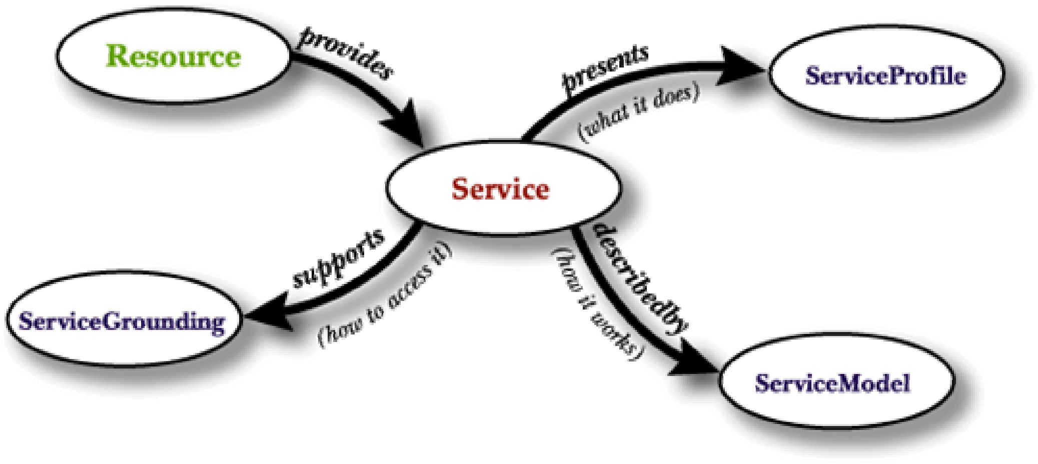


**Figure 2.3** Structure of OWL-S [15]

Content of sample OWL-S files are shown in Appendix A.

OWL and OWL-S are used to describe Web services semantically. In OWL, businesses are defined; in OWL-S, services that use these business concepts are modeled. This provides better support for service discovery, composition, invocation, choreography and monitoring. With the help of these ontology documents Web services can find each other automatically and interoperate autonomously.

## 2.6 Semantic Web Service Composition

As it is stated in Semantic Web section (section 2.4), semantic Web makes the Web machine understandable. Therefore, composition of Web services can be done in an automated way in semantic Web. If Web services, requirements and constraints are defined semantically, computers can compose Web services according to requirements and constraints with using automatic reasoning and inference techniques.

As it is mentioned in section 2.2, the composition of Web services requires finding Web services based on their capabilities and the recognition of these services that can be matched together to create a composition. Besides Web service discovery, it includes invocation, composition and interoperation, and Web service execution processes. In Semantic Web service composition, the ultimate goal is to fully automate all of these processes.

There are four conceptually separate phases in composition of semantic Web services. These are Specification, Matchmaking, Selection, and Generation. Specification is the definition of composition by the user. Matchmaking is composite plan generation. Selection is selection of best plan according to quality of service compositions. Generation is generation of the executable code.

## 2.7 Related Works

There are several related works in the literature on semantic modeling for web service composition [12, 19, and 22], semantic service repositories [10, 17] and service matching [9, 18]. In this section these related works are examined. Their contributions to problems that are stated in Section 2.3 are determined.

In [12], an ontology based Web service composition method is proposed. Instead of a standard Web ontology language such as OWL-S, they propose a new ontology model in which service model is defined over WSDL for Web service composition in order to define service attributes such as message, service, quality, operation and parameter. However this new ontology brings standardization problems.

Matchmaking is done by Matchmaker module according to predefined composability rules. Matchmaker module takes the specification of composition as input and generates composition plans. Two services are composable if their predefined attributes such as purposes, domains, messages, operation modes and bindings are compatible. Messages of services are compatible if number of parameters, their data types, business roles, and units of services are same. Each created composition is stored as stored template. Also compositions defined by domain experts are considered as stored templates. While a new composition is created, it is examined whether the composition is a subgraph of a stored template. If so, it is decided that the composition is useful.

Best composition plan is selected from many composition plans according to quality properties such as fee, security, privacy, time, availability and latency. Composite service is generated from the selected plan.

Matchmaking process needs much processing time. Therefore, according to their performance tests, most of the time is spent on checking composability in composition process.

In METEOR-S [22], a Web service composition framework with features such as dynamic failure handler and reconfiguration is proposed. This framework is proposed for configuring and executing dynamic Web processes. Web processes are workflows created using Web services. They correspond to the Web service compositions in our study. They use integer linear programming approach as composition method. They define semantic model for functional capabilities which describe service and non functional capabilities which describe service constraints of Web services.

This study contains failure handling mechanism which is not considered in the thesis. In cases of Web service failures, an approach is proposed which can reconfigure the process at run time, without violating the process constraints. Parts of the process may be reconfigured in the case of error during invocation.

To handle data mismatches between different suppliers, it includes an ontology mediator which handles the mapping between ontologies. They also implement interaction protocol mediator for different interaction (business) protocols of different suppliers. Our work puts emphasis on semantic service discovery, matching and modeling guidance in composition context.

In [19], a semi automatic composition generation method is proposed. The generated composition is directly executable through the WSDL grounding of the services. It has capability to find matching services. The matching of two services is done using the information in the service profile. It is an OWL reasoner built on Prolog.

In [10, 17], it is shown that OWL-S and UDDI complement each other. In order to add semantic capabilities, a mapping is defined between OWL-S document and UDDI registry record. They map some fields between OWL-S document and UDDI registry record. OWL-S / UDDI translator module perform this operation. However they do not propose a complete mechanism for semantic queries. In this work, we propose a different approach for adding semantic capabilities for Web service registry.

In [18], a new semantic similarity algorithm is proposed. It defines various degrees on similarity on the basis of the inheritance relation in ontology model. The result of the match depends on the degree of similarity between the concepts in the match. The degree of match is determined by the minimal distance between concepts in the taxonomy tree. There are four kinds of similarity levels. Exact matches are preferrable, Plug-in matches are the next best level. Subsumes is the third best level. Fail is the lowest level and it represents an unacceptable result. The proposed semantic matching operation in our study is based on this algorithm. We extend this matching approach with new features.

In [9], Racer Description Logic reasoner is used for semantic matching. Description Logic reasoner is used to compare ontology based service descriptions. Description Logics are knowledge representation formalisms. They are based on the notion of concepts (unary predicates, classes) and roles (binary relations), and are mainly characterized by constructors that allow complex concepts and roles to be built from atomic ones. A DL reasoner can be used as matcher module because it can check whether two concepts subsume each other. Intersection level is considered as a matching level in a different manner from our approach.

COSS [2] is context aware Web service composition system. For service discovery and matching, context information is utilized. Context is defined as a situation of a person, place or object that is relevant to the user and application. Location, Speed, Time, Personal Interests, weather forecast and car status can be given as examples of context. It is stated that considering the context can improve the quality of the matching process because recall and precision rates improve with using context information in service discovery process. Context providers are implemented to provide context inputs. GPS and weather station are examples of context providers.

COSS uses ontologies to model contexts and interrelations among them. Also COSS uses ontologies for semantic matchmaking. The matchmaking algorithm which is based on distance operation like [18] can handle synonyms like ‘buy’ and ‘purchase’ and homonyms like ‘order’ which has more than one meaning.

COSS service matching algorithm is as follows: The first step is considering service types and service outputs. If one of these conditions is broken, then there is no match. The second step is considering inputs of services. Mismatch input number is considered and they are inspected whether they can be gathered from context providers. As a last predefined properties like place or price are considered as matching criteria. According to missing attributes and inputs; services are ranked.

Context input is provided by implemented context providers. However implementing or finding a previously developed context providers are difficult in real applications and thus not practical.

WordNet [4] is a semantic model that captures semantic relations in English words. It has similar features to the proposed semantic domain model. However proposed semantic domain model is for composition purpose and has many different features.

Proposed matching algorithms of these studies take only inheritance hierarchy into consideration. However, attribute similarity is also an important factor in matchmaking process. For example, there is a relation between camera and cell phone with camera capability. Both of them have taking photograph property. This relation can not be found with matching algorithms that only takes inheritance hierarchy into consideration, unless any inheritance relation is created. On the other hand, defining an inheritance relation between these two entities does comply with object orientation rules. Also in inheritance relation while distance increases, similarity decreases. Another deficiency of many of the current semantic matching algorithms is not to consider predefined mappings. Semantic matching algorithms can improve their precision and recall values with considering mappings because in real applications there is not any standard, common ontology. Also in these studies, there is not any guiding mechanism for ordinary users while they are creating the service composition.

# CHAPTER 3

# SEMANTIC PROCESSES FOR CONSTRUCTING COMPOSITE WEB SERVICES

In this section our proposed and implemented approaches for semantic Web service composition are explained in detail. Basic contributions of this work can be listed as follows:

1- A wrapper layer around UDDI registry is added in order to provide semantic search capabilities.

2- For semantic service discovery and interoperability checking, previous semantic matching algorithms are enhanced and augmented with intersection and semantic mapping techniques.

3- In order to extract more semantic information and to guide the user in modeling the composition, a semantic domain model is proposed.

In this section, firstly Semantic Wrapper Layer and its function are described in Section 3.1. Then, proposed matching algorithms are explained in Section 3.2. In Section 3.3, service composability checking is described and how matching algorithms are used in service composability checking operation is shown. In Section 3.4 semantic mapping method is determined. In Section 3.5, proposed inference method and its tasks are illustrated. Lastly, in Section 3.6, implementation issues are stated.

## 3.1 Semantic Wrapper Layer

As it is stated in current problems section (Section 2.3), current UDDI registries are not semantic based and can only perform keyword based searches. Although there is a huge effort towards development of a semantic Web service registry as in [10, 17], there is not any standard semantic Web service registry yet. Therefore, in this study, keyword based UDDI registry is used for semantic Web services; however, it is extended with a wrapper layer to add semantic search capabilities.

To implement semantic service discovery, we add semantic wrapper layer which adapts UDDI registry as a semantic registry. In order to keep a service's semantic information (OWL-S file) URI, tModel field of UDDI registry is used. Metadata of services can be residing in any path. Only URIs are put in the service records of UDDI registry. While service providers register services to registry, they set service OWL-S file URI to "tModel" field of service record. By this way, in the service registry, service semantics are related with the advertised services.

Another extension that supports the semantic search is generation of a set of semantically close keywords from a single keyword. When a query of Web service is requested from the semantic wrapper layer with a keyword and a service template, firstly, given keyword's similar concepts are obtained by using semantic matching module. These similar concepts and the original keyword are added to keyword list. Thus, from a keyword, a set of keywords are obtained. For example, if the user wants to search Vehicle; Car, Limousine and Truck are searched too because all of them are a kind of Vehicle. After keyword list is prepared, UDDI registry is queried with this set of keywords. UDDI search finds set of candidate Web services for each keyword in the keyword list. After candidate services are obtained, their semantic files which are OWL-S documents are accessed from the URI which is stored in "tModel" field of the service record. Once more, semantic matching module is used in order to find out whether candidate services are appropriate. In this case, semantic matching module's service matching algorithm, which compares service template and gathered semantic files of candidate services, is used. (Service template is a definition of desired service with expected inputs and outputs.) Service discovery process is shown in Figure 3.1.

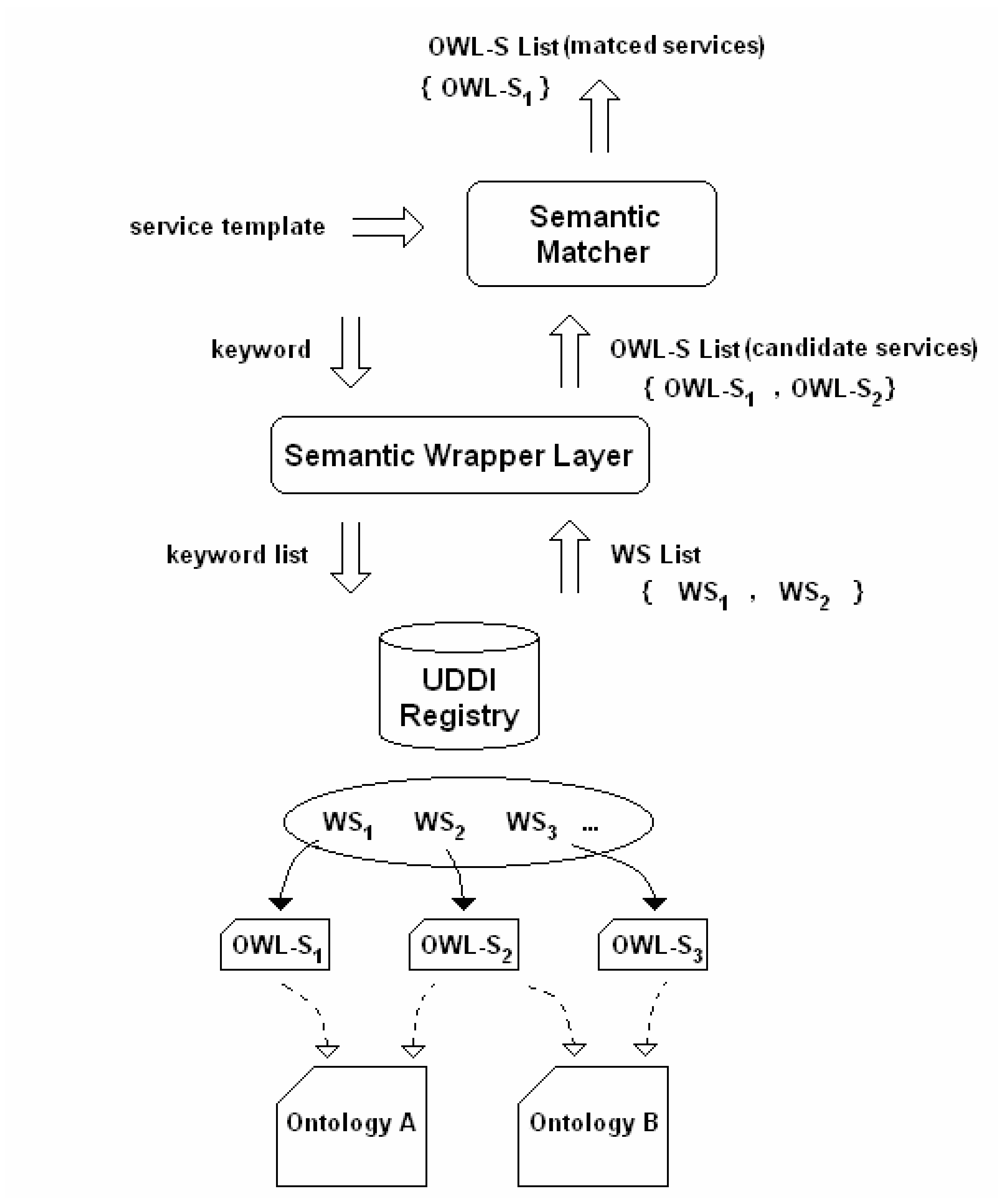


**Figure 3.1** Semantic Service Querying

As it is stated above searching service operation is done in two steps: Firstly, text based search in UDDI registry is done. This is used for rough filtering. Then OWL-S files are accessed and matching is considered. By this method semantic descriptions of services can be accessed when they are needed. From keyword based UDDI registry, semantic Web services are accessed. Thus this additional layer over UDDI allows semantic discovery.

## 3.2 Semantic Matching

Semantic Matching is used for finding similarities between semantic concepts or semantic services. Finding similarities between concepts facilitates machine understandability. By finding similarities between concepts, it can be inferred whether given concepts are the same or they are replaceable. Forming a keyword based similarity considers only text based similarities. This method may lead to incorrect and missing results. Semantic matching is used in both service discovery and service composability cheking operations.

There are many services in registries. Which ones are suitable for the user requirements and constraints? Which ones are suitable for interoperability with each other? These two questions show the necessity of semantic service discovery operation in service composition process. Today, most of the existing service discovery applications perform keyword based search. This decreases the quality of matching and causes mismatches between requested and found services.

Keyword based search takes only syntactic similarity into consideration. However by using this method, similarity can not be found between syntactically different but semantically same or similar concepts, which are called synonyms. Therefore, this causes finding fewer results. In addition to this, by using this method, similarity can be found wrongly between syntactically same but semantically different concepts, which are called homonyms.

Since semantic matching takes meaning of concepts into consideration instead of their labels, semantic matching (capability matching) provides discovery of more relevant results. By using semantic matching, similarities between concepts can be found completely and correctly. Because of all concepts' meaning is explicitly defined, computers can understand concepts and can evaluate similarities.

Synonyms and homonyms can be detected easily. In addition, contrary concepts can be inferred as well.

By reasoning on ontology documents, matchings that would not match syntactically can be derived. For instance, let's consider a restaurant that serves 'meal' (e.g. foods, desserts). If the user wants to order a 'kebab', a syntactic mismatch occurs. But there is a semantic relation between 'kebab' with 'food' in domain ontology. There is an inheritance relation between these concepts as it is shown in figure 3.2. So it is derived that 'kebab' is a kind of 'food'. This derivation is a semantic match. As it is stated in [2] this requires that service requests and service descriptions are not described by using keywords but by their properties which are related to the concepts from the shared ontology.

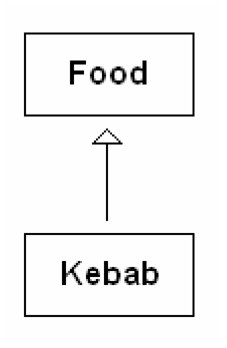


**Figure 3.2** Ontology Model for Food and Kebab concepts

In this work, semantic matching is used in both service discovery and service composability operations. Proposed semantic matching method includes two matching algorithms. First one is concept matching algorithm and the second one is service matching algorithm.

### 3.2.1 Concept Matching

Concept matching operation is finding similarity degree between two concepts. The proposed concept matching algorithm is based on the matching algorithm given in[18], in which concepts are compared according to their hierarchy in ontology model. This hierarchy is inheritance relation between concepts. This algorithm is extended in this work in such a way that the similarity between attributes of concepts and predefined mappings are considered as well. If attributes are similar or if there is a previously defined mapping, it is inferred that concepts are similar.

The proposed concept matching algorithm is as follows: Let $C_1$ and $C_2$ be given concepts and the aim is to find similarity degree of $C_2$ to $C_1$. As the first operation inheritance relation is considered. $C_2$ is searched in ontology documents in order to find similarities in the model hierarchy. If $C_1$ and $C_2$ are the same concepts or $C_2$ inherits $C_1$ (which is "is-a relation" between $C_2$ and $C_1$ and means "$C_2$ is a $C_1$") then the similarity degree of the match is called "exact match". If there is an already available mapping from $C_2$ to $C_1$, then similarity degree is considered as "maps match". If $C_2$ still inherits $C_1$, but hierarchy path is of more than one level, then similarity degree between them is considered as "plug-in match". On the contrary if $C_1$ inherits $C_2$, then the similarity degree is considered as "subsume match". If there is not any hierarchy relation between concepts, then as a second operation attribute similarity is considered. If $C_1$ and $C_2$ have some common/similar (this similarity also requires running matching algorithm recursively on attributes), then these concepts are considered as partially overlapping and the similarity degree of match is "intersection match". The intersection indicates that only a part of concepts are similar. In this case matching hit ratio is calculated to find the ratio of similar attributes. Matching hit ratio of $C_2$ to $C_1$ similarity is the ratio of the number of similar attributes of $C_1$ and $C_2$ to the number of all attributes of $C_1$. If there is not any hierarchy and parameter similarity, then these concepts are considered as unmatched and their similarity degree becomes "disjoint match". "maps match" and "intersection match" methods are extensions to the matching algorithm of [18].

Below, similarity degrees are listed.

- exact matching
  - *$C_1$ and $C_2$ are same, equivalent or "$C_2$ extends" $C_1$ (one step)*
- maps matching
  - *There is a mapping defined from $C_2$ to $C_1$*
- plugin matching
  - *if $C_2$ is subconcept of $C_1$*
- subsume matcing
  - *if $C_2$ is superconcept of $C_1$*
- intersection matching
  - *if there is an intersection between $C_1$ and $C_2$*
- disjoint matching
  - *no relation between $C_1$ and $C_2$*

Concept matching algorithm and functions that is used are shown below. If concepts are not same or synonyms and there is not any defined mapping, maximum similarity between inheritance checking and intersection checking is returned.

```
//this function is for finding a similarity from c2 to c1
areConceptsMatch(Concept c1, Concept c2):Matching Degree{
      if(areConceptsSame(c1, c2)) return EXACT
      if(isThereAnyMapping(c2, c1)) return MAPS
      matchDegreeForInheritanceChecking =
            inheritanceChecking(c1, c2)
      matchDegreeForIntersectionChecking =
            intersectionChecking(c1, c2)
      return  maxSimilarity(matchDegreeForInheritanceChecking,
                  matchDegreeForIntersectionChecking)
}
```

```
//two concepts are same if they have same names and same ontology
// paths or they are synonyms
areConceptsSame(Concept c1, Concept c2):boolean{
      return (c1.getName().equals(c2.getName()) AND
            c1.getOntologyPath().
            equals(c2.getOntologyPath())) OR
            areConceptsSynonyms(c1, c2)
}
```

```
//predefined mappings are searched in this function
isThereAnyMapping(Concept c1, Concept c2):boolean{
      for each mapping of defined mappings
```

```
            if(mapping.from().equals(c1) AND
                  mapping.to().equals(c2)) return true
      return false
}
```

```
areConceptsHomonyms(Concept c1, Concept c2):boolean{
      return c1.getName().equals(c2.getName()) AND
            (NOT(c1.getOntologyPath().
                  equals(c2.getOntologyPath())))
}
```

```
areConceptsSynonyms(Concept c1, Concept c2):boolean{
      return (NOT(c1.getName().equals(c2.getName()))) AND
            c1.getOntologyPath().equals(c2.getOntologyPath())
}
```

```
inheritanceChecking(Concept c1, Concept c2):Matching Degree{
      if(areConceptsSame(c1, c2.getSuperConcept()))
            return EXACT
      for each super concept of C2
            if(areConceptsSame(super concept, c1)
                  return PLUG_IN
      for each super concept of C1
            if(areConceptsSame(super concept, c2)
                  return SUBSUME
      return FAIL
}
```

```
intersectionChecking(Concept c1, Concept c2):
                     Matching Degree, hit ratio{
      for each attribute of c1
            for each attribute of c2{
                  matchingDegree =
                        areConceptsMatch(attribute of c1,
                                         attribute of c2)
                  if(matchingDegree >= ACCEPTABLE_DEGREE){
                        matchedAttributes++
                        break // for one “for loop”
                  }
            }
      hit ratio = 100 * matchedAttributes /
                        c1.getAttributeSize()
      if(hit ratio == 0) return FAIL
      return INTERSECTS, hit ratio
}
```

To illustrate the algorithm degrees, sample ontology models are shown in Figure 3.3, 3.4, 3.5 and 3.6.

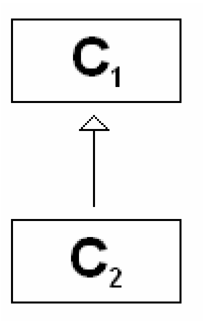


**Figure 3.3** Sample Ontology Model

In Figure 3.3, similarity degree of $C_2$ to $C_1$ is exact because $C_2$ is a $C_1$. Similarity degree of $C_1$ to $C_2$ is subsume.

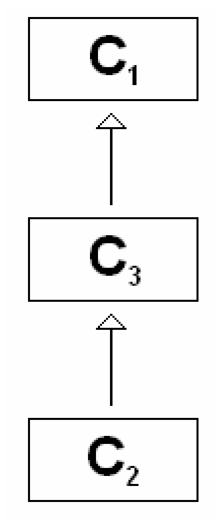


**Figure 3.4** Sample Ontology Model

In Figure 3.4, similarity degree of $C_2$ to $C_1$ is plug in because again $C_2$ is a $C_1$ but in this case hierarchy path is two levels.

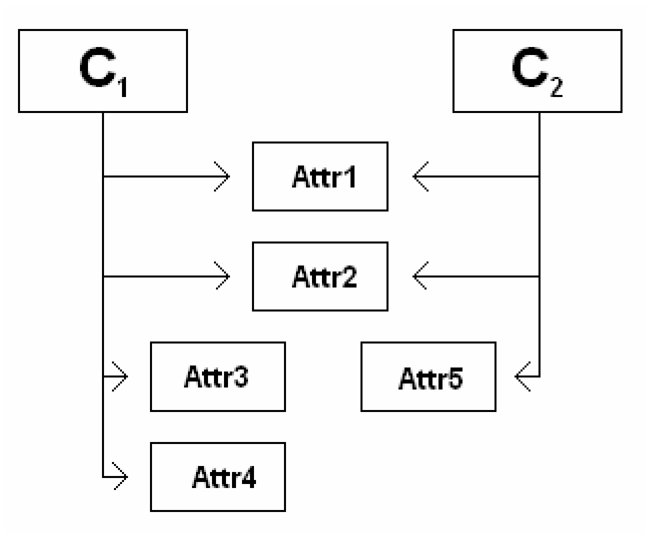


**Figure 3.5** Sample Ontology Model

In Figure 3.5, there is not any hierarchy relation between $C_1$ and $C_2$. In this case attribute similarity is considered. There is an intersection between them because both of them have Attr1 and Attr2 attributes. However their matching hit ratios are different. Matching hit ratio of $C_2$ to $C_1$ similarity is 50% because $C_2$ can cover two of four attributes of $C_1$. Matching hit ratio of $C_1$ to $C_2$ similarity is 66.7% because $C_1$ can cover two of three attributes of $C_2$.

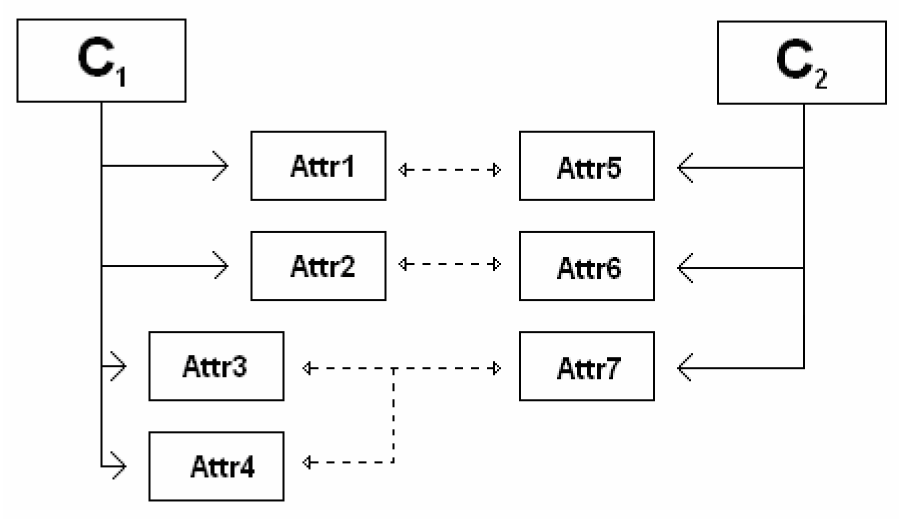


**Figure 3.6** Sample Ontology Model

In figure 3.6, there is not any hierarchy and intersection relation between $C_1$ and $C_2$. However, there is a predefined mapping between them. Therefore these concepts are similar.

To show how the algorithm finds results, some examples can be given. With using House.owl, Meal.owl and Travel.owl ontology documents [Appendix A], the concept matching algorithm is run for some samples. Obtained results are listed below:

(In the below examples, the goal is to find the similarity of concept2 to the concept1.)
*For concept1 is Room and concept2 is Room, matching result is 'EXACT'*
*For concept1 is Parquet and concept2 is Laminant, matching result is 'EXACT'*
*For concept1 is Laminant and concept2 is Parquet, matching result is 'SUBSUME'*
*For concept1 is Room and concept2 is Bathroom, matching result is 'INTERSECTION (Hit ratio % : 100)'*
*For concept1 is Floor and concept2 is Parquet, matching result is 'EXACT'*
*For concept1 is Wooden and concept2 is Parquet, matching result is 'SUBSUME'*
*For concept1 is Floor and concept2 is Wooden, matching result is 'PLUG_IN'*
*For concept1 is Wooden and concept2 is Floor, matching result is 'SUBSUME'*
*For concept1 is Room and concept2 is Bathroom, matching result is 'INTERSECTION (Hit ratio % : 100)'*
*For concept1 is Room and concept2 is Wall, matching result is 'INTERSECTION (Hit ratio % : 40)'*
*For concept1 is Wall and concept2 is Room, matching result is 'INTERSECTION (Hit ratio % : 100)'*
*For concept1 is Roof and concept2 is Door, matching result is 'FAIL'*
*For concept1 is Terrace and concept2 is Room, matching result is 'INTERSECTION (Hit ratio % : 100)'*

*For concept1 is Room and concept2 is Terrace, matching result is 'INTERSECTION (Hit ratio % : 80)'*
*For concept1 is Fiyat and concept2 is Price, matching result is 'MAPS'*
*For concept1 is FruitJuice and concept2 is Drink, matching result is 'SUBSUME'*
*For concept1 is Drink and concept2 is FruitJuice, matching result is 'PLUG_IN'*
*For concept1 is Pizza and concept2 is Food, matching result is 'SUBSUME'*
*For concept1 is Food and concept2 is Pizza, matching result is 'EXACT'*
*For concept1 is OrangeJuice and concept2 is Drink, matching result is 'SUBSUME'*
*For concept1 is OrangeJuice and concept2 is WithoutAlcohol, matching result is 'SUBSUME'*
*For concept1 is OrangeJuice and concept2 is FruitJuice, matching result is 'SUBSUME'*
*For concept1 is FruitJuice and concept2 is OrangeJuice, matching result is 'EXACT'*
*For concept1 is WithoutAlcohol and concept2 is OrangeJuice, matching result is 'PLUG_IN'*
*For concept1 is Drink and concept2 is OrangeJuice, matching result is 'PLUG_IN'*
*For concept1 is ArrivalTime and concept2 is DepartureTime, matching result is 'INTERSECTION (Hit ratio % : 100)'*
*For concept1 is DestinationPlace and concept2 is DeparturePlace, matching result is 'INTERSECTION (Hit ratio % : 100)'*
*For concept1 is Price and concept2 is DeparturePlace, matching result is 'FAIL'*

EXACT, PLUG-IN, SUBSUME matching results are found by considering inheritance relation between concepts. Because of inheritance relation is defined clearly in House.owl ontology document, the validity of these results can be seen from the ontology document easily. Inheritance hierarcy of some concepts in House ontology are shown in Figure 3.7. Considering this figure, EXACT, PLUG-IN and SUBSUME matching results can be discovered easily. For example, since there is a single step inheritance relation between Parquet and Floor concepts, Parquet is similar to Floor as EXACT matching degree. Reverse is SUBSUME matching degree. Floor is similar to Parquet as SUBSUME matching degree. If there is more than single step inheritance relation between concepts, the matching degree is PLUG-IN. Laminant is similar to Floor as PLUG_IN matching degree.

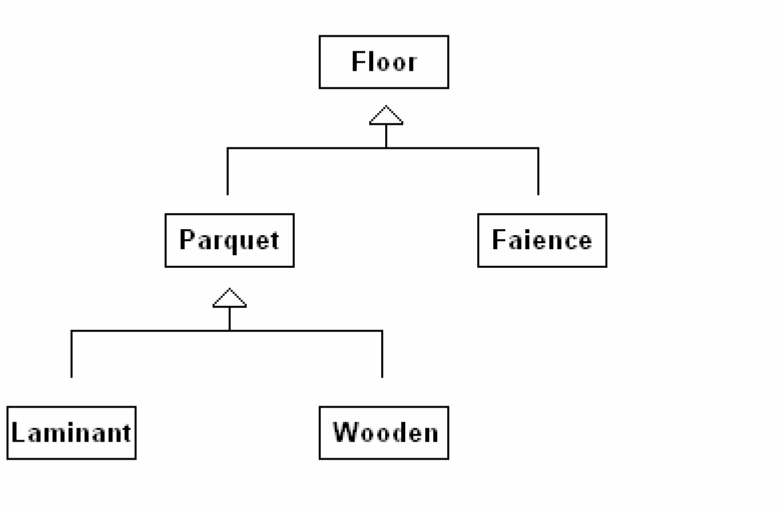


**Figure 3.7** Ontology Model for “Floor” concept

Precision of inheritance based matchings are 100%. Therefore, all semantic matching methods based on inheritance relation finds same results for EXACT, PLUG-IN, SUBSUME matchings. However, the recall of this matching method is low if there is limited inheritance relation between concepts.

INTERSECTION matching results of the matching algorithm are based on attribute relation between concepts. By using this method, more number of similarities between concepts can be inferred and the recall value can be improved. As it is shown in Figure 3.8, intersection based matching algorithms can find similarities between Room and Wall concept pairs in which there is not any inheritance relation. They have similar properties which can not be determined by inheritance based matching methods.

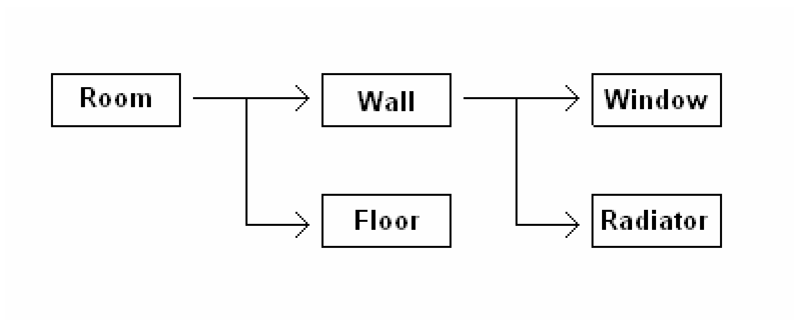


**Figure 3.8** Ontology Model for “Room” concept

Matching result MAPS shows that there is a predefined mapping for the concepts. For example because of there is a mapping which is defined from Price concept to Fiyat concept, the matching result for them is MAPS.

By using intersection method, similarities can be found between irrelevant concepts. For example by this method, similarity between Room and Wall objects are found. But using Wall concept instead of Room concept may be inadequate, because Wall concept can not cover all properties of Room concept. But reverse is more meaningful, Room concept can be used instead of Wall concept because Room concept can cover all properties of Room concept. To differentiate this state, hit ratio parameter is used. Hit ratio parameter shows how much percentage requested concept’s parameters can be covered by found concept. High hit ratio means high similarity. Similarities which have low hit ratios can be presented as suggestions to users. For example, if user requests camera, cell phones which has requested camera capabilities can be added to the results. Because found cell phones cover camera’s all properties. However, if the user requests cell phone which has camera capabilities, cameras can be added to the results as suggestions. Because camera can not cover cell phone’s all properties.

The proposed concept matching algorithm can also handle homonyms and synonyms. In this study, data class which is created for concepts consists of both concept name and ontology model of concept. Homonyms can be handled because of they have different ontology models. If names of concepts are same and ontology models of them are different, they are considered as homonyms. If names of concepts are different and ontology models of them are same, they are considered as synonyms.

### 3.2.2 Service Matching

Proposed service matching algorithm for semantic Web services compares the capabilities provided by any of the advertised services with the capabilities needed by the requester. Capability matching algorithms use the service descriptions in the service profile. Proposed semantic service matching algorithm is based on considering service's input and output parameters. If inputs and outputs of services are similar, it is inferred that the services are similar. As they define the outcome of a service, output parameter is more decisive on this process. Therefore, if outputs are not similar, it is directly inferred that the services are unmatched. Similarity between outputs and inputs are determined by using the concept similarity algorithm as described above. Proposed service matching algorithm uses proposed concept matching to find similarity between parameters.

The proposed service matching algorithm is as follows. Let $S_1$ and $S_2$ be given web services. $S_1$ is a concrete service in a service registry and $S_2$ is an abstract service requested by the user. The aim is to find similarity degree between $S_1$ and $S_2$. As the first step, output parameters are compared. If they are not similar, it is inferred that the services are "unmatched". If the outputs are similar but only some of the inputs are partially similar, then it is inferred that the services "intersect". The intersection indicates that the services are partially similar. If the outputs and all inputs are matched (no missing inputs), then it is inferred that there is an "exact match" between the services.

Below similarity degrees of service matching algorithms are listed:

- exact service matching
    - *outputs and inputs are exact*

- intersection service matching
    - *if outputs are exact and inputs are similar*
- no service matching
    - *outputs are disjoint*

Service matching algorithm is shown below. For each parameter of service, proposed concept matching algorithm is used. Similarity of each parameter is compared with acceptable and min. acceptable match degrees. These acceptable degrees which includes hit ratio of intersections, can be adjustable according to the needs. Similarities bigger than acceptable match degree are considered as matchings. Similarities bigger than min. acceptable match degree are considered as suggestions.

```
areServicesMatch(WebServiceTemplate wsTemp, WebService ws):
      Service Matching Degree, service hit ratio{
      if(areConceptsMatch(wsTemp.getOutput(),
            ws.getOutput()) = FAIL)
                  return SERVICE_FAIL, 0
      for each primitive ontology object of ws
            for each primitive ontology object of wsTemp{
                  concept match degree = areConceptsMatch(
                        primitive ontology object of WSTemp,
                        primitive ontology object of WS)
                  if(concept match degree >=
                        ACCEPTABLE_MATCH_DEGREE){
                        matchedParameters++
                   (primitive ontology object of ws).addMatch
                       (primitive ontology object of wsTemp)
                  }else if(concept match degree >=
                        MIN_ACCEPTABLE_MATCH_DEGREE)
                  (primitive ontology object of ws).
                        addSuggestedMatch
                  (primitive ontology object of wsTemp)
            }
      service hit ratio = 100 * matchedParameters /
```

```
                  wsTemp.getParameterSize()
      if(service hit ratio == 0)
            return SERVICE_FAIL, service hit ratio
      else if(service hit ratio == 100)
            return SERVICE_EXACT, service hit ratio
      else return SERVICE_INTERSECTS, service hit ratio
}
```

Semantic service matching algorithm can be used in applications in a different way. In real applications it is not important whether two services are similar or not. In real applications there are some parameters and the goal is to find services that can match the parameters. In other words, the most important issue in service matching is to find out whether candidate service can cover requested parameters of requested service. This operation’s implementation is as follows: as the first operation, all primitive attributes of all candidate services are found. All primitive attributes of a service contains each parameter’s primitive attributes. Each parameter has its own primitive attributes, primitive attributes of each object attribute and all primitive attributes of super concepts. The reason of finding primitive types is that primitive types are the only types in which values can be set. Also all primitive types of each requested parameters are found. For each primitive type of service, two lists are created. In one of them, candidate requested primitive types which are similar concepts are put, in the other, suggested requested primitive types which are not similar but suggestable concepts are put. Each candidate service’s primitive type is compared with each requested primitive type of abstract service by concept matching algorithm to find similarity between them. If there is high similarity, then requested primitive type is put into service primitive type’s candidate list. If there is low similarity degree but service primitive type can cover requested primitive type, then requested primitive type is put into service primitive type’s suggested list. These suggested matches are presented to the user to obtain user intervention when high similar matches could not be found. If all primitive types of service has candidate or suggested primitives, then we can say that service is matched.

Similarity degrees of resulting matches for the composition are used for determining the quality of service composability. Generated service compositions are ranked in descending similarity degrees.

With using Travel.owl ontology document and; PegasusService.owl, THYService.owl and AtlasJetService.owl OWL-S files [Appendix A], the service matching algorithm is run for some samples. The obtained results are listed below:

<u>*Requested Service Parameters*</u>

*- **parameterName**: myArrivalTime **parameterType**: ArrivalTime.Time **value**: 14/05/2007*

*- **parameterName**: myDepartureTime **parameterType**: DepartureTime.Time **value**: 13/05/2007*

*- **parameterName**: myDestinationPlace **parameterType**: DestinationPlace.Place **value**: Izmir*

*- **parameterName**: myDeparturePlace **parameterType**: DeparturePlace.Place **value**: Ankara*

For given required parameters, service matching results are as follows:

***Matching Results:***

*1) For PegasusService, matching result is 'SERVICE_EXACT'*

<u>*Matched Service Parameters*</u>

***Service parameterName**: time **parameterType**: Time.DepartureTime*

<u>*Candidate concepts for service parameter :time :::*</u>

***candidate parameterName**: myDepartureTime **parameterType**: DepartureTime.Time **value**: 13/05/2007*

<u>*Suggested concepts for service parameter :time :::*</u>

***suggested parameterName**: myArrivalTime **parameterType**: ArrivalTime.Time **value**: 14/05/2007*

***Service parameterName:** time **parameterType**:ArrivalTime.Time*

<u>*Candidate concepts for service parameter :time :::*</u>

***candidate parameterName**: myArrivalTime **parameterType**: ArrivalTime.Time **value**: 14/05/2007*

<u>*Suggested concepts for service parameter :time :::*</u>

***suggested parameterName:** myDepartureTime **parameterType**: DepartureTime.Time **value**: 13/05/2007*

***Service parameterName:** place **parameterType**: DeparturePlace.Place*

<u>*Candidate concepts for service parameter :place :::*</u>

***candidate parameterName:** myDeparturePlace **parameterType**: DeparturePlace .Place **value**: Ankara*

<u>*Suggested concepts for service parameter :place :::*</u>

***suggested parameterName:** myDestinationPlace **parameterType**: DestinationPlace.Place **value:** Izmir*

***Service parameterName:** place **parameterType**:DestinationPlace.Place*

<u>*Candidate concepts for service parameter :place :::*</u>

***candidate parameterName**: myDestinationPlace **parameterType**:DestinationPlace.Place **value**: Izmir*

<u>*Suggested concepts for service parameter :place :::*</u>

***suggested parameterName**: myDeparturePlace **parameterType**: DeparturePlace.Place **value**: Ankara*

*2) For THYService, matching result is 'SERVICE_INTERSECTION (Hit ratio % : 50)'*

<u>*Matched Service Parameters*</u>

***Service parameterName**: place **parameterType**: DeparturePlace.Place*

<u>*Candidate concepts for service parameter :place :::*</u>

***candidate parameterName**: myDeparturePlace **parameterType**: DeparturePlace.Place **value**: Ankara*

<u>*Suggested concepts for service parameter :place :::*</u>

***suggested parameterName**: myDestinationPlace **parameterType**: DestinationPlace.Place **value**: Izmir*

***Service parameterName**: time **parameterType**: Duration.DepartureTime.Time*

*There is not any candidate concept for service parameter :time*

<u>*Suggested concepts for service parameter :time :::*</u>

***suggested parameterName**: myArrivalTime **parameterType**: ArrivalTime.Time **value**: 14/05/2007*

***suggested parameterName**: myDepartureTime **parameterType**: DepartureTime.Time **value**: 13/05/2007*

***Service parameterName**: time **parameterType**: Duration.ArrivalTime.Time*

*There is not any candidate concept for service parameter :time*

<u>*Suggested concepts for service parameter :time :::*</u>

***suggested parameterName**: myArrivalTime **parameterType**:ArrivalTime.Time **value**: 14/05/2007*

***suggested parameterName**: myDepartureTime **parameterType**: DepartureTime. Time **value**: 13/05/2007*

***Service parameterName**: place **parameterType**: DestinationPlace.Place*

<u>*Candidate concepts for service parameter :place :::*</u>

***candidate parameterName**: myDestinationPlace **parameterType**: DestinationPlace.Place **value**: Izmir*

<u>*Suggested concepts for service parameter :place :::*</u>

***suggested parameterName**: myDeparturePlace **parameterType**: DeparturePlace.Place **value**: Ankara*

*3) For AtlasJetService, matching result is 'SERVICE_EXACT'*

<u>*Matched Service Parameters*</u>

***Service parameterName**: place **parameterType**: DestinationPlace.Place*

<u>*Candidate concepts for service parameter :place :::*</u>

***candidate parameterName**: myDestinationPlace **parameterType**: DestinationPlace.Place **value**: Izmir*

<u>*Suggested concepts for service parameter :place :::*</u>

***suggested parameterName**: myDeparturePlace **parameterType**: DeparturePlace.Place **value**: Ankara*

***Service parameterName**: place **parameterType**: DeparturePlace.Place*

<u>*Candidate concepts for service parameter :place :::*</u>

***candidate parameterName**: myDeparturePlace **parameterType**: DeparturePlace.Place **value**: Ankara*

<u>*Suggested concepts for service parameter :place :::*</u>

***suggested parameterName**: myDestinationPlace **parameterType**:DestinationPlace.Place **value**: Izmir*

***Service parameterName**: time **parameterType**:AtlasJetArrivalTime.Time*

<u>*Candidate concepts for service parameter :time :::*</u>

***candidate parameterName**: myArrivalTime **parameterType**:ArrivalTime.Time **value**: 14/05/2007*

<u>*Suggested concepts for service parameter :time :::*</u>

***suggested parameterName**: myDepartureTime **parameterType**: DepartureTime.Time **value**: 13/05/2007*

***Service parameterName**: time **parameterType**: AtlasJetDepartureTime.Time*

<u>*Candidate concepts for service parameter :time :::*</u>

***candidate parameterName**: myDepartureTime **parameterType**: DepartureTime.Time **value**: 13/05/2007*

<u>*Suggested concepts for service parameter :time :::*</u>

***suggested parameterName**: myArrivalTime **parameterType**: ArrivalTime.Time **value**: 14/05/2007*

As it is shown above, requested service parameters are “ArrivalTime” object parameter, “DepartureTime” object parameter, “DeparturePlace” object parameter and “DestinationPlace” object parameter. The structures of these objects are shown in Figures 3.9, 3.10, 3.11 and 3.12 respectively. “ArrivalTime” and “DepartureTime” concepts has “Time” concept as object property. “DeparturePlace” and “DestinationPlace” concepts has “Place” concept as object property. Although structures of “ArrivalTime” and “DepartureTime” are the same, their semantics are different. “Time” of “ArrivalTime” is different from “Time” of “DepartureTime”. Concept matching result for these two concepts is *intersection*. They can be used for one another only when there are not any other matches. If a service parameter type is “ArrivalTime” and requested parameter type is “DepartureTime”, “DepartureTime” can be added to suggestion list of “ArrivalTime”. Suggestion list can be displayed to the user in order to provide certainity.

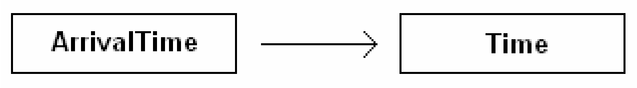


**Figure 3.9** Structure of “ArrivalTime” concept

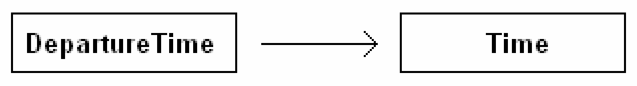


**Figure 3.10** Structure of “DepartureTime” concept

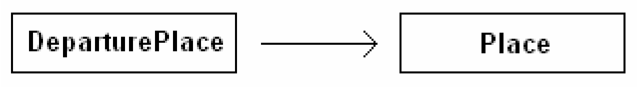


**Figure 3.11** Structure of “DeparturePlace” concept

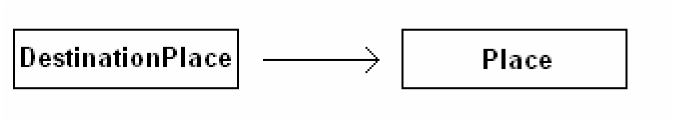


**Figure 3.12** Structure of “DestinationPlace” concept

As shown in the results above, three services are found for requested concepts. These are PegasusService, THYService and AtlasJetService. Also mappings of the requested parameters into the related parameters of the found services are displayed. For each input of a service, there are two collection data structures. One of them collects candidate inputs and the other one collects suggested inputs. If a requested parameter matches with an input completely (*exact* match), then the requested parameter is added to the candidate input list of the input. If there is an *intersection* matching between the requested parameter and the input, then the requested parameter is added to the suggested input list of the input.

The structures of found services are given in Figure 3.13, Figure 3.14 and Figure 3.15 respectively.

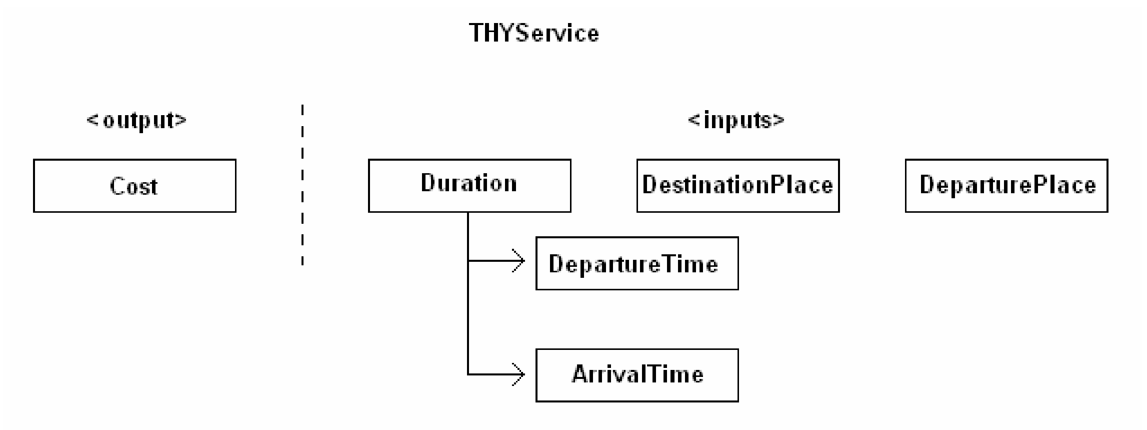


**Figure 3.13** Structure of “THYService” service

As shown in the Figure 3.13, THYService has "Cost" object as output and "Duration", "DestinationPlace" and "DeparturePlace" objects as inputs. "Duration" object has two object parameters which are "DepartureTime" and "ArrivalTime".

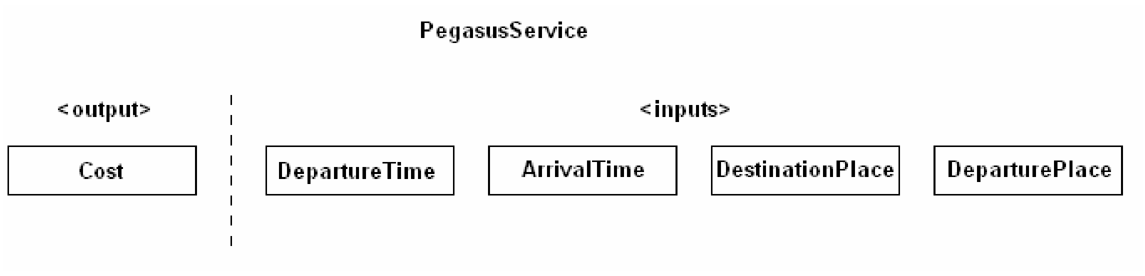


**Figure 3.14** Structure of "PegasusService" service

Figure 3.14 shows structure of PegasusService. PegasusService has "Cost" object as output and "DepartureTime", "ArrivalTime", "DestinationPlace" and "DeparturePlace" as inputs.

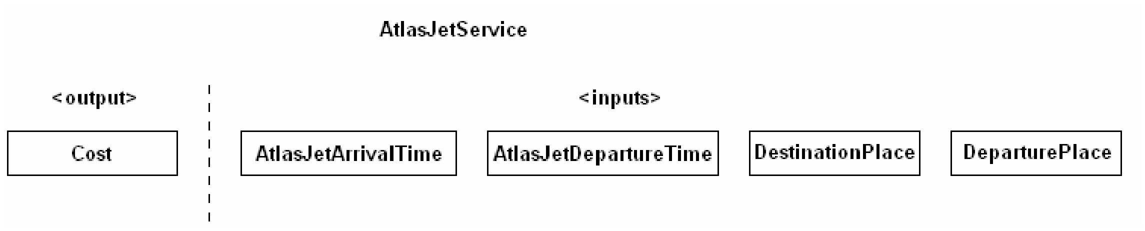


**Figure 3.15** Structure of "AtlasJetService" service

As shown in Figure 3.15, AtlasJetService has “Cost” object as output and “AtlasJetArrivalTime”, “AtlasJetDepartureTime”, “DestinationPlace” and “DeparturePlace” as inputs. Structures of “AtlasJetArrivalTime” and “AtlasJetDepartureTime” are shown in Figure 3.16. There is an inheritance relation between “AtlasJetArrivalTime” and “ArrivalTime” objects. So “AtlasJetArrivalTime” has all attributes (in other words capabilities) of “ArrivalTime”. “AtlasJetDepartureTime” inherits “DepartureTime” and it “is-a” “DepartureTime”.

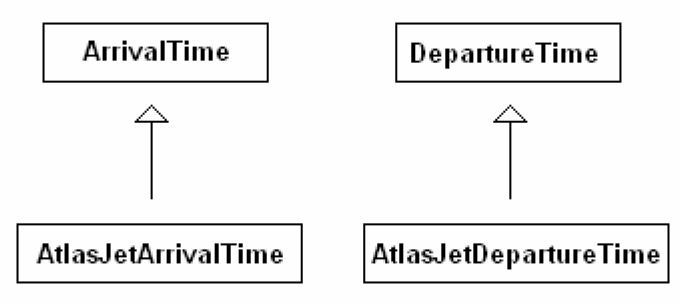


**Figure 3.16** Ontology Model for AtlasJetArrivalTime and AtlasJetDepartureTime concepts

As shown from the structures of concepts, all services cover requested inputs (“DepartureTime”, “ArrivalTime”, “DestinationPlace” and “DeparturePlace”). PegasusService is found as SERVICE_EXACT matching degree because all requested parameters are matched with the PegasusService’s inputs. But THYService is found as SERVICE_INTERSECTION matching degree. Although THYService covers all requested parameters, it can not be accepted as SERVICE_EXACT matching, because its “DepartureTime” and “ArrivalTime” parameters are defined in “Duration” domain which may add different semantics. But because of type of “Duration” ’s “DepartureTime” attribute is same with the requested “DepartureTime” and type of “Duration” ’s “ArrivalTime” attribute is same with the requested "ArrivalTime", these matchings can be shown as suggestions. AtlasJetService is found as SERVICE_EXACT matching degree. AtlasJetService has not "ArrivalTime" and "DepartureTime" input types. Instead, it has "AtlasJetArrivalTime" and "AtlasJetDepartureTime" input types in which there is an inheritance relation between requested parameters. So its match is accepted as SERVICE_EXACT matching degree.

## 3.3 Semantic Web Service Composability Check

Proposed matching algorithm is used for both finding matching services and checking the composability of the services in the composite service model. Services with complementary functionalities are discovered. Syntactic and semantic features of Web services are compared in order to determine whether two services are composable. Composability is the comparison of the syntactic and semantic features of Web services to determine whether two services are interoperable. It refers to the process of checking if Web services to be composed can be actually interact with each other.

Services $S_1$ and $S_2$ such that $S_1 \rightarrow S_2$, (i.e., $S_1$ precedes $S_2$), are composable if $S_2$'s input parameters can be obtained from requested parameters of abstract services and the output parameters of $S_1$ and other preceding services.

Service composion algorithm is shown below. For each service, it is considered whether it can be callable by using the service template parameters. Each service's output is added to service template for the next services.

```
areServicesComposable (CompositeServiceTemplate csTemp,
                        CompositeService cs):hit ratio{
      for each service of cs{
            match degree = areServicesMatch(csTemp,
                                              service of Cs)
            if(match degree >= SERVICE_MATCH_HIT_RATIO)
            matchedServices++
        csTemp.addParameters(cs.getOutputParameters())
    }
    hit ratio = 100 * matchedServices / cs.getServiceSize()
    return hit ratio
}
```

With using Travel ontology, an example can be considered. The example composite service is "TravelingService" (Figure 3.17) and it contains "KAirlinesService", "HiltonHotelService" and "HavasShuttleService" services. This order is same as service orders in composition. The OWL-S files of these services are given in Appendix A.

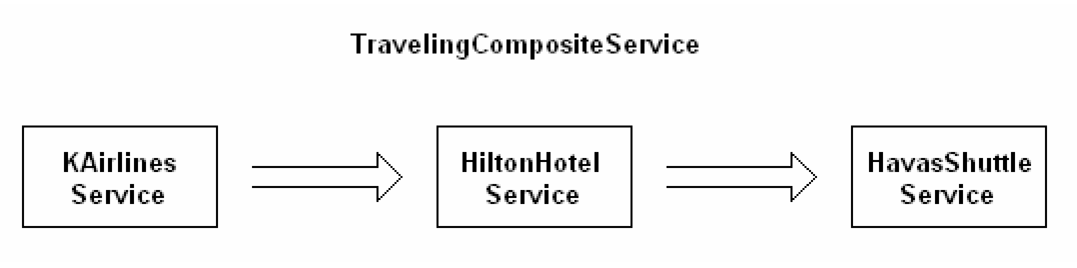


**Figure 3.17** Structure of Traveling Composite Service

Structures of "KAirlinesServices", "HavasShuttleService" and "HiltonHotelService" are shown in Figures 3.18, 3.20 and 3.22 respectively.

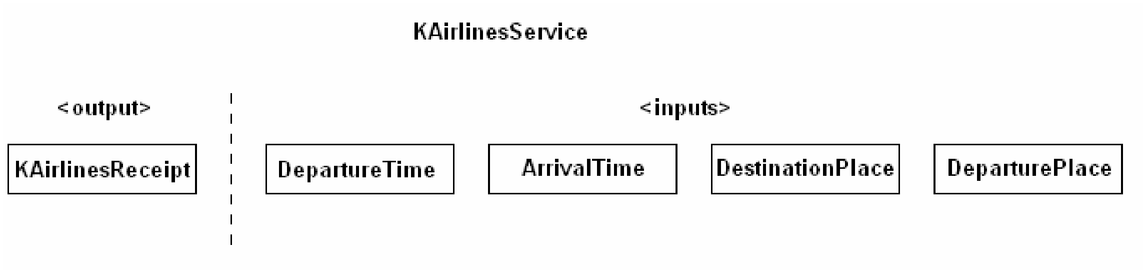


**Figure 3.18** Structure of KAirlinesService

"KAirlinesService" has output parameter which is "KAirlinesReceipt" type. "KAirlineReceipt" has "Cost" and "Airport" object attributes. The structure of "KAirlineReceipt" is shown in Figure 3.19.

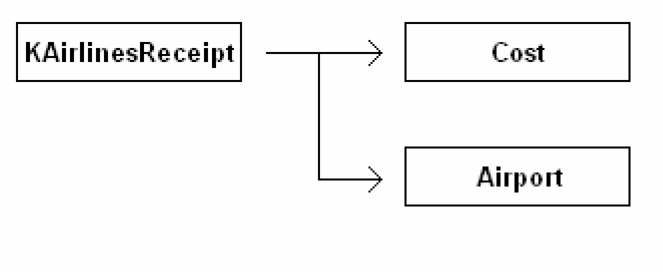


**Figure 3.19** Structure of KAirlinesReceipt concept

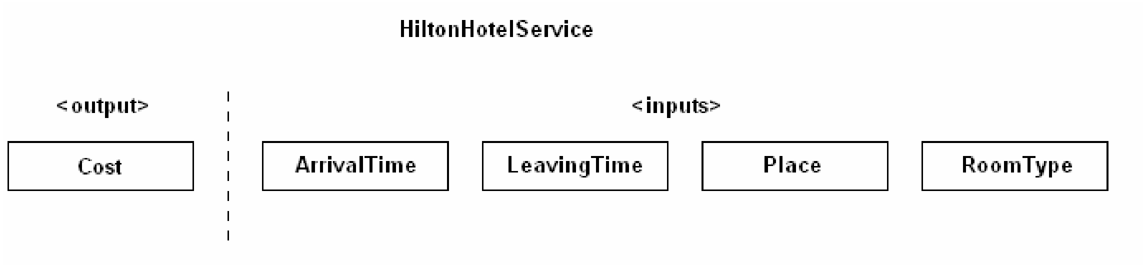


**Figure 3.20** Structure of HiltonHotelService

Structure of “HiltonHotelService” ‘s “LeavingTime” parameter is shown in Figure 3.21.

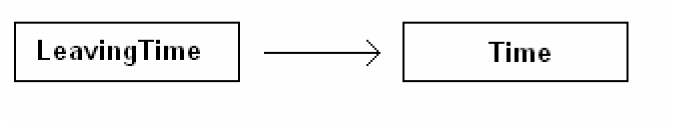


**Figure 3.21** Structure of LeavingTime concept

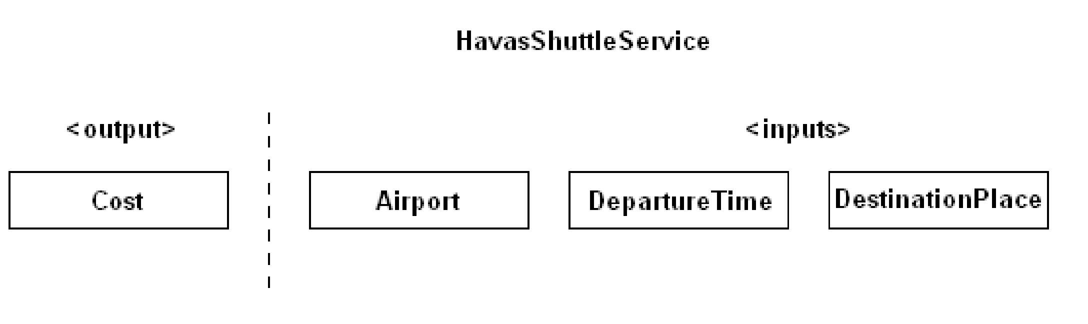


**Figure 3.22** Structure of HavasShuttleService

Requested service parameters and result of composition process is as follows:

*Requested Service Parameters*

***- parameterName****: myArrivalTime* ***parameterType****: ArrivalTime.Time* ***value****: 14/05/2007*

***- parameterName****: myDepartureTime* ***parameterType****: DepartureTime.Time* ***value****: 13/05/2007*

***- parameterName****: myDestinationPlace* ***parameterType****: DestinationPlace.Place* ***value****: Izmir*

*- **parameterName**: myDeparturePlace **parameterType**: DeparturePlace.Place **value**: Ankara*

***Matching Results:***

*1) For TravelingService composition, matching result is 'SERVICE_INTERSECTION (Hit ratio % : 66)'*

*For service: KAirlinesService*

*<u>Matched Service Parameters</u>*

***Service parameterName**: time **parameterType**: DepartureTime.Time*

*<u>Candidate concepts for service parameter :time :::</u>*

***candidate parameterName**: myDepartureTime **parameterType**: DepartureTime.Time **value**: 13/05/2007*

*<u>Suggested concepts for service parameter :time :::</u>*

***suggested parameterName**: myArrivalTime **parameterType**: ArrivalTime.Time **value**: 14/05/2007*

***Service parameterName**: place **parameterType**: DeparturePlace.Place*

*<u>Candidate concepts for service parameter :place :::</u>*

***candidate parameterName**: myDeparturePlace **parameterType**: DeparturePlace.Place **value**: Ankara*

*<u>Suggested concepts for service parameter :place :::</u>*

***suggested parameterName**: myDestinationPlace **parameterType**: DestinationPlace.Place **value**: Izmir*

***Service parameterName**: time **parameterType**: ArrivalTime.Time*

*<u>Candidate concepts for service parameter :time :::</u>*

***candidate parameterName**: myArrivalTime **parameterType**: ArrivalTime.Time **value**: 14/05/2007*

*<u>Suggested concepts for service parameter :time :::</u>*

***suggested parameterName**: myDepartureTime **parameterType**: DepartureTime.Time **value**: 13/05/2007*

***Service parameterName**: place **parameterType**: DestinationPlace.Place*

<u>*Candidate concepts for service parameter :place :::*</u>

***candidate parameterName**: myDestinationPlace **parameterType**: DestinationPlace.Place **value**: Izmir*

<u>*Suggested concepts for service parameter :place :::*</u>

***suggested parameterName**: myDeparturePlace **parameterType**: DeparturePlace.Place **value**: Ankara*

*For service: HiltonHotelService*

<u>*Matched Service Parameters*</u>

***Service parameterName**: roomType **parameterType**: RoomType*

*There is not any candidate concept for service parameter :roomType*

*There is not any suggested concept for service parameter :roomType*

***Service parameterName**: time **parameterType**: LeavingTime.Time*

*There is not any candidate concept for service parameter :time*

<u>*Suggested concepts for service parameter :time :::*</u>

***suggested parameterName**: myArrivalTime **parameterType**: ArrivalTime.Time **value**: 14/05/2007*

***suggested parameterName**: myDepartureTime **parameterType**: DepartureTime.Time **value**: 13/05/2007*

***Service parameterName**: time **parameterType**: ArrivalTime.Time*

<u>*Candidate concepts for service parameter :time :::*</u>

***candidate parameterName**: myArrivalTime **parameterType**: ArrivalTime.Time **value**: 14/05/2007*

<u>*Suggested concepts for service parameter :time :::*</u>

***suggested parameterName**: myDepartureTime **parameterType**: DepartureTime.Time **value**: 13/05/2007*

***Service parameterName****: place* ***parameterType****: Place*

*There is not any candidate concept for service parameter :place*

<u>*Suggested concepts for service parameter :place :::*</u>

***suggested parameterName****: myDestinationPlace* ***parameterType****: DestinationPlace.Place* ***value****: Izmir*

***suggested parameterName****: myDeparturePlace* ***parameterType****: DeparturePlace.Place* ***value****: Ankara*

*For service: HavasShuttleService*

<u>*Matched Service Parameters*</u>

***Service parameterName****: place* ***parameterType****: DestinationPlace.Place*

<u>*Candidate concepts for service parameter :place :::*</u>

***candidate parameterName****: myDestinationPlace* ***parameterType****: DestinationPlace.Place* ***value****: Izmir*

<u>*Suggested concepts for service parameter :place :::*</u>

***suggested parameterName****: myDeparturePlace* ***parameterType****: DeparturePlace.Place* ***value****: Ankara*

***Service parameterName****: time* ***parameterType****: DepartureTime.Time*

<u>*Candidate concepts for service parameter :time :::*</u>

***candidate parameterName****: myDepartureTime* ***parameterType****: DepartureTime.Time* ***value****: 13/05/2007*

<u>*Suggested concepts for service parameter :time :::*</u>

***suggested parameterName****: myArrivalTime* ***parameterType****: ArrivalTime.Time* ***value****: 14/05/2007*

***Service parameterName****: airport* ***parameterType****: Airport*

*There is not any candidate concept for service parameter :airport*

<u>*Suggested concepts for service parameter :airport :::*</u>

***suggested parameterName**: airportName **parameterType**: KAirlinesReceipt.Airport **value**: null*

As it can be seen from the result, requested parameters do not contain “Airport” attribute which is required for “HavasShuttleService”. However “KAirlinesService” has “Airport” attribute in its output. Therefore “KAirlinesService” and “HavasShuttleService” can be placed in composition because all attibutes of these two services can be covered by all requested parameters and “KAirlinesService”’s output attributes. But “HiltonHotelService” has “RoomType” attribute which can not be covered. So the composition “TravelingService”’ hit ratio is 66%. Also “LeavingTime” attribute of “HiltonHotelService” can not be covered. But “ArrivalTime” and “DestinationTime” parameters which can cover “LeavingTime” attribute are suggested to the user for selecting.

After composition creation operation is finished, a new OWL-S file can be created for the composite service by using OWL-S files of each service in the composition.

## 3.4 Semantic Mapping

Semantic Mapping is an operation in which semantic similarities are defined between different semantic concepts. With the help of human intervention new semantic similarities can be created. Then these similarities can be used in matching process and similarity algorithms can be run correctly for different concepts that have the same meaning.

One domain can be defined differently in various ontology documents. This is mostly due to the fact that the organizations may have their own ontology models according to their business structures. This causes many different ontology models for each domain. It is not easy to change ontology models of organizations with a common ontology model because each organization can have many legacy systems. Therefore, it may not be possible to meet under a common ontology. In order to provide interoperability and standardization, mapping must be defined between these different ontologies that model similar ontologies. Ontology mapping is a solution to the semantic heterogeneity problem faced by information management systems. Ontology mapping finds correspondences between semantically related entities of the input ontologies. Thus, mapping ontologies enables the knowledge and data expressed in the matched ontologies to interoperate.

Ontology mapping is a key interoperability enabler for the semantic Web, since it takes the ontologies as input and determines as output correspondences between the semantically related entities of those ontologies. Two ontology model for the same domain can be mapped each other. Mapping can be defined for each concept of one of the model to the other. This case is shown in Figure 3.23. Semantic Mapper knows two ontology documents “A.owl” and “B.owl”. When a concept is used from “A.owl”, semantic mapper knows how this concept can be represented in “B.owl”. By this way an application that uses one ontology model can communicate with other application that uses the other ontology model.

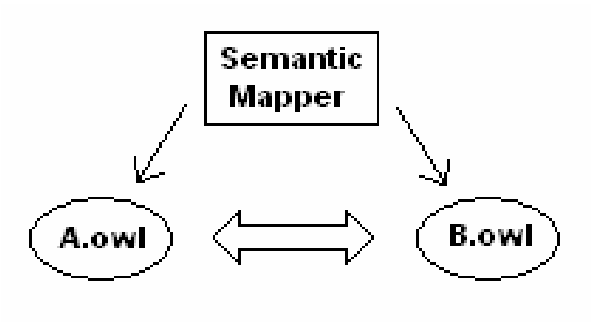


**Figure 3.23** Semantic Mapping

As Semantic Mapping can be defined between ontology documents, it also can be defined between some ontology concepts. In this work, semantic mapping between ontology concepts is utilized in order to enhance the proposed semantic matching algorithm.

Mapping operation requires human intervention. In this study, user can define mappings whenever he/she wants. When a mapping is defined by the user, it is added to the mapping list. In semantic matching process, mappings in this mapping list are considered because mapping between two concepts shows a similarity between these concepts. For example, if a mapping is defined from concept X to concept Y; this shows concept X can cover all attributes of concept Y; therefore, concept X can be used in the place of concept Y when necessary. In semantic matching process, mappings in the given mapping list are considered since the mappings indicate the similarity and replaceability of the concepts.

Transformation (bridge) is important in mapping process. It shows the correspondence between the attributes of concepts. In this thesis, basic transformations are implemented. Let A be an attribute object of concept to which mapping is defined and B be an attribute object of concept from which mapping is defined. In this work, user can define the following three types of transformations between attributes of concepts:

1. A = B
2. A = B OPERATION constant, where OPERATION = {Addition, Subtraction, Multiplication and Division} and constant is a numeric value
3. A = constant, where constant is an alphanumeric value.

A is a primitive attribute object of concept to which mapping is defined. B is a primitive attribute object of concept from which mapping is defined. Constant can be any value. But in the second method it must be numeric. Addition, Subtraction, Multiplication and Division are defined operations.

Figure 3.24 shows an example mapping transformation. “Price” and “Fiyat” (which means price in Turkish) are similar concepts. They have the same meaning however their currencies are different. This transformation is defined by using mapping screen, which is “valueInTl = valueIn$ * foreign currency exchange rate”.

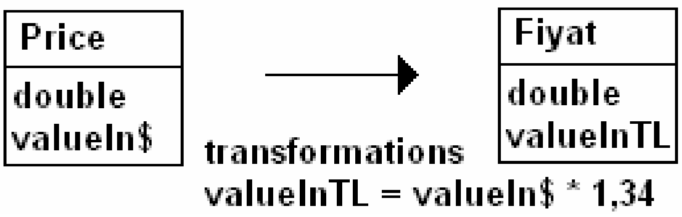
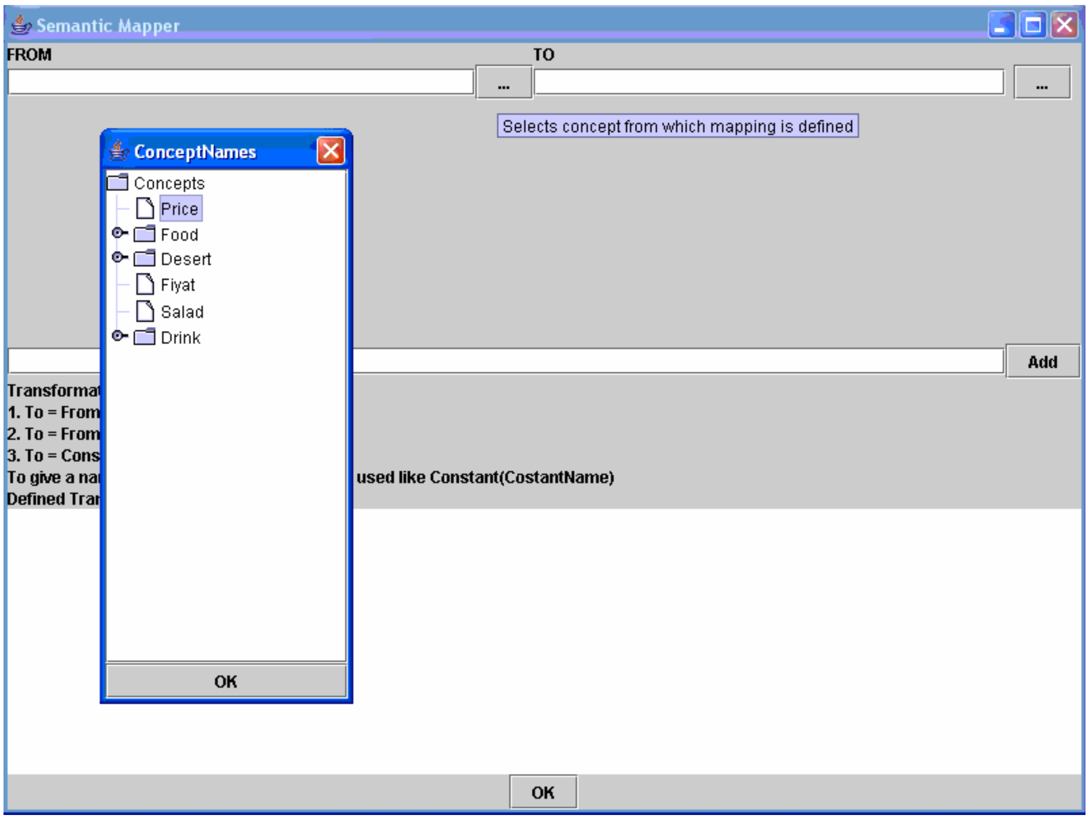


**Figure 3.24** Mapping from Price to Fiyat

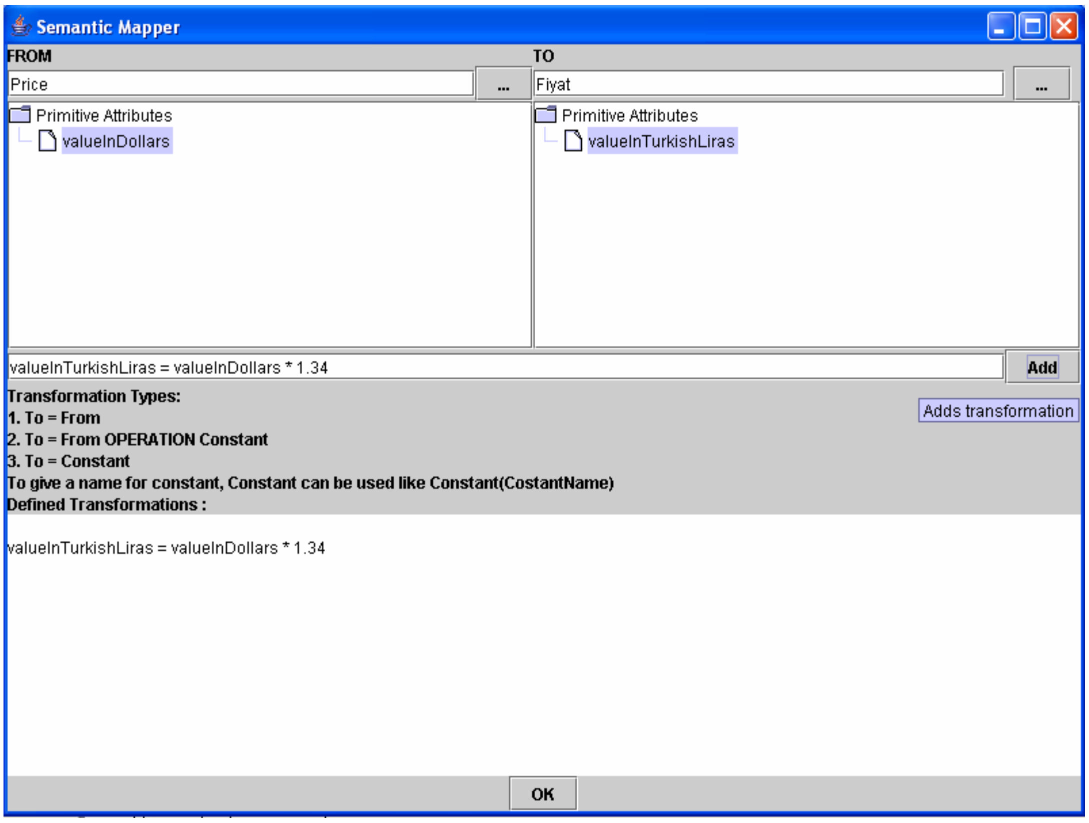
To define this mapping, user opens Semantic Mapper frame. As it is shown in Figure 3.25, firstly, user selects concepts for which mapping are defined.



Transformat
1. To = From
2. To = From
3. To = Cons
To give a na                used like Constant(CostantName)
Defined Tra

**Figure 3.25** A screen of Concept Selection

Then the user selects attributes of concepts and defines a transformation. With using “Add” button, transformation is checked and added to transformation list. This is shown in Figure 3.26.



Price

Fiyat

Primitive Attributes
valueInDollars

Primitive Attributes
valueInTurkishLiras

valueInTurkishLiras = valueInDollars * 1.34

Transformation Types:
1. To = From
2. To = From OPERATION Constant
3. To = Constant
To give a name for constant, Constant can be used like Constant(CostantName)
Defined Transformations :

valueInTurkishLiras = valueInDollars * 1.34

**Figure 3.26** A screen of Transformation addition

Each defined mapping is written to an XML document. The content of this document for the mapping described above is as follows:

```
<?xml version="1.0" encoding="ISO-8859-9"?>

<Mappings>
      <MappingDefinition>
            <MappingFrom>
                  <OntologyPath>
                        file:///E:\sampleOwlDocuments\Price.owl
                  </OntologyPath>
                  <TypeName>Price</TypeName>
            </MappingFrom>

            <MappingTo>
                  <OntologyPath>
                        file:///E:\sampleOwlDocuments\Fiyat.owl
                  </OntologyPath>
                  <TypeName>Fiyat</TypeName>
            </MappingTo>

            <Transformations>
                  <Transformation>
                        <OntologyObjectTo>
                              <Name>valueInTurkishLiras</Name>
                              <Value></Value>
                        </OntologyObjectTo>

                        <OntologyObjectFrom>
                              <Name>valueInDollars</Name>
                              <Value></Value>
                        </OntologyObjectFrom>

                        <Operation>4</Operation>

                        <Constant>
                              <Name>dollarCurrency</Name>
                              <Value>1.35</Value>
                        </Constant>

                        <Type>2</Type>
                  </Transformation>
            </Transformations>
      </MappingDefinition>
</Mappings>
```

Defined mappings are stored in this XML file. When mapping application is loaded into memory, it reads the mapping file and when a new mapping is defined, it is written to the mapping file.

## 3.5 Semantic Domain Model

Ontology documents which are used for definition of concepts and relations between concepts, is not exactly adequate for semantic information inference. More relations should be extracted from different ontology documents. Also it is difficult to define a complete ontology of a domain so different ontologies can have complementary concepts. These concepts must be combined. In addition, according to domain or action to be done, new relations can arise between concepts in which there is not any relation between each other. In order to model complementary concepts and the relation between the concepts and relevant actions, and to infer more information about these relationships, an action-based semantic domain model is developed. The model is action based because actions define works to be done. By considering actions, more-relevant relations can be discovered.

Semantic domain model composes semantic relations between concepts and actions. In ontology documents concepts and inheritance and attribute relations are defined. Actions must be defined or modeled as well. An action is an activity that is accomplished with objects at a certain place and time. Concepts are modeled as objects that are associated with actions. By using this model, actions, concepts that can be used with actions, domains which describe businesses of actions, places and time in which actions take place are modeled. In addition, relations between actions can be determined as well. It is possible to suggest new, related and complementary actions on the basis of the requested action. Orders of actions can be inferred and ordering suggestions can be presented to the user. Actions are related with concepts according to a determined domain because relation between an action and a concept is meaningful only in a determined domain. To provide this kind of relation DomainConcept data structure is used. The structure of the model is shown in Figure 3.27.

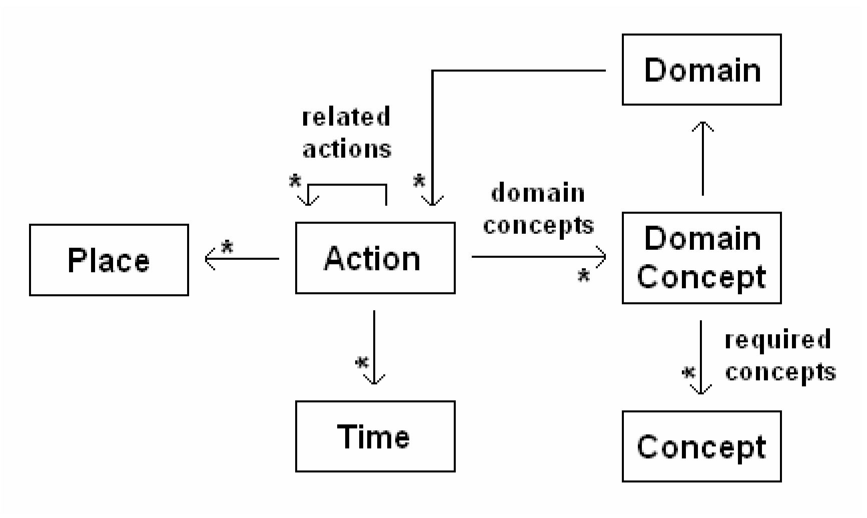


**Figure 3.27** Semantic Domain Model

By using this model, information extraction and inference becomes more easily and semantically. Some of the information that can be inferred by using this model is as follows:

- Required concepts of an action can be inferred
- An action can be inferred if domain and required concepts are known
- Related actions of an action can be inferred and suggested to the user as a guidance service
- Actions that are relevant to a domain can be inferred
- The place of an action can be inferred.
- Both domain guidance and action guidance can be provided

By using this model, different concepts in different ontologies can be related with actions. For example, there is not any relation between “plane” and “hotel” concepts. However in traveling domain, hotel booking and plane reservation actions are modeled as related.

As it is stated in [2]; if user request is “order food”, the information that a “restaurant is needed” should be inferred. Proposed model can provide such kind of information since “order food” is modeled as an action and “restaurant” is a related concept of “order food” action. In addition, complementary concepts and actions can be defined. By this way, for example, if user wants to buy a computer, buying a printer can be suggested.

Some inference examples are shown below.

- *BATH is done in BATHROOM with using WATER and SOAP*

These concepts can be defined in “Bath” domain but this information can be modeled in this proposed method. “Bath” is modeled as action, “Bathroom” is modeled as place, “WATER” and “SOAP” are modeled as required objects.

- *FOOD or MEAL is EATEN in RESTAURANT or KITCHEN with KNIFE, SPOON, FORK*

In this example, two same actions which are “Food” and “Meal” are considered. Place information can be “Restaurant” or “Kitchen”. “KNIFE”, “SPOON” and “FORK” are required objects.

- *ELECTRIC SOCKETS are INSTALLED*
- *WALL or WINDOW or DOOR or ROOM or HOUSE or  ROOF are DYED*
- *WINDOW or DOOR or RADIATOR are SET*
- *PARQUET is EMERIED*
- *ROOF is PLASTERED*
- *LAMINANT is PLASTERED*

Some information samples for “House Construction” domain that can be modeled by the proposed method are shown above.

Defined relations are stored in an XML file. In order to facilitate loading and saving operations, its structure is same as the model that is shown in Figure 3.27.

This proposed method can be considered as an interpretation of sentences. Structures of sentences can be considered as basis of our ontology model. As we know that sentences are composition of verbs, adjectives, adverbs and nouns. The critic one is verb. Verb is the part of speech that expresses existence, action, or occurrence in most languages. It tells operation. To perform a verb, some objects required. Also these actions are done in places.

With these capabilities, the proposed semantic domain model can be used in service composition process. It is utilized in guidance the modeling and service discovery steps. Web services perform an action, therefore they can be considered as actions in semantic domain model. Web service parameters are the required concepts of the action. When user wants to request a service template that will take part in the composition, he/she determines an action. The selected action's required concepts are listed to the user. Among this list, user selects a set of concepts that meets his/her needs. In service discovery step, matching is done according to the selected concepts to find candidate Web services. User can also use semantic domain model as a guiding system. According to domain which is selected by the user, actions are listed. User can select the related actions (services) from this list. Once actions are selected, user may ask for a list of related concepts as well. Hence, the user can model the composite service without having detailed information on the domain.

## 3.6 Implementation Issues

In this section, some of the main implementation details are illustrated. Models which are created for this thesis are shown. In addition, class diagrams that are created for implementations are shown.

### 3.6.1 Created Packages

In this section, created packages for implementation are illustrated. Package dependencies are shown in Figure 3.28.

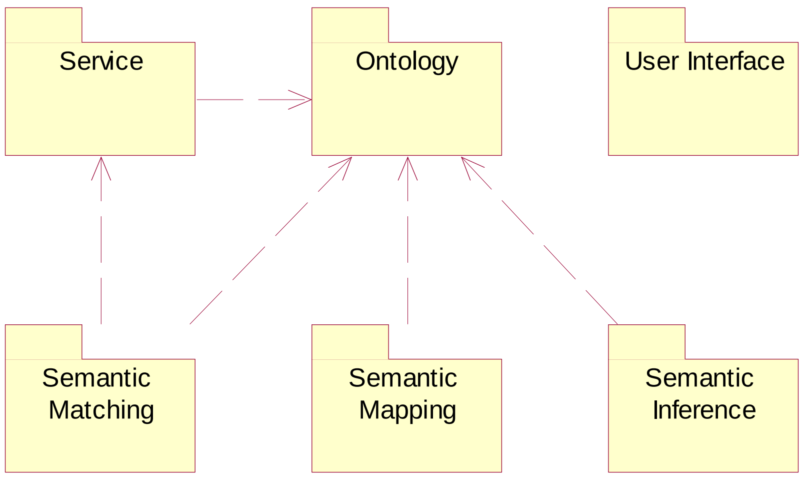


**Figure 3.28** Created Packages

### 3.6.1.1 Ontology package

This package is responsible for Ontology related operations. Definition and access of ontology concepts is implemented in this package. Class diagram of Ontology package is shown in Figure 3.29.

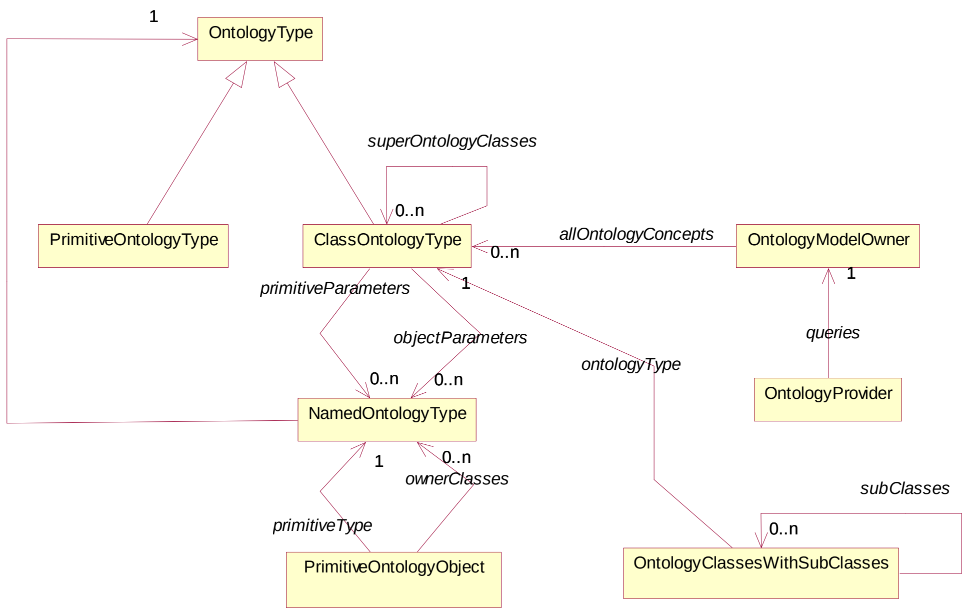


**Figure 3.29** Class Diagram of Ontology Package

OntologyType is data class for ontology concepts. It has two sub classes which are PrimitiveOntologyType and ClassOntologyType. NamedOntologyType is used for parameters. It includes OntologyType and adds name property to this data structure. PrimitiveOntologyObject is used for instances of PrimitiveOntologyType. OntologyClassWithSubClasses is data class to form ontology concepts in tree order. In ClassOntologyType, subclasses are not included naturally.

OntologyModelOwner contains all read ontology concepts, which are cached in the memory. Ontology documents can be loaded before all operations or ontology concepts can be lazy loaded when they are required. While an ontology concept is loaded from an ontology document, as the first operation, super concepts and concept attributes of that concept are loaded recursively by this method. Then ontology concept's primitive parameters are loaded. Loaded ontology concepts are put into ModelOwner as a ClassOntologyType object. ModelOwner has all of the read concepts. This prevents loading of the same concepts more than once. By this way efficiency is increased. In ModelOwner concept cache; concepts exist according to their names and ontology file paths which show ontology files in which concepts are defined.

When a concept is needed, ModelOwner is queried. If the concept is found, it is used, otherwise, the concept is loaded and then it is put to the ModelOwner. Super concept or concept attribute of an ontology concept can be in different ontology files. In this case, while loading operation, other ontology files are found and needed concepts are loaded.

OntologyProvider which is one of the most important classes is used for all ontological operations such as loading semantic concepts and semantic Web services. It is also used to query model owner according to semantic requirements.

#### 3.6.1.2 Service package

Service Package contains classes that are related with Web Services. The class diagram of this package is shown in Figure 3.30.

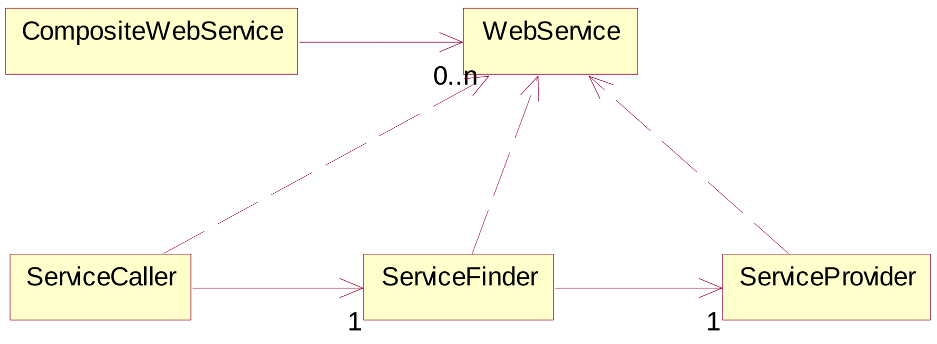


**Figure 3.30** Class Diagram of Service Package

WebService is a class which represents Web services. CompositeWebService is class which represents Web service compositions. ServiceFinder is used search Web services from ServiceProvider. ServiceProvider queries Web service registries like UDDI to find web services. ServiceCaller is used to call a service.

#### 3.6.1.3 Semantic Matching package

Semantic matching classes are in this package. The class diagram of this package is shown in Figure 3.31. SemanticMatcher is used when a matching is requested. ConceptMatcher is implementation of the concept matching algorithm. ServiceMatcher is implementation of the service matching algorithm. ParameterHitRatio is the data class for similarity ratio of intersection matching result.

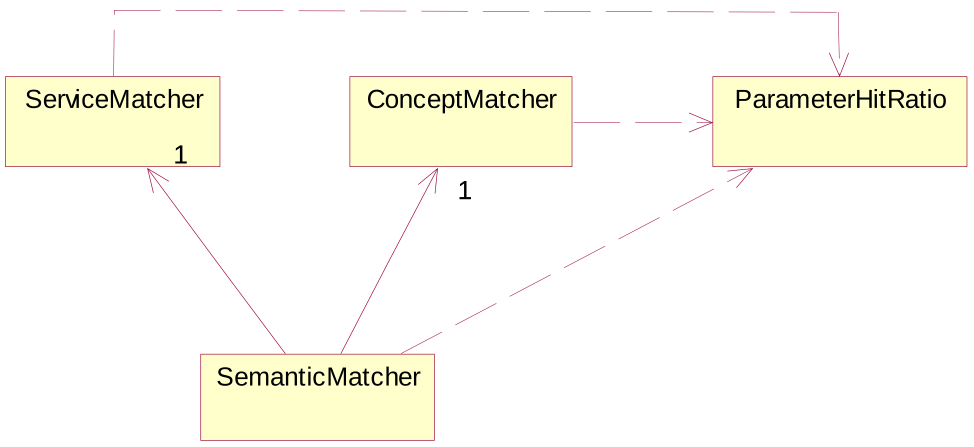


**Figure 3.31** Class Diagram of Semantic Matching Package

#### 3.6.1.4 Semantic Mapping package

This package is used for the semantic mapping operations. The class diagram of this package is shown in Figure 3.32. IMappingConstants has the mapping constants which are used when writing and reading defined mappings. Defined mappings are saved in a mapping file which is in XML format. MappingReader is used for reading mappings from the mapping file and MappingWriter is used for writing mappings to the mapping file. MappingDefinition is a data class for mappings. One mapping is defined by a MappingDefinition object. A mapping is defined between two ClassOntologyType which is in Ontology package. For parameters of these objects transformations are defined. Transformation class is data class to define transformation. A transformation is defined between two PrimitiveOntologyObject which is in Ontology package. Constant is a data class to define constants in transformations. SemanticMapper is a class in which all defined mappings exist. It provides other modules to access predefined mappings or to define new mappings.

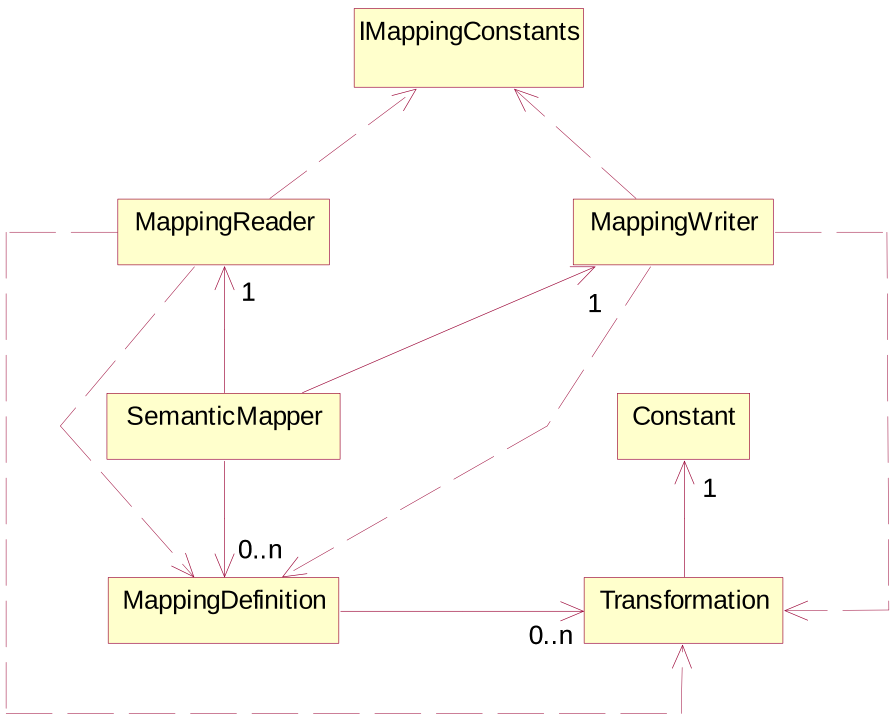


**Figure 3.32** Class Diagram of Semantic Mapping Package

### 3.6.1.5 Semantic Inference package

This package is created for implementation of the semantic domain model. The class diagram of this package is shown in Figure 3.33. Action, Concept, Domain and DomainConcept classes form semantic domain model. ModelOwner class supplies semantic domain model. When an action, a concept or a domain class is required, it is searched in ModelOwner. InferenceEngine is used for semantic querying like finding. Ontolog is used for adding to ModelOwner and querying ModelOwner.

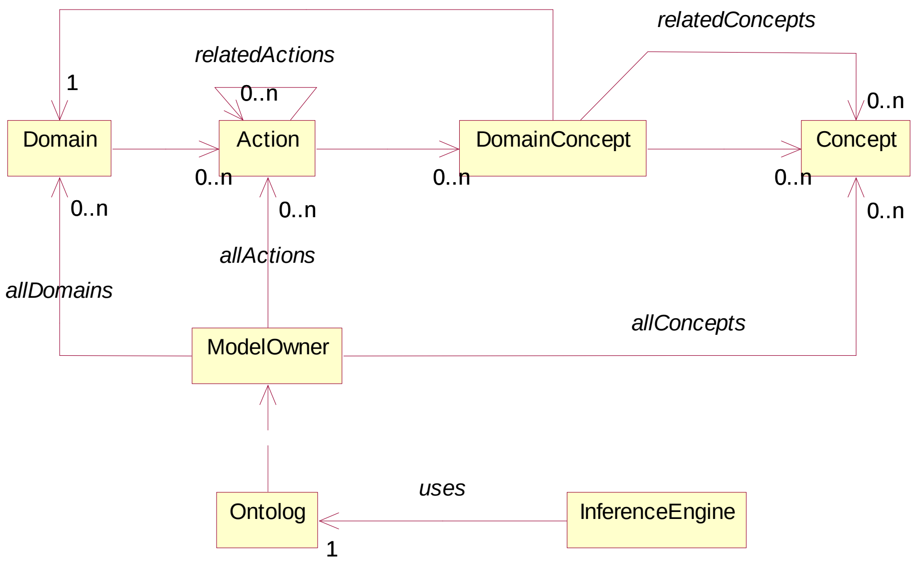


**Figure 3.33** Class Diagram of Semantic Inference Package

#### 3.6.1.6 User Interface package

This package contains user interface components such as ConceptAttributesPanel and ConceptNamesPanel. ConceptAttributesPanel is used for displaying all attributes of a concept. ConceptNamesPanel is used for displaying all concept names in the system.

#### 3.6.1.7 Other Important Classes

SemanticManager class is used as Facade class in front of applications which they use these proposed methods to add semantic methods. CWSF is such an application that uses this class for all operations related with semantic.

### 3.6.2 Used Technologies

This work is implemented by using Java [6] programming language. The most important library that is used in this study is OWL-S API [16]. OWL-S API is a Java based API which provides reading, writing and executing Semantic Web Servises described in OWL-S. Only deficiency of OWL-S API is the definition of complex XSLT transformations. Each object attribute needs XSLT transformations to map Web service attributes. If the object has object attributes rather than primitive attributes, transformations of the object causes errors in OWL-S API.

# CHAPTER 4

# EVALUATION

In this section, evaluation of the proposed methods is presented. Firstly, semantic matching algorithm is evaluated. Then, evaluation of semantic composition method is given. Three ontology documents for three different domains are used for these evaluations. These ontology documents are:

1) Meal.owl : In this ontology; foods and restaurant system are defined,

2) House.owl : In this ontology; House and its basic structures are defined

3) Traveling.owl : In this ontology, traveling business is defined.

In addition, for service matching evaluation, several service definition files are used. The contents of these ontology documents are given in Appendix A. By using these ontology documents, sample data sets are created. The similarities between concepts are determined explicitly. Then algorithms are run to evaluate the success rates to find similairities.

Recall and precision are two basic quality parameters to evaluate the matching algorithms. Recall is the ratio of the number of the relevant items found to the total number of relevant items. Precision is the ratio of the number of the relevant items found to the total number of items retrieved. For example if there are 15 total relevant items and an algorithm finds 8 items in which 5 of them is relevant; then we can say that recall of the algorithm is: 5/15 (33.3%) and precision of algorithm is: 5/8 (67.5%). High precision and high recall are desired properties for a matching algorithm. Precision value 100% is obtained if algorithm finds no irrelevant items. Recall value 100% is obtained if algorithm finds all relevant items.

In this section, the proposed algorithms are evaluated under recall and precision.

## 4.1 Semantic Matching Evaluation

In this part semantic matching algorithms are evaluated. Algorithms are evaluated with respect to how good they can find a requested item (concept or service) by taking similarities into consideration. Semantic matching algorithm is evaluated in two parts. In the first part evaluation of the semantic concept matching algorithm is explained. In the second part evaluation of the semantic Web service matching algorithm is explained.

### 4.1.1 Concept Matching Evaluation

In this evaluation, the aim is to find similarities between some relevant and some irrelevant concepts in our ontology documents. It is expected from the concept matching algorithm to find similarities between relevant concepts and to find dissimilarities between irrelevant concepts.

The data set contains a set of similarity candidates where actual relevant similarity count is 30. Total sample data count is 40 but only 30 of them are really similar. In Table I, the results of three matching methods according to recall and precision quality parameters are given.

Actual relevant concept count: 30

**Table I:** Concept Matching Results

| | Keyword based | Inheritance based | Proposed Method |
|---|---|---|---|
| Relevant concepts | 5 relevant found, 25 missed | 20 relevant found, 10 missed | 27 relevant found, 3 irrelevant |
| Recall | 16.7% | 66.7 % | 90 % |
| Precision | 100 % | 100 % | 90 % |

Keyword based matching is not a semantic process. Therefore, it provides only syntactic checking. It gives no result or gives many irrelevant results. This method's precision is 100% on the sample data set. However, if there were some homonyms in sample data set, precision of the keyword based search would decrease since it would have found irrelevant concepts as well. The recall of this method is low since it only does syntactic checking. The proposed approach has high recall than the inheritance based matching in [18]. The proposed matching algorithm finds 30 relevant similarities among 40 candidate similarities. But in fact 3 of them are irrelevant. Precision of inheritance based matching can be considered as higher than proposed approach however in this experiment 100% hit ratios are accepted. By adjusting hit ratio property, precision and recall values of the proposed algorithm can be increased. Precision and recall values of the proposed algorithm increased to 100% if intersections which have more than 80% hit ratios are accepted and intersections which have less than 80% hit ratios are considered as suggestions.

### 4.1.2 Service Matching Evaluation

In this evaluation, the aim is to find candidate services similar to the requested Web services. Service matching degree is used as a quality measurement to indicate the similarity of the requested service template and discovered candidate service.

Results of the experiments are shown in Table II. Proposed method is compared with keyword based and inheritance based matching based techniques.

Actual relevant service count: 8

**Table II:** Service Matching Results

| | Keyword matching based Service Matching | Inheritance matching based Service Matching | Proposed Service Matching |
|---|---|---|---|
| Relevant services | 2 relevant found, 6 missed | 3 relevant found, 5 missed | 5 relevant found, 3 missed |
| Recall | 25 % | 37.5 % | 62.5 % |
| Precision | 100 % | 100 % | 100 % |

Proposed approach has higher precision and recall since it checks attribute similarity additionally. Also in proposed service matching, as explained in 3.2.2 for each parameter there are candidate list and suggested list. With the help of human intervention, the suggested list can be considered in matching process. The recall of our approach can be increased to 100% if suggested lists are accepted.

## 4.2 Web Service Composition Evaluation

In Figure 4.1, service composition execution time is displayed with respect to the number of requested parameters. Composite service is “TravelingService” which is described in section 3.3. As seen from the figure, most of the processing time is used for the matching. Service composition operation includes service matching and service matching operation includes concept matching task. Concept matching takes half of the total processing time. Service matching operation time includes concept matching operation time plus time passed for finding attributes of service and finding hit ratio of service matching. Composition operation time includes service matching time plus time passed for finding composition hit ratio time.

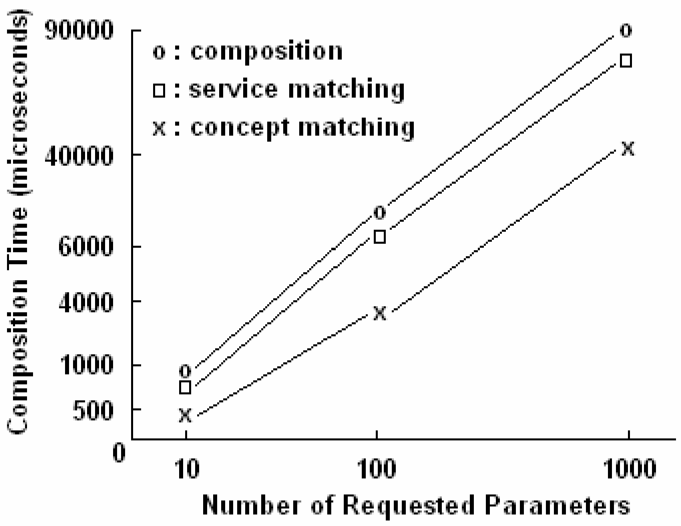


**Figure 4.1** Time required for service composition processes

As shown in the figure, composition time increases linearly with the increase in the number of requested parameters.

The cost of loading ontology concepts is high. In this experiment there are 89 ontology concepts in the system. Although these concepts are in the same computer, loading of these concepts takes approximately 500 milliseconds.

The experiments are conducted on a personal computer which has Intel P4 3GHz processor and 1 GB of RAM.

# CHAPTER 5

# FUTURE WORK AND CONCLUSION

In this work, new semantic based techniques are proposed in order to facilitate the Web service composition process. These approaches include new semantic matching methods for finding both concept similarity and service similarity. Proposed matching algorithms can find complex similarities. A simple semantic discovery method for semantic Web services is added. In addition, to improve semantic inference, new semantic domain model which can capture the relationships among the concepts and between the concepts and the actions (services) is proposed. This model can be also used for guiding user in definition phase of Web service composition process. A basic mapping tool is implemented and defined mappings are used in the matching process.

These new approaches are used in modeling the composite Web service, service discovery and interoperability checking. With the help of these methods, Web service composition process becomes easier, less error prone and more automated. Computers can understand these structures so human intervention decreases. The experiments show that the proposed matching algorithms increase the quality of matching.

As a future work, improvement of proposed semantic domain model can be considered. This model is just in its early phase of development. It is open to new extensions which will increase the amount and quality of semantic inferences. Structure of the model can be extended and can cover other attributes like "place", "time" etc. It is aimed that computers can make every inference and reach to a decision, so they can implement operations without human intervention.

In the future, this model can be extended with natural language processing modules so that the user describes the composite model in natural language and the system analyzes the structure of sentences in order to find verbs, objects, time and place and considering these as actions, required objects so on.

Another direction for future work is addition of new capabilities for semantic service registry. New studies can be implemented for semantic service registry which is not UDDI based.

In addition, semantic mapping is a subject which has many challenging problems. In this work, we only considered a simple mapping mechanism to increase the quality of semantic matching. This module can be extended with more complex mapping transformations and with automatic or semi-automatic techniques for extraction of transformation requiring less human intervention.

# REFERENCES

[1] B. Benatallah, M. Dumas, M. C. Fauvet and F.A. Rabhi, 2002. Towards Patterns of Web Services Composition. In Gorlatch, S. and F. Rabhi (Eds.), Patterns and Skeletons for Parallel and Distributed Computing. Springer Verlag (UK).

[2] T. Broens, Context-aware, Ontology based, Semantic Service Discovery (COSS), Thesis for a Master of Science degree in Telematics from the University of Twente, Enschede, The Netherlands, July 2004.

[3] Choco constraint programming system, http://choco.sourceforge.net/, August 2007.

[4] C. Fellbaum, A Semantic Network of English: The Mother of All WordNETs, Computers and Humanities, 209-220, 1998.

[5] D. Fensel, and C. Bussler, 2002. Semantic web enabled web services. In Proceedings of International Semantic Web Conference (ISWC'2002), volume 2342.

[6] Java, http://java.sun.com/, August 2007.

[7] E. Karakoc, K. Kardas, and P. Senkul, A Workflow-based Web Service Composition System, 2006 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2006 Workshops)(WI-IATW'06).

[8] E. Karakoc, Web Service Composition Under Resource Allocation Constraints, Thesis for a Master of Science degree in Computer Engineering from the University of METU, Turkey, April 2007

[9] L. Li and I. Horrocks, A software framework for matchmaking based on semantic web technology. In Proceedings of the 12th International Conference on the World Wide Web, Budapest, Hungary, May 2003.

[10] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. R. Payne, M. Sabou, M. Solanki, N. Srinivasan, K. Sycara (SRI, CMU, Univ. Toronto), Bringing Semantics to Web Services: The OWL-S Approach, First International Workshop on Semantic Web Services

and Web Process Composition (SWSWPC 2004) 6-9, 2004, San Diego, California, USA.

[11] S. McIlraith, T. C. Son, and H. Zeng. Semantic Web services. IEEE Intelligent Systems, 16(2):46–53, March/April 2001.

[12] B. Medjahed, A. Bouguettaya, A.K. Elmagarmid, Composing Web services on the Semantic Web, VLDB J. 12(4): 333-351, 2003.

[13] N. Milanovic and M. Malek, Current solutions for Web service composition, IEEE Internet Computing, 2004

[14] OWL, Web Ontology Language, http://www.w3.org/2004/OWL , August 2007.

[15] OWL-S, Semantic Markup for web services, OWL white paper, http://www.daml.org/services/owls/1.0/owl-s.pdf , August 2007.

[16] OWL-S API, Maryland Information and Network Dynamics Lab Semantic Web Agents Project, http://www.mindswap.org/2004/owl-s/api , August 2007.

[17] M. Paolucci, T. Kawamura, T. R. Payne, K. Sycara, Importing the Semantic Web in UDDI, WES 2002: 225-236.

[18] M. Paolucci, T. Kawamura, T. R. Payne, K. Sycara, Semantic Matching of Web Services Capabilities, International Semantic Web Conference, 333-347, 2002.

[19] E. Sirin, J. Hendler, B. Parsia, Semi Automatic Composition of Web Services using Semantic Descriptions, in: Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003, 2002.

[20] SOAP, Simple Object Access Protocol; http://www.w3.org/TR/SOAP, August 2007.

[21] UDDI, Universal Description, Discovery and Integration of Web Services; http://www.uddi.org, August 2007.

[22] K. Verma, K Gomadam, A. P. Sheth, J. A. Miller, Z. Wu , The METEOR-S Approach for Configuring and Executing Dynamic Web Processes, LSDIS Lab Technical Report, University of Georgia, Athens, Georgia, USA, 2004.

[23] Web Service, http://en.wikipedia.org/wiki/Web_service, August 2007

[24] WSDL, Web Service Description Language; http://www.w3.org/TR/wsdl, August 2007.

[25] XML, eXtensible Markup Language, www.w3.org/XML/, August 2007.

# APPENDIX A

# ONTOLOGY DOCUMENTS

In this section, ontology documents which are created for thesis study are listed.

## A.1 House.owl

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/unnamed.owl#"
    xml:base="http://www.owl-ontologies.com/House.owl">

  <owl:Class rdf:ID="House"/>
  <owl:Class rdf:ID="Wall"/>
  <owl:Class rdf:ID="Roof"/>
  <owl:Class rdf:ID="HFloor"/>
  <owl:Class rdf:ID="Tavan"/>
  <owl:Class rdf:ID="Window"/>
  <owl:Class rdf:ID="Parquet"/>

  <owl:Class rdf:about="#Parquet">
    <rdfs:subClassOf rdf:resource="#HFloor"/>
  </owl:Class>
  <owl:Class rdf:ID="Laminant">
    <rdfs:subClassOf rdf:resource="#Parquet"/>
  </owl:Class>
  <owl:Class rdf:ID="Wooden">
    <rdfs:subClassOf rdf:resource="#Parquet"/>
  </owl:Class>
  <owl:Class rdf:ID="Bathroom"/>
  <owl:Class rdf:ID="Faience">
    <rdfs:subClassOf rdf:resource="#HFloor"/>
  </owl:Class>
```

```
<owl:Class rdf:ID="Radiator"/>
<owl:Class rdf:ID="Terrace"/>
<owl:Class rdf:ID="Door"/>
<owl:Class rdf:ID="Room"/>
<owl:ObjectProperty rdf:ID="tDuvar">
  <rdfs:domain rdf:resource="#Terrace"/>
  <rdfs:range rdf:resource="#Wall"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="bDuvar">
  <rdfs:domain rdf:resource="#Bathroom"/>
  <rdfs:range rdf:resource="#Wall"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="pencere">
  <rdfs:range rdf:resource="#Window"/>
  <rdfs:domain rdf:resource="#Wall"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="kalorifer">
  <rdfs:domain rdf:resource="#Wall"/>
  <rdfs:range rdf:resource="#Radiator"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="oda">
  <rdfs:domain rdf:resource="#House"/>
  <rdfs:range rdf:resource="#Room"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="duvar">
  <rdfs:domain rdf:resource="#Room"/>
  <rdfs:range rdf:resource="#Wall"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="bTaban">
  <rdfs:range rdf:resource="#HFloor"/>
  <rdfs:domain rdf:resource="#Bathroom"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="cati">
  <rdfs:range rdf:resource="#Roof"/>
  <rdfs:domain rdf:resource="#House"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="tTaban">
  <rdfs:range rdf:resource="#HFloor"/>
  <rdfs:domain rdf:resource="#Terrace"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="bTavan">
    <rdfs:range rdf:resource="#Tavan"/>
    <rdfs:domain rdf:resource="#Bathroom"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="kapi">
    <rdfs:range rdf:resource="#Door"/>
    <rdfs:domain rdf:resource="#House"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="tavan">
    <rdfs:domain rdf:resource="#Room"/>
    <rdfs:range rdf:resource="#Tavan"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="taban">
    <rdfs:domain rdf:resource="#Room"/>
    <rdfs:range rdf:resource="#HFloor"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="Cins">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Parquet"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="Alan">
    <rdfs:domain rdf:resource="#HFloor"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
</rdf:RDF>
```

## A.2 Travel.owl

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns="http://www.owl-ontologies.com/travel.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
```

```
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
xml:base="http://www.owl-ontologies.com/travel.owl">
<owl:Class rdf:ID="Activity"/>
<owl:Class rdf:ID="Museums">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Sightseeing"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="BudgetHotelDestination">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:ID="Destination"/>
        <owl:Restriction>
          <owl:someValuesFrom>
            <owl:Class>
              <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:ID="BudgetAccommodation"/>
                <owl:Class rdf:ID="Hotel"/>
              </owl:intersectionOf>
            </owl:Class>
          </owl:someValuesFrom>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="hasAccommodation"/>
          </owl:onProperty>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>

</owl:Class>
<owl:Class rdf:ID="Capital">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="City"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="#Museums"/>
      <owl:onProperty>
```

```
        <owl:ObjectProperty rdf:ID="hasActivity"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Vehicle"/>
<owl:Class rdf:ID="Beach">
  <rdfs:subClassOf rdf:resource="#Destination"/>
</owl:Class>
<owl:Class rdf:ID="Sunbathing">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Relaxation"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="NationalPark">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom>
        <owl:Class rdf:ID="Hiking"/>
      </owl:someValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasActivity"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:ID="RuralArea"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasAccommodation"/>
      </owl:onProperty>
      <owl:someValuesFrom>
        <owl:Class rdf:ID="Campground"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

```
<owl:Class rdf:ID="BackpackersDestination">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Destination"/>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#hasAccommodation"/>
          </owl:onProperty>
          <owl:someValuesFrom>
            <owl:Class rdf:about="#BudgetAccommodation"/>
          </owl:someValuesFrom>
        </owl:Restriction>
        <owl:Restriction>
          <owl:someValuesFrom>
            <owl:Class>
              <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:ID="Sports"/>
                <owl:Class rdf:ID="Adventure"/>
              </owl:unionOf>
            </owl:Class>
          </owl:someValuesFrom>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#hasActivity"/>
          </owl:onProperty>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>

</owl:Class>
<owl:Class rdf:about="#Sightseeing">
  <owl:disjointWith>
    <owl:Class rdf:about="#Relaxation"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#Sports"/>
  </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#Activity"/>
```

```
    <owl:disjointWith>
      <owl:Class rdf:about="#Adventure"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:ID="RetireeDestination">

    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Destination"/>
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="#hasAccommodation"/>
            </owl:onProperty>
            <owl:someValuesFrom>
              <owl:Restriction>
                <owl:hasValue>
                  <AccommodationRating rdf:ID="ThreeStarRating">
                    <owl:differentFrom>
                      <AccommodationRating rdf:ID="TwoStarRating">
                        <owl:differentFrom>
                          <AccommodationRating
rdf:ID="OneStarRating">
                            <owl:differentFrom
rdf:resource="#ThreeStarRating"/>
                            <owl:differentFrom
rdf:resource="#TwoStarRating"/>
                          </AccommodationRating>
                        </owl:differentFrom>
                        <owl:differentFrom
rdf:resource="#ThreeStarRating"/>
                      </AccommodationRating>
                    </owl:differentFrom>
                    <owl:differentFrom
rdf:resource="#OneStarRating"/>
                  </AccommodationRating>
                </owl:hasValue>
                <owl:onProperty>
                  <owl:ObjectProperty rdf:ID="hasRating"/>
              </owl:onProperty>
            </owl:Restriction>
          </owl:someValuesFrom>
        </owl:Restriction>
        <owl:Restriction>
          <owl:someValuesFrom rdf:resource="#Sightseeing"/>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#hasActivity"/>
          </owl:onProperty>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="AccommodationRating">
  <owl:equivalentClass>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <AccommodationRating rdf:about="#OneStarRating"/>
        <AccommodationRating rdf:about="#TwoStarRating"/>
        <AccommodationRating rdf:about="#ThreeStarRating"/>
      </owl:oneOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="UrbanArea">
  <rdfs:subClassOf rdf:resource="#Destination"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#RuralArea"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:about="#Campground">
  <owl:disjointWith>
    <owl:Class rdf:ID="BedAndBreakfast"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Accommodation"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
```

```
    <owl:Restriction>
      <owl:hasValue rdf:resource="#OneStarRating"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasRating"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:disjointWith>
    <owl:Class rdf:about="#Hotel"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:about="#City">
  <rdfs:subClassOf rdf:resource="#UrbanArea"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasAccommodation"/>
      </owl:onProperty>
      <owl:someValuesFrom>
        <owl:Class rdf:ID="LuxuryHotel"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Safari">
  <rdfs:subClassOf rdf:resource="#Sightseeing"/>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Adventure"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="QuietDestination">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Destination"/>
        <owl:Class>
          <owl:complementOf>
            <owl:Class rdf:ID="FamilyDestination"/>
          </owl:complementOf>
          </owl:Class>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:about="#Hiking">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Sports"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#Relaxation">
    <owl:disjointWith rdf:resource="#Sightseeing"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#Adventure"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Sports"/>
    </owl:disjointWith>
    <rdfs:subClassOf rdf:resource="#Activity"/>
  </owl:Class>
  <owl:Class rdf:about="#FamilyDestination">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Destination"/>
          <owl:Restriction>
            <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
            >1</owl:minCardinality>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="#hasAccommodation"/>
            </owl:onProperty>
          </owl:Restriction>
          <owl:Restriction>
            <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
            >2</owl:minCardinality>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="#hasActivity"/>
          </owl:onProperty>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:about="#Adventure">
  <owl:disjointWith>
    <owl:Class rdf:about="#Sports"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Relaxation"/>
  <owl:disjointWith rdf:resource="#Sightseeing"/>
  <rdfs:subClassOf rdf:resource="#Activity"/>
</owl:Class>
<owl:Class rdf:ID="Yoga">
  <rdfs:subClassOf rdf:resource="#Relaxation"/>
</owl:Class>
<owl:Class rdf:about="#Sports">
  <rdfs:subClassOf rdf:resource="#Activity"/>
  <owl:disjointWith rdf:resource="#Adventure"/>
  <owl:disjointWith rdf:resource="#Sightseeing"/>
  <owl:disjointWith rdf:resource="#Relaxation"/>
</owl:Class>
<owl:Class rdf:about="#Hotel">
  <owl:disjointWith rdf:resource="#Campground"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#BedAndBreakfast"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Accommodation"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Town">
  <rdfs:subClassOf rdf:resource="#UrbanArea"/>
</owl:Class>
<owl:Class rdf:about="#RuralArea">
  <rdfs:subClassOf rdf:resource="#Destination"/>
  <owl:disjointWith rdf:resource="#UrbanArea"/>
</owl:Class>
```

```
<owl:Class rdf:about="#Accommodation">
</owl:Class>
<owl:Class rdf:ID="Surfing">
  <rdfs:subClassOf rdf:resource="#Sports"/>
</owl:Class>
<owl:Class rdf:about="#BudgetAccommodation">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Accommodation"/>
        <owl:Restriction>
          <owl:someValuesFrom>
            <owl:Class>
              <owl:oneOf rdf:parseType="Collection">
                <AccommodationRating rdf:about="#OneStarRating"/>
                <AccommodationRating rdf:about="#TwoStarRating"/>
              </owl:oneOf>
            </owl:Class>
          </owl:someValuesFrom>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#hasRating"/>
          </owl:onProperty>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:about="#BedAndBreakfast">
  <owl:disjointWith rdf:resource="#Hotel"/>
  <rdfs:subClassOf rdf:resource="#Accommodation"/>
  <owl:disjointWith rdf:resource="#Campground"/>
</owl:Class>
<owl:Class rdf:ID="BunjeeJumping">
  <rdfs:subClassOf rdf:resource="#Adventure"/>
</owl:Class>
<owl:Class rdf:about="#LuxuryHotel">
  <rdfs:subClassOf rdf:resource="#Hotel"/>
  <rdfs:subClassOf>
    <owl:Restriction>
```

```
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasRating"/>
      </owl:onProperty>
      <owl:hasValue rdf:resource="#ThreeStarRating"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="bus">
  <rdfs:subClassOf rdf:resource="#Vehicle"/>
</owl:Class>
<owl:Class rdf:ID="Farmland">
  <rdfs:subClassOf rdf:resource="#RuralArea"/>
</owl:Class>
<owl:Class rdf:ID="Contact"/>
<owl:Class rdf:ID="plane">
  <rdfs:subClassOf rdf:resource="#Vehicle"/>
</owl:Class>
<owl:ObjectProperty rdf:about="#hasActivity">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="isOfferedAt"/>
  </owl:inverseOf>
  <rdfs:range rdf:resource="#Activity"/>
  <rdfs:domain rdf:resource="#Destination"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasRating">
  <rdfs:domain rdf:resource="#Accommodation"/>
  <rdfs:range rdf:resource="#AccommodationRating"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasContact">
  <rdfs:range rdf:resource="#Contact"/>
  <rdfs:domain rdf:resource="#Activity"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isOfferedAt">
  <rdfs:range rdf:resource="#Destination"/>
  <owl:inverseOf rdf:resource="#hasActivity"/>
  <rdfs:domain rdf:resource="#Activity"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasAccommodation">
  <rdfs:range rdf:resource="#Accommodation"/>
```

```
    <rdfs:domain rdf:resource="#Destination"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasPart">
    <rdfs:domain rdf:resource="#Destination"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>
    <rdfs:range rdf:resource="#Destination"/>
  </owl:ObjectProperty>
  <owl:FunctionalProperty rdf:ID="hasCity">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#Contact"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="hasZipCode">
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#Contact"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="hasStreet">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#Contact"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="hasEMail">
    <rdfs:domain rdf:resource="#Contact"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <RuralArea rdf:ID="Woomera"/>
  <Beach rdf:ID="BondiBeach"/>
  <NationalPark rdf:ID="BlueMountains"/>
  <NationalPark rdf:ID="Warrumbungles"/>
  <Capital rdf:ID="Canberra"/>
  <Beach rdf:ID="CurrawongBeach"/>
  <LuxuryHotel rdf:ID="FourSeasons"/>
  <Capital rdf:ID="Sydney">
    <hasPart rdf:resource="#BondiBeach"/>
    <hasAccommodation rdf:resource="#FourSeasons"/>
    <hasPart rdf:resource="#CurrawongBeach"/>
  </Capital>
  <RuralArea rdf:ID="CapeYork"/>
  <Town rdf:ID="Coonabarabran"/>
  <City rdf:ID="Cairns"/>
</rdf:RDF>
```

## A.3 Meal.owl

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/unnamed.owl#"
  xml:base="http://www.owl-ontologies.com/unnamed.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Pizza">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Food"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="AppleJuice">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="FruitJuice"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Chicken">
    <rdfs:subClassOf rdf:resource="#Food"/>
```

```
</owl:Class>
<owl:Class rdf:ID="OrangeJuice">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#FruitJuice"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Rice-pudding">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="WithMilk"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Ayran">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="WithoutAlcohol"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Coke">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#WithoutAlcohol"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Ice-cream">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#WithMilk"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Meat">
  <rdfs:subClassOf rdf:resource="#Food"/>
</owl:Class>
<owl:Class rdf:ID="WithAlcohol">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Drink"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Desert"/>
<owl:Class rdf:ID="Baklava">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="WithoutMilk"/>
  </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#WithMilk">
    <rdfs:subClassOf rdf:resource="#Desert"/>
  </owl:Class>
  <owl:Class rdf:ID="Fish">
    <rdfs:subClassOf rdf:resource="#Food"/>
  </owl:Class>
  <owl:Class rdf:ID="Salad"/>
  <owl:Class rdf:ID="Kadayif">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#WithoutMilk"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#WithoutAlcohol">
    <rdfs:subClassOf rdf:resource="#Drink"/>
  </owl:Class>
  <owl:Class rdf:about="#FruitJuice">
    <rdfs:subClassOf rdf:resource="#WithoutAlcohol"/>
  </owl:Class>
  <owl:Class rdf:about="#WithoutMilk">
    <rdfs:subClassOf rdf:resource="#Desert"/>
  </owl:Class>
</rdf:RDF>
```

## A.4 PegasusService.owl

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE uridef [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY service "http://www.daml.org/services/owl-
s/1.1/Service.owl">
  <!ENTITY profile "http://www.daml.org/services/owl-
s/1.1/Profile.owl">
  <!ENTITY process "http://www.daml.org/services/owl-
s/1.1/Process.owl">
  <!ENTITY grounding "http://www.daml.org/services/owl-
s/1.1/Grounding.owl">
]>
```

```
<rdf:RDF
  xmlns:rdf="&rdf;#"
  xmlns:rdfs="&rdfs;#"
  xmlns:owl="&owl;#"
  xmlns:xsd="&xsd;#"
  xmlns:service="&service;#"
  xmlns:profile="&profile;#"
  xmlns:process="&process;#"
  xmlns:grounding="&grounding;#"
 >


<owl:Ontology rdf:about="">
      <owl:imports rdf:resource="&service;"/>
      <owl:imports rdf:resource="&profile;"/>
      <owl:imports rdf:resource="&process;"/>
      <owl:imports rdf:resource="&grounding;"/>
</owl:Ontology>

<!-- Service description -->
<service:Service rdf:ID="PegasusServiceService">
      <service:presents rdf:resource="#PegasusServiceProfile"/>

      <service:describedBy rdf:resource="#PegasusServiceProcess"/>

      <service:supports rdf:resource="#PegasusServiceGrounding"/>
</service:Service>

<!-- Profile description -->
<profile:Profile rdf:ID="PegasusServiceProfile">
      <service:isPresentedBy
rdf:resource="#PegasusServiceService"/>

      <profile:serviceName
xml:lang="en">PegasusService</profile:serviceName>
      <profile:textDescription
xml:lang="en">PegasusServiceService</profile:textDescription>

      <profile:hasInput rdf:resource="#departureTime"/>
      <profile:hasInput rdf:resource="#arrivalTime"/>
      <profile:hasInput rdf:resource="#destination"/>
      <profile:hasInput rdf:resource="#departure"/>
      <profile:hasOutput rdf:resource="#cost"/>
</profile:Profile>

<!-- Process description -->
<process:AtomicProcess rdf:ID="PegasusServiceProcess">
      <service:describes rdf:resource="#PegasusServiceService"/>

      <process:hasInput rdf:resource="#departureTime"/>
      <process:hasInput rdf:resource="#arrivalTime"/>
      <process:hasInput rdf:resource="#destination"/>
      <process:hasInput rdf:resource="#departure"/>

      <process:hasOutput rdf:resource="#cost"/>
</process:AtomicProcess>

<process:Input rdf:ID="departureTime">
```

```
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Departur
eTime.owl#DepartureTime</process:parameterType>
      <rdfs:label>departureTime</rdfs:label>
</process:Input>
<process:Input rdf:ID="arrivalTime">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/ArrivalT
ime.owl#ArrivalTime</process:parameterType>
      <rdfs:label>arrivalTime</rdfs:label>
</process:Input>
<process:Input rdf:ID="destination">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Destinat
ionPlace.owl#DestinationPlace</process:parameterType>
      <rdfs:label>destination</rdfs:label>
</process:Input>
<process:Input rdf:ID="departure">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Departur
ePlace.owl#DeparturePlace</process:parameterType>
      <rdfs:label>departure</rdfs:label>
</process:Input>

<process:Output rdf:ID="cost">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Cost.owl
#Cost</process:parameterType>
      <rdfs:label>cost</rdfs:label>
</process:Output>

</rdf:RDF>
```

## A.5 THYService.owl

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE uridef [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY service "http://www.daml.org/services/owl-
s/1.1/Service.owl">
  <!ENTITY profile "http://www.daml.org/services/owl-
s/1.1/Profile.owl">
  <!ENTITY process "http://www.daml.org/services/owl-
s/1.1/Process.owl">
  <!ENTITY grounding "http://www.daml.org/services/owl-
s/1.1/Grounding.owl">
]>

<rdf:RDF
  xmlns:rdf="&rdf;#"
  xmlns:rdfs="&rdfs;#"
  xmlns:owl="&owl;#"
  xmlns:xsd="&xsd;#"
  xmlns:service="&service;#"
  xmlns:profile="&profile;#"
  xmlns:process="&process;#"
  xmlns:grounding="&grounding;#"
 >


<owl:Ontology rdf:about="">
      <owl:imports rdf:resource="&service;"/>
      <owl:imports rdf:resource="&profile;"/>
      <owl:imports rdf:resource="&process;"/>
      <owl:imports rdf:resource="&grounding;"/>
</owl:Ontology>

<!-- Service description -->
<service:Service rdf:ID="THYServiceService">
      <service:presents rdf:resource="#THYServiceProfile"/>

      <service:describedBy rdf:resource="#THYServiceProcess"/>

      <service:supports rdf:resource="#THYServiceGrounding"/>
</service:Service>

<!-- Profile description -->
<profile:Profile rdf:ID="THYServiceProfile">
      <service:isPresentedBy rdf:resource="#THYServiceService"/>

      <profile:serviceName
xml:lang="en">THYService</profile:serviceName>
      <profile:textDescription
xml:lang="en">THYServiceService</profile:textDescription>

      <profile:hasInput rdf:resource="#duration"/>
      <profile:hasInput rdf:resource="#destination"/>
      <profile:hasInput rdf:resource="#departure"/>
      <profile:hasOutput rdf:resource="#cost"/>
</profile:Profile>

<!-- Process description -->
<process:AtomicProcess rdf:ID="THYServiceProcess">
      <service:describes rdf:resource="#THYServiceService"/>

      <process:hasInput rdf:resource="#duration"/>
      <process:hasInput rdf:resource="#destination"/>
      <process:hasInput rdf:resource="#departure"/>

      <process:hasOutput rdf:resource="#cost"/>
</process:AtomicProcess>

<process:Input rdf:ID="duration">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Duration
.owl#Duration</process:parameterType>
      <rdfs:label>duration</rdfs:label>
</process:Input>
<process:Input rdf:ID="destination">
```

```
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Destinat
ionPlace.owl#DestinationPlace</process:parameterType>
      <rdfs:label>destination</rdfs:label>
</process:Input>
<process:Input rdf:ID="departure">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Departur
ePlace.owl#DeparturePlace</process:parameterType>
      <rdfs:label>departure</rdfs:label>
</process:Input>

<process:Output rdf:ID="cost">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Cost.owl
#Cost</process:parameterType>
      <rdfs:label>cost</rdfs:label>
</process:Output>

</rdf:RDF>
```

## A.6 AtlasJetService.owl

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE uridef [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY service "http://www.daml.org/services/owl-
s/1.1/Service.owl">
  <!ENTITY profile "http://www.daml.org/services/owl-
s/1.1/Profile.owl">
  <!ENTITY process "http://www.daml.org/services/owl-
s/1.1/Process.owl">
  <!ENTITY grounding "http://www.daml.org/services/owl-
s/1.1/Grounding.owl">
]>

<rdf:RDF
  xmlns:rdf="&rdf;#"
  xmlns:rdfs="&rdfs;#"
  xmlns:owl="&owl;#"
  xmlns:xsd="&xsd;#"
  xmlns:service="&service;#"
  xmlns:profile="&profile;#"
  xmlns:process="&process;#"
  xmlns:grounding="&grounding;#"
 >


<owl:Ontology rdf:about="">
      <owl:imports rdf:resource="&service;"/>
      <owl:imports rdf:resource="&profile;"/>
      <owl:imports rdf:resource="&process;"/>
      <owl:imports rdf:resource="&grounding;"/>
</owl:Ontology>

<!-- Service description -->
<service:Service rdf:ID="AtlasJetServiceService">
      <service:presents rdf:resource="#AtlasJetServiceProfile"/>

      <service:describedBy rdf:resource="#AtlasJetServiceProcess"/>

      <service:supports rdf:resource="#AtlasJetServiceGrounding"/>
</service:Service>

<!-- Profile description -->
<profile:Profile rdf:ID="AtlasJetServiceProfile">
      <service:isPresentedBy
rdf:resource="#AtlasJetServiceService"/>

      <profile:serviceName
xml:lang="en">AtlasJetService</profile:serviceName>
      <profile:textDescription
xml:lang="en">AtlasJetServiceService</profile:textDescription>

      <profile:hasInput rdf:resource="#departureTime"/>
      <profile:hasInput rdf:resource="#arrivalTime"/>
      <profile:hasInput rdf:resource="#destination"/>
      <profile:hasInput rdf:resource="#departure"/>
      <profile:hasOutput rdf:resource="#cost"/>
</profile:Profile>

<!-- Process description -->
<process:AtomicProcess rdf:ID="AtlasJetServiceProcess">
      <service:describes rdf:resource="#AtlasJetServiceService"/>

      <process:hasInput rdf:resource="#departureTime"/>
      <process:hasInput rdf:resource="#arrivalTime"/>
      <process:hasInput rdf:resource="#destination"/>
      <process:hasInput rdf:resource="#departure"/>

      <process:hasOutput rdf:resource="#cost"/>
</process:AtomicProcess>

<process:Input rdf:ID="departureTime">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/AtlasJet
DepartureTime.owl#AtlasJetDepartureTime</process:parameterType>
      <rdfs:label>departureTime</rdfs:label>
</process:Input>
<process:Input rdf:ID="arrivalTime">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/AtlasJet
ArrivalTime.owl#AtlasJetArrivalTime</process:parameterType>
      <rdfs:label>arrivalTime</rdfs:label>
</process:Input>
<process:Input rdf:ID="destination">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Destinat
ionPlace.owl#DestinationPlace</process:parameterType>
```

```
      <rdfs:label>destination</rdfs:label>
</process:Input>
<process:Input rdf:ID="departure">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Departur
ePlace.owl#DeparturePlace</process:parameterType>
      <rdfs:label>departure</rdfs:label>
</process:Input>

<process:Output rdf:ID="cost">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Cost.owl
#Cost</process:parameterType>
      <rdfs:label>cost</rdfs:label>
</process:Output>

</rdf:RDF>
```

## A.7 KAirlinesService.owl

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE uridef [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY service "http://www.daml.org/services/owl-
s/1.1/Service.owl">
  <!ENTITY profile "http://www.daml.org/services/owl-
s/1.1/Profile.owl">
  <!ENTITY process "http://www.daml.org/services/owl-
s/1.1/Process.owl">
  <!ENTITY grounding "http://www.daml.org/services/owl-
s/1.1/Grounding.owl">
]>

<rdf:RDF
  xmlns:rdf="&rdf;#"
  xmlns:rdfs="&rdfs;#"
  xmlns:owl="&owl;#"
  xmlns:xsd="&xsd;#"
  xmlns:service="&service;#"
  xmlns:profile="&profile;#"
  xmlns:process="&process;#"
  xmlns:grounding="&grounding;#"
 >


<owl:Ontology rdf:about="">
      <owl:imports rdf:resource="&service;"/>
      <owl:imports rdf:resource="&profile;"/>
      <owl:imports rdf:resource="&process;"/>
      <owl:imports rdf:resource="&grounding;"/>
</owl:Ontology>
```

```
<!-- Service description -->
<service:Service rdf:ID="KAirlinesServiceService">
      <service:presents rdf:resource="#KAirlinesServiceProfile"/>

      <service:describedBy
rdf:resource="#KAirlinesServiceProcess"/>

      <service:supports rdf:resource="#KAirlinesServiceGrounding"/>
</service:Service>

<!-- Profile description -->
<profile:Profile rdf:ID="KAirlinesServiceProfile">
      <service:isPresentedBy
rdf:resource="#KAirlinesServiceService"/>

      <profile:serviceName
xml:lang="en">KAirlinesService</profile:serviceName>
      <profile:textDescription
xml:lang="en">KAirlinesServiceService</profile:textDescription>

      <profile:hasInput rdf:resource="#departureTime"/>
      <profile:hasInput rdf:resource="#arrivalTime"/>
      <profile:hasInput rdf:resource="#destination"/>
      <profile:hasInput rdf:resource="#departure"/>
      <profile:hasOutput rdf:resource="#kAirlinesReceipt"/>
</profile:Profile>

<!-- Process description -->
<process:AtomicProcess rdf:ID="KAirlinesServiceProcess">
      <service:describes rdf:resource="#KAirlinesServiceService"/>

      <process:hasInput rdf:resource="#departureTime"/>
      <process:hasInput rdf:resource="#arrivalTime"/>
      <process:hasInput rdf:resource="#destination"/>
      <process:hasInput rdf:resource="#departure"/>

      <process:hasOutput rdf:resource="#kAirlinesReceipt"/>
</process:AtomicProcess>

<process:Input rdf:ID="departureTime">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Departur
eTime.owl#DepartureTime</process:parameterType>
      <rdfs:label>departureTime</rdfs:label>
</process:Input>
<process:Input rdf:ID="arrivalTime">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/ArrivalT
ime.owl#ArrivalTime</process:parameterType>
      <rdfs:label>arrivalTime</rdfs:label>
</process:Input>
<process:Input rdf:ID="destination">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Destinat
ionPlace.owl#DestinationPlace</process:parameterType>
      <rdfs:label>destination</rdfs:label>
</process:Input>
```

```
<process:Input rdf:ID="departure">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Departur
ePlace.owl#DeparturePlace</process:parameterType>
      <rdfs:label>departure</rdfs:label>
</process:Input>

<process:Output rdf:ID="kAirlinesReceipt">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/KAirline
sReceipt.owl#KAirlinesReceipt</process:parameterType>
      <rdfs:label>cost</rdfs:label>
</process:Output>

</rdf:RDF>
```

## A.8 HiltonHotelService.owl

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE uridef [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY service "http://www.daml.org/services/owl-
s/1.1/Service.owl">
  <!ENTITY profile "http://www.daml.org/services/owl-
s/1.1/Profile.owl">
  <!ENTITY process "http://www.daml.org/services/owl-
s/1.1/Process.owl">
  <!ENTITY grounding "http://www.daml.org/services/owl-
s/1.1/Grounding.owl">
]>

<rdf:RDF
  xmlns:rdf="&rdf;#"
  xmlns:rdfs="&rdfs;#"
  xmlns:owl="&owl;#"
  xmlns:xsd="&xsd;#"
  xmlns:service="&service;#"
  xmlns:profile="&profile;#"
  xmlns:process="&process;#"
  xmlns:grounding="&grounding;#"
 >


<owl:Ontology rdf:about="">
      <owl:imports rdf:resource="&service;"/>
      <owl:imports rdf:resource="&profile;"/>
      <owl:imports rdf:resource="&process;"/>
      <owl:imports rdf:resource="&grounding;"/>
</owl:Ontology>

<!-- Service description -->
```

```
<service:Service rdf:ID="HiltonHotelServiceService">
      <service:presents rdf:resource="#HiltonHotelServiceProfile"/>

      <service:describedBy
rdf:resource="#HiltonHotelServiceProcess"/>

      <service:supports
rdf:resource="#HiltonHotelServiceGrounding"/>
</service:Service>

<!-- Profile description -->
<profile:Profile rdf:ID="HiltonHotelServiceProfile">
      <service:isPresentedBy
rdf:resource="#HiltonHotelServiceService"/>

      <profile:serviceName
xml:lang="en">HiltonHotelService</profile:serviceName>
      <profile:textDescription
xml:lang="en">HiltonHotelServiceService</profile:textDescription>

      <profile:hasInput rdf:resource="#arrivalTime"/>
      <profile:hasInput rdf:resource="#leavingTime"/>
      <profile:hasInput rdf:resource="#place"/>
      <profile:hasInput rdf:resource="#roomType"/>
      <profile:hasOutput rdf:resource="#cost"/>
</profile:Profile>

<!-- Process description -->
<process:AtomicProcess rdf:ID="HiltonHotelServiceProcess">
      <service:describes
rdf:resource="#HiltonHotelServiceService"/>

      <process:hasInput rdf:resource="#arrivalTime"/>
      <process:hasInput rdf:resource="#leavingTime"/>
      <process:hasInput rdf:resource="#place"/>
      <process:hasInput rdf:resource="#roomType"/>

      <process:hasOutput rdf:resource="#cost"/>
</process:AtomicProcess>

<process:Input rdf:ID="arrivalTime">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/ArrivalT
ime.owl#ArrivalTime</process:parameterType>
      <rdfs:label>duration</rdfs:label>
</process:Input>
<process:Input rdf:ID="leavingTime">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/LeavingT
ime.owl#LeavingTime</process:parameterType>
      <rdfs:label>destination</rdfs:label>
</process:Input>
<process:Input rdf:ID="place">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Place.ow
l#Place</process:parameterType>
      <rdfs:label>departure</rdfs:label>
</process:Input>
```

```
<process:Input rdf:ID="roomType">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/RoomType
.owl#RoomType</process:parameterType>
      <rdfs:label>departure</rdfs:label>
</process:Input>

<process:Output rdf:ID="cost">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Cost.owl
#Cost</process:parameterType>
      <rdfs:label>cost</rdfs:label>
</process:Output>

</rdf:RDF>
```

### A.9 HavasShuttleService.owl

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE uridef [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY service "http://www.daml.org/services/owl-
s/1.1/Service.owl">
  <!ENTITY profile "http://www.daml.org/services/owl-
s/1.1/Profile.owl">
  <!ENTITY process "http://www.daml.org/services/owl-
s/1.1/Process.owl">
  <!ENTITY grounding "http://www.daml.org/services/owl-
s/1.1/Grounding.owl">
]>

<rdf:RDF
  xmlns:rdf="&rdf;#"
  xmlns:rdfs="&rdfs;#"
  xmlns:owl="&owl;#"
  xmlns:xsd="&xsd;#"
  xmlns:service="&service;#"
  xmlns:profile="&profile;#"
  xmlns:process="&process;#"
  xmlns:grounding="&grounding;#"
 >


<owl:Ontology rdf:about="">
      <owl:imports rdf:resource="&service;"/>
      <owl:imports rdf:resource="&profile;"/>
      <owl:imports rdf:resource="&process;"/>
      <owl:imports rdf:resource="&grounding;"/>
</owl:Ontology>

<!-- Service description -->
```

```
<service:Service rdf:ID="HavasShuttleServiceService">
      <service:presents
rdf:resource="#HavasShuttleServiceProfile"/>

      <service:describedBy
rdf:resource="#HavasShuttleServiceProcess"/>

      <service:supports
rdf:resource="#HavasShuttleServiceGrounding"/>
</service:Service>

<!-- Profile description -->
<profile:Profile rdf:ID="HavasShuttleServiceProfile">
      <service:isPresentedBy
rdf:resource="#HavasShuttleServiceService"/>

      <profile:serviceName
xml:lang="en">HavasShuttleService</profile:serviceName>
      <profile:textDescription
xml:lang="en">HavasShuttleServiceService</profile:textDescription>

      <profile:hasInput rdf:resource="#airport"/>
      <profile:hasInput rdf:resource="#destinationHotel"/>
      <profile:hasInput rdf:resource="#departureTime"/>
      <profile:hasOutput rdf:resource="#cost"/>
</profile:Profile>

<!-- Process description -->
<process:AtomicProcess rdf:ID="HavasShuttleServiceProcess">
      <service:describes
rdf:resource="#HavasShuttleServiceService"/>

      <process:hasInput rdf:resource="#airport"/>
      <process:hasInput rdf:resource="#destinationHotel"/>
      <process:hasInput rdf:resource="#departureTime"/>

      <process:hasOutput rdf:resource="#cost"/>
</process:AtomicProcess>

<process:Input rdf:ID="airport">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Airport.
owl#Airport</process:parameterType>
      <rdfs:label>airport</rdfs:label>
</process:Input>
<process:Input rdf:ID="destinationHotel">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Destinat
ionPlace.owl#DestinationPlace</process:parameterType>
      <rdfs:label>destinationHotel</rdfs:label>
</process:Input>
<process:Input rdf:ID="departureTime">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Departur
eTime.owl#DepartureTime</process:parameterType>
      <rdfs:label>departureTime</rdfs:label>
</process:Input>
```

```
<process:Output rdf:ID="cost">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">http://localhost:8080/examples/Cost.owl
#Cost</process:parameterType>
      <rdfs:label>cost</rdfs:label>
</process:Output>

</rdf:RDF>
```