SEMANTICALLY ENRICHED WEB SERVICE COMPOSITION IN MOBILE ENVIRONMENTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

K. ALPAY ERTÜRKMEN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF INFORMATION SYSTEMS

SEPTEMBER 2003

Approval of the Graduate School of Informatics.

Prof. Dr. Neşe Yalabik
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Onur Demirörs
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Asuman Doğaç
Supervisor

Examining Committee Members

Prof. Dr. Asuman Doğaç

Prof. Dr. Semih Bilgen

Assoc. Prof. Dr. Onur Demirörs

Assist. Prof. Dr. Ahmet Coşar

Dr. Altan Koçyiğit

ABSTRACT

SEMANTICALLY ENRICHED WEB SERVICE COMPOSITION IN MOBILE ENVIRONMENTS

Ertürkmen, Kulubey Alpay

M.Sc., Department of Information Systems

Supervisor: Prof. Dr. Asuman Doğaç

September 2003, 110 pages

Web Services are self-contained, self-describing, modular applications that can be published, located, and invoked through XML artefacts across the Web. Web services technologies can be applied to many kinds of applications, where they offer considerable advantages compared to the old world of product-specific APIs, platform-specific coding, and other "brittle" technology restrictions.

Currently there are millions of web services available on the web due to the increase in e-commerce business volume. Web services can be discovered using public registries and invoked through respective interfaces. However how to automatically find, compose, invoke and monitor the web services is still an issue. The automatic discovery, composition, invocation and monitoring of web services require that semantics will be attached to service definitions.

The focus of this thesis is on the composition of web services. The approach taken is to extend the DAML-S ontology that is used to define the semantics of services to include the "succeeding services" for any service provided. These definitions for individual service instances are declared by the service providers.

They are presented to the users of the service to construct a workflow in a mobile environment. The workflow generated is represented both graphically in the mobile device and in XML-format as a BPEL4WS document.

The aim of this thesis is to prove that it is possible to build a semi-automatic web service composition utility incorporating semantic constructs, using a mobile device. The generated workflow is suitable for deployment on an engine where it can be executed multiple times with different configurations.

Keywords: Web Service Composition, Web Service Semantics, DAML-S, BPEL4WS, OWL-S.

ÖZ

MOBİL ORTAMDA SEMANTİKLE ZENGİNLEŞTİRİLMİŞ AĞ SERVİSİ DÜZENLENMESİ

Ertürkmen, Kulubey Alpay
Yüksek Lisans, Bilişim Sistemleri Bölümü
Tez Yöneticisi: Prof. Dr. Asuman Doğaç

Eylül 2003, 110 sayfa

Ağ servisleri kendi içlerinde kapalı, kendi kendilerini tanımlayabilen, Web üzerinden XML elemanlar kullanarak yayınlanabilen, bulunabilen ve çalıştırılabilen modüler uygulamalardır. Ağ servis teknolojileri çok çeşitli yerlerde uygulanabilir ve buralarda eski tip API'ler, platforma bağlı kodlar gibi, kırılgan teknoloji kısıtlamalarına kıyasla büyük avantajlar sunarlar.

Artan elektronik ticaret hacmine bağlı olarak şu anda ağ üzerinde milyonlarca ağ servisi bulunmaktadır. Ağ servisleri saklayıcılar üzerinden bulunabilir ve gerekli arayüzler üzerinden çalıştırılabilir. Fakat bu servislerin otomatik olarak nasıl bulunacağı, düzenleneceği, çalıştırılacağı ve gözleneceği henüz bir tartışma konusudur. Ağ servislerinin otomatik bulunması, düzenlenmesi, çalıştırılması ve gözlenmesi icin servis tanımlarına semantiğin de eklenmesi gerekmektedir.

Bu tez ağ servislerinin düzenlenmesi üzerinedir. İzlenen yol servis semantiklerinin tanımlanmasında kullanılan DAML-S ontolojisinin, sunulan servisler icin "takip eden servis" tanımını içerecek şekilde genişletilmesidir. Bu servis

tanımları servisi sunanlar tarafından belirlenir ve kullanıcıya bir iş akışı çıkarması icin mobil bir ortamda sunulur. çıkarılan iş akısı hem görsel olarak mobil cihazda, hem de XML formatında bir BPEL4WS dökümanı şeklinde oluşturulur.

Bu tezin amacı semantik yapılar içeren yarı-otomatik bir ağ servisi düzenleme uygulamasının bir mobil cihazda bile gerçekleştirilebileceğini gostermektir. Oluşturulan iş akışı ise bir iş akış motoruna yüklenerek değişik girdilerle birçok kere çalıştırılmaya uygundur.

Anahtar Kelimeler: Ağ Servis Düzenlenmesi, Ağ Servis Semantiği, DAML-S, BPEL4WS, OWL-S.

To my family

ACKNOWLEDGMENTS

I would like to thank my supervisor Prof.Dr.Asuman Doğaç for her guidance, support, patience and motivation during this study.

I would like to thank Yıldıray Kabak, Meltem Sönmez and Yasemin Salihoğlu for their help.

Finally, I am grateful to Gökçe Banu Laleci for her vision, endless patience and invaluable support.

TABLE OF CONTENTS

ABSTRACT iii

ÖZ v

DEDICATON vii

ACKNOWLEDGMENTS viii

TABLE OF CONTENTS ix

LIST OF TABLES xi

LIST OF FIGURES xii

LIST OF ACRONYMS AND ABBREVIATIONS xiii

CHAPTER

1 Introduction 1

2 Enabling Technologies 6

2.1 Web Services 6

2.1.1 WSDL 8

2.2 Semantic Web and Web Service Semantics 14

2.2.1 DAML+OIL 17

2.2.2 DAML-S 21

2.3 BPEL4WS 30

2.4 J2ME 36

3 System Architecture 40

4 Design and Implementation 47

4.1 Module Objectives 47

4.2 Ontology Processing 48

4.3 Messaging Scheme 54
4.4 Presentation (Mobile) Layer 58
4.5 Application Layer 68
4.5.1 Controller Sub-Module 69
4.5.2 BPEL4WS Generator Sub-Module 71
5 Conclusions and Future Work 74
REFERENCES 76
APPENDICES 81
A Classes of Mobile Module 81
B Classes of Application Module 83
C Extended DAML-S Profile 85
D DAML-S Tourism Mini-Ontology 97
E Generated BPEL4WS Document 103
F WSDL Documents 106
F.1 SeekHotels.com.wsdl 106
F.2 WorldofWonders.com.wsdl 107
F.3 Health-onHotel.wsdl 108
F.4 Ph.DCard.wsdl 108
F.5 WHOOPSParcelService.wsdl 109

LIST OF TABLES

4.1 Ontology Database Tables 54
4.2 DAML-S Ontology - Ontology Database Mapping 55
4.3 Messaging Scheme 59

LIST OF FIGURES

1.1 Global System View 4

2.1 Web Service Model 7
2.2 Top level of the service ontology 23
2.3 Generic and implementation specific J2ME architecture 39

3.1 The Overall System Architecture 45

4.1 Logical Relationships in Ontology Database 53
4.2 Overall System Flow Chart 56
4.3 Sequence Diagram for Initialization 56
4.4 Sequence Diagram for Service Selection 57
4.5 Sequence Diagram for Termination 58
4.6 User Interface Map for the Mobile Module 59
4.7 Mobile Module Welcome Message Screen 60
4.8 Mobile Module Main Menu 61
4.9 Mobile Module List of Generic Services 62
4.10 Mobile Module List of Service Instances 63
4.11 Mobile Module Input Properties Text Boxes 64
4.12 Mobile Module List of Workflow Constructs 65
4.13 Mobile Module Graphical Representation of the Workflow . . . 66
4.14 Mobile Module List of Succeeding Services 67

LIST OF ACRONYMS AND ABBREVIATIONS

API: Application Programming Interface

BPEL4J: Business Process Execution Language for Web Services Java Runtime

BPEL4WS: Business Process Execution Language for Web Services

B2B: Business to Business e-Commerce

B2C: Business to Consumer e-Commerce

CDC: Connected Device Configuration

CEO: Chief Executive Officer

CLDC: Connected Limited Device Configuration

DAML-S: DARPA Agent Markup Language based Web Service Ontology

ebXML: Electronic Business using Extensible Markup Language

EDI: Electronic Data Interchange

EU: European Union

GUI: Graphical User Interface

HTML: Hyper Text Markup Language

HTTP: Hyper Text Transfer Protocol

IBM: International Business Machines

JVM: Java Virtual Machine

J2EE: Java 2 Enterprise Edition

J2ME: Java 2 Micro Edition

J2SE: Java 2 Standard Edition

MIDP: Mobile Information Device Profile

OWL: Web Ontology Language

OWL-S: Web Ontology Language based Web Service Ontology

PDA: Personal Digital Assistant

RDF: Resource Description Framework

RDFS: Resource Description Framework Schema

RPC: Remote Procedure Call

SMTP: Simple Mail Transfer Protocol

SOAP: Simple Object Access Protocol

SQL: Standard Query Language

TCP/IP: Transmission Control Protocol / Internet Protocol

UDDI: Universal Description, Discovery and Integration of Web Services

URI: Uniform Resource Identifier

URL: Uniform Resource Locator

WSDK: IBM WebSphere Software Development Kit for Web Services

WSDL: Web Services Description Language

WSFL: Web Services Flow Language

WWW: World Wide Web

W3C: World Wide Web Consortium

XML: Extensible Markup Language

CHAPTER 1

Introduction

This thesis describes how semantically enriched web services can be composed and used in mobile environments.

As the hardware components of computers get smaller in size and offer more computational power, mobile computing became a reality. As of today, state-of-the-art PDAs (Personal Digital Assistant) offer the same computational power, memory and storage space as the high-end desktop computer of 5-6 years ago. The small size and high computing power, combined with the wireless communication technologies offering high data transmission speeds up to tens of megabits per second, created a new breed of communication technologies.

In the meantime, web has evolved from a communication medium to a social concept and pulled the business into itself. Businesses used the web as a digital marketplace where reaching millions was a matter of seconds. Also finding business partners are much easy in this medium.

Web Services carrying their roots from Electronic Data Interchange (EDI) come into existence on the web to create large volumes of e-Commerce both in Business-to-Business (B2B) and Business-to-Consumer (B2C) domains.

The format centered web contains vast amount of data and information, where nothing could be found easily. The creators of the web have a vision: The Semantic Web [24], is the only way to solve this problem, where every piece of

data and information is understandable by the computers as well as humans. Extensible Markup Language (XML); the elegant solution to messaging, data storage, data structuring etc. in the sense that it is flexible, extensible, simple and platform-independent; came in to the scene to offer its potential. XML was one of the interesting areas of computer research. Many extensions had been done to XML, the most important of which for the Semantic Web is the Resource Description Framework (RDF) [32]. RDF introduced the graph theory to XML to describe everything as a triplet: "Subject hasProperty Object".

DAML-OIL [12] is an effort to create classes that describe part of an object domain. It uses RDF constructs to describe the relationships between domains. These classes and properties build up the basis for the semantic constructs the Semantic Web needs.

DAML-S [11] builds on DAML-OIL to provide an ontology for describing Web Services. DAML-S aims the following four concepts to be realized: Automatic discovery, invocation, composition and monitoring of web services. Automatic discovery of web services needs the integration of the semantic constructs with public registries like Universal Description, Discovery and Integration of Web Services (UDDI) [36] and Electronic Business using eXtensible Markup Language (ebXML) [17]. In the work described in [23] DAML-S ontology is converted into ebXML registry constructs and successfully stored and queried through the ebXML registry.

Automatic invocation of web services has become a reality with the introduction of technologies like Web Services Description Language (WSDL) [39] and Simple Object Access Protocol (SOAP) [34] that define the implementations of web services in a platform-independent manner. With the aid of the semantic definitions of the services, developers can build platforms where services can be automatically discovered from public registries and invoked using WSDL definitions through protocols like HTTP or SOAP.

Automatic composition of web services requires more research in the sense that although the services can be discovered, some matchmaking process is

needed to check whether the services can actually work together in a composition. Current research builds on the fact that outputs of a given web service would provide a perfect or an approximate match to the inputs of another service, thus the service may succeed the current one in the flow [31].

In this thesis a semi automated approach is taken where the possible succeeding services is discovered automatically and presented to the user. The decision on the choice of the succeeding service is taken by the user.

In this thesis, the DAML-S service ontology is extended to include succeeding service definitions for each service. The choice of succeeding service definitions are left to the service providers to decide. Also a mini-tourism ontology is defined to demonstrate how the inputs and the outputs of the services are semantically matchable.

The system is designed in three layers: presentation, application and data layer. Data Layer stores the service ontology in an Ontology Database. Application Layer is responsible of connecting the Presentation Layer to the Data Layer and producing an XML representation of the workflow generated. The Presentation Layer presents service information to the user and gets user inputs.

The XML representation for the workflow is chosen to be Business Process Execution Language for Web Services (BPEL4WS) [3]. BPEL4WS is designed to represent web service compositions in XML format and provide an engine where this workflow can be executed. An alpha version BPEL4WS engine (BPEL4J) is available from IBM's "alphaWorks Emerging Technologies" [1].

A workflow is created with the assumption that it will be executed repeatedly. A single-use workflow generation is not a feasible effort. Since the workflow resides in the engine for a period of time, it can be executed several times. The workflows can be executed with the same configuration, or with a new set of inputs every time it is executed. A web interface can be configured to take the new set of inputs. The process may also be configured as a web service. In this case, protocols such as SOAP can be used to invoke the BPEL4WS workflow that has been composed and deployed on BPEL4J. A complex banking trans-

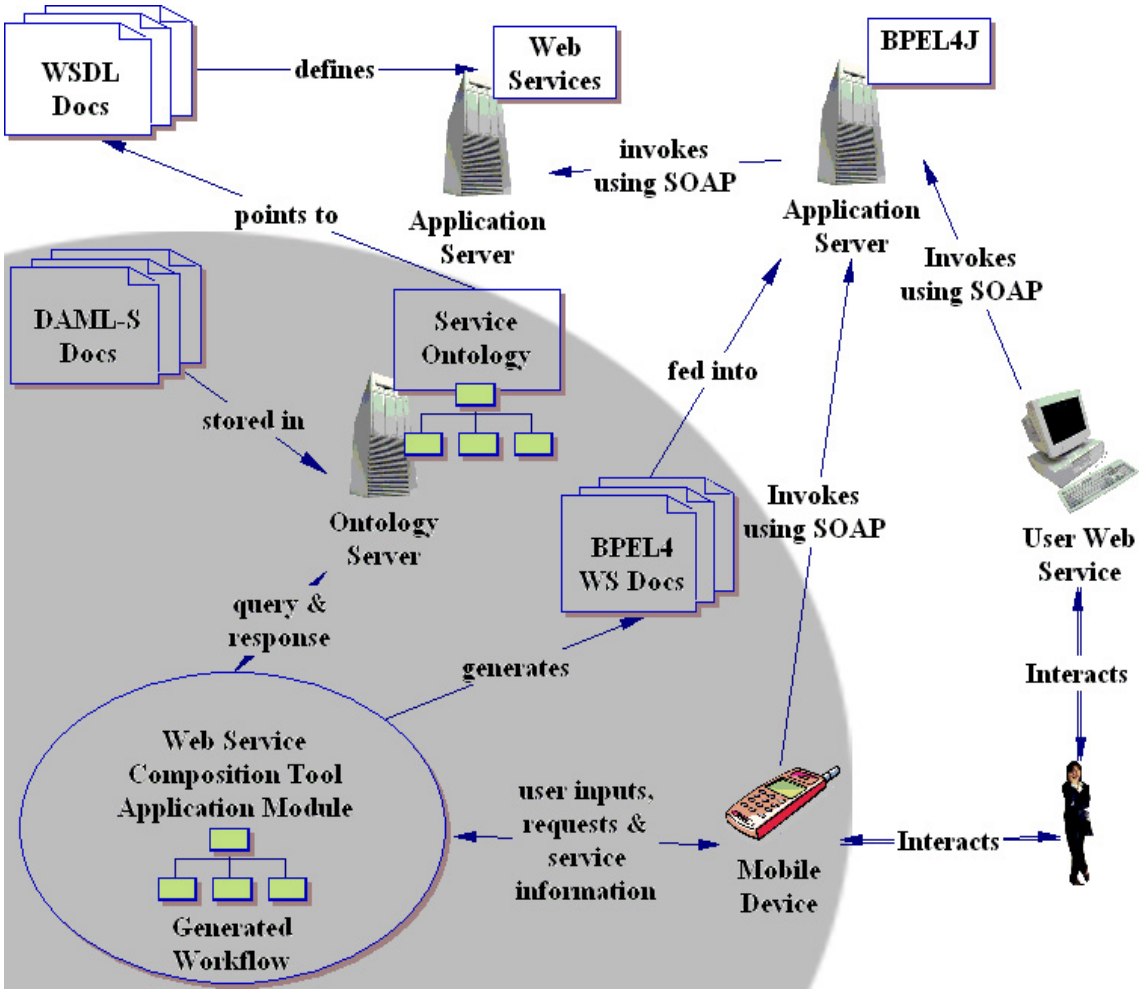

Figure 1.1: Global System View

action; that involves several steps like sale of stocks or bonds, money transfer and payment; and which is executed repeatedly can be defined as a workflow and deployed on a BPEL4J engine. Similarly a workflow for frequent travellers which discovers hotels and flights, reserves rooms and flight tickets, makes payments can be defined so that the user can execute the process with different set of inputs.

In this thesis, BPEL4WS documents that can be deployed on a BPEL engine like BPEL4J are generated. The validity of the documents generated is checked by the BPEL4J Editor developed by IBM alphaWorks [1].

The presentation layer stated above is implemented in a cellular phone to prove that it is possible to perform such complex activities in a limited device. The cellular phone can show the workflow generated through a graphical user interface and let the user navigate on the workflow.

The system developed in this thesis is designed to be a part of a larger system that is presented in Figure 1.1. The part of it that is implemented in this thesis is shown in shaded area.

The thesis is organized as follows: Technologies employed in the thesis are explained in Chapter 2. System Architecture is presented in Chapter 3. Design decisions and system implementation is explained in Chapter 4. The objectives of the three layers are given in Section 4.1. Section 4.2 explains how the DAML-S ontology is processed. Section 4.3 explains the messaging scheme developed. Section 4.4 and 4.5 include the implementation details of the Presentation and Application Layers respectively. The thesis is concluded and future work are given in Chapter 5.

CHAPTER 2

Enabling Technologies

2.1 Web Services

Web Services are self-contained, self-describing, modular applications that can be published, located, and invoked through XML artefacts across the Web [41]. Web services perform functions that can be anything from simple requests to complicated business processes. A sample Web service might provide stock quotes, process credit card transactions, or accept purchase orders automatically. Individual Web Services can be composed to accomplish more complex business processes.

Web services technologies can be applied to many kinds of applications, where they offer considerable advantages compared to the old world of product-specific APIs, platform-specific coding, and other "brittle" technology restrictions [7]:

- Loose coupling – if the message format stays the same, the system won't break just because software on one side or the other changes.
- Ability to use any operating system, any programming language, any vendor's software, any object model – by focussing on the message format and standards for governing the exchange of messages, Web services hides implementation technology choices from partners.

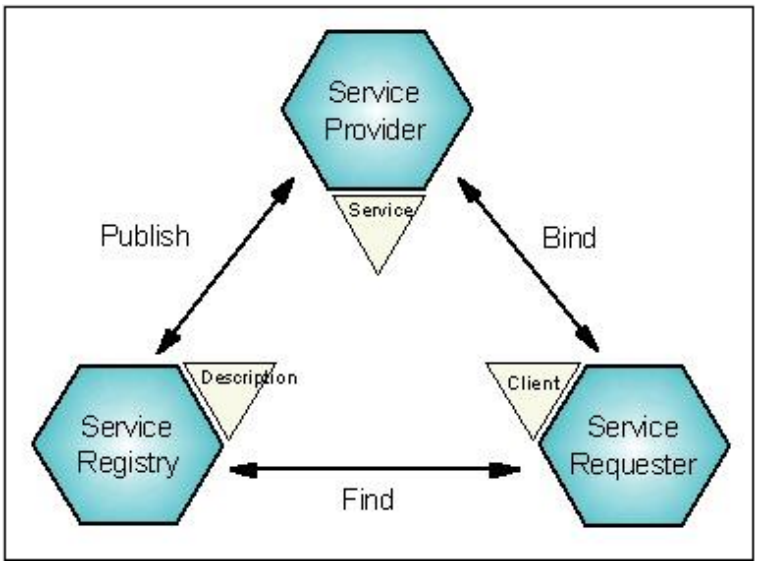

Figure 2.1: Web Service Model

- Late binding – current version of the service can be determined by looking in a registry, further protecting software from inevitable change.
- Reusable code available on the Web with new kinds of functions that integrate real-time information.
- Faster integration by using standard descriptions that application development tools use to assist the programmer with service access requirements or to generate code.

Interactions among Web services involve three types of participants: service provider, service registry and service requester as presented in Figure 2.1. The components in Web Service model can be described as follows [42]:

- *Service:* This is the application being provided for use by requesters that fit the prerequisites specified by the Service Provider. Its implementation is deployed on a network accessible platform. It is described through a service description language mostly in WSDL (Web Services Description Language) [39]. Its description and access policies have been published to a registry.

- *Service Provider:* From a business perspective, this is the owner of the service. From an architectural perspective, this is the platform that provides access to the service.

- *Service Registry:* This is a searchable repository of service descriptions where service providers publish their services and service requesters find services and obtain binding information for services. There are two well known service registries: Electronic Business XML (ebXML) [17] Registries and the The Universal Description, Discovery, Integration framework (UDDI) [36] Registries. Service Providers *advertise* (publish) the availability of their e-business service to one or more service registries, or to remove the advertisement of (unpublish) their service.

- *Service Requestor:* From a business perspective, this is the business that requires certain function to be fulfilled. From an architectural perspective, this is the application or client that is looking for and invoking a service. Service Requestors interact with one or more service registries to *discover* a set of e-business services that it can interact with to provide a solution. They negotiate with Service Providers to *access* and *invoke* e-business services. The universal standard for invoking Web services is SOAP (Simple Object Access Protocol) [34], which is an XML based messaging and remote procedure call (RPC) mechanism.

2.1.1 WSDL

The introduction of Web services has created excitement within many technical circles. This excitement stems from many roots; two of which are the promises of interoperability and speedy time-to-market. Interoperability is partly achieved through the use of common, open protocols like HTTP and SOAP [34]. This is not enough, however, to make the implementation of a server process completely transparent to the client.

The client also has to know about the data types, parameters, return types, location, and transmission details of Web services. There is a need for a meta-protocol that can describe these essentials in a non-vendor and non-implementation specific manner so that the client and server might be completely decoupled [38].

Web Services Description Language (WSDL) [39] enables one to separate the description of the abstract functionality offered by a service from concrete details of a service description such as "how" and "where" that functionality is offered.

WSDL is an XML-based file format that describes not only these Web service interface details but also how the abstract interface is tied to a given transport protocol (HTTP, SMTP, etc.,) and encoding (SOAP, etc.,). The advantages of WSDL, however, are more than just those that come from interoperability. WSDL is a World Wide Web Consortium (W3C) [43] Standard. Being bound to a specification, vendors are able to create tools that use WSDL to generate not only client run-time code for interacting with a Web service but also tools that use WSDL to generate server-side template code.

WSDL, then, not only does help disparate processes interoperate but also helps to lower the development time required to Web service-enable the client and server. The IBM WebSphere SDK for Web Services (WSDK) [37] comes with the *WSDL2WebService* tool that uses WSDL to help the developer get his/her project up and running by generating the client run-time code and server template code for a Web service [38].

WSDL describes Web services starting with the *messages* that are exchanged between the service provider and requestor. The messages themselves are described abstractly and then bound to a concrete network protocol and message format. A message consists of a collection of typed data items. An exchange of messages between the service provider and requestor are described as an *operation*. A collection of operations is called a *portType*. A service contains a collection of ports, where each *port* is an implementation of a *portType*, which includes all the concrete details needed to interact with the service [39].

These constructs are expressed in a WSDL document as follows:

PortTypes: A portType describes the operations provided by a Web service. It is like a Java interface in that it describes a set of operations. It combines message elements into an operation.

Messages and types: A message is a data element. It is used by an operation to carry the data of the operation. Messages describe the communication between client and service, by listing the data types exchanged. The types are described in the types element, which is usually done with XML Schema [45]. Types are like Java classes and primitive types.

Operations, messages, and faults: An operation is like a Java method. It consists of incoming, outgoing, and fault messages. It is possible to consider an incoming message for an operation like a method's parameters in Java programming language. An outgoing message for an operation can be thought like a method's return type in Java programming language. A fault message can be considered as a Java exception.

Bindings: A binding binds a portType to a particular protocol (for example, SOAP 1.1, HTTP GET/POST, or MIME).

Services: A service defines the connection information for a particular binding. Services can have one or more ports, each of which define a different connection method (for example, HTTP / SMTP, etc.).

The <types> WSDL element allows you to specify the data types which are required by the Web service interface a WSDL file describes, no matter how simple or complex they are. The WSDL Specification puts no requirements on what protocol is used to define types although it supports *XML Schema* for its canonical type protocol. XML Schema is both common and open, so its use within WSDL keeps the Web service definition free from the type-specifics of programming languages and vendor-specific types. Following is two type declarations employing XML schema:

```
<wsdl:types>
  <xsd:schema>
    <xsd:simpleType name="EmailAddressType">
```

```
      <xsd:restriction base="xsd:string">
        <xsd:pattern value=".+@.+"/>
      </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="PriceType">
      <xsd:restriction base="xsd:decimal">
        <xsd:totalDigits value="6"/>
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:schema>
</wsdl:types>
```

The <message> element groups data (whose types are defined in the <types> element) into a signature for a logical network transmission and binds them to a name. This name is used to reference a <message> within the context of an operation definition. Each instance of data within a <message> is declared in a <part> element as follows.

```
<wsdl:definitions xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/ ...>
  ...
  <wsdl:message name="Subscribe">
    <wsdl:part name="email" type="tns:EmailAddressType"/>
    <wsdl:part name="until" type="xsd:date"/>
  </wsdl:message>

  <wsdl:message name="PurchaseResponse">
    <wsdl:part name="paymentResponse" type="tns:ResponseConfirmationType" />
    <wsdl:part name="purchaseDate" type="xsd:date" />
  </wsdl:message>
```

The <portType> element is the WSDL equivalent of a Java interface. It groups references to <message> elements into logical operations that a process can execute on another process and binds them to a name. An <operation> can contain <input>, <output>, and <fault> elements. For all three elements the message attribute references a <message> element defined in the WSDL file. The <input> element declares the requirements of a client request transmission to a Web service. The <output> declares the content of a Web service's

response. The <fault> element describes any message-level exceptions that occurred while trying to respond to the client's request.

```
<wsdl:portType name="DVDRenting">
    <wsdl:operation name="SubscribeToSpecialsAlertList">
      <wsdl:input message="tns:Subscribe" />
    </wsdl:operation>

    <wsdl:operation name="RentDVD">
      <wsdl:input message="tns:RentalRequest" />
      <wsdl:output message="tns:PurchaseResponse" />
      <wsdl:fault name="tns:RentFault" message="tns:PaymentFault" />
    </wsdl:operation>
  </wsdl:portType>
```

The WSDL elements listed so far have all been abstract as far as a particular transport or messaging protocol is concerned (that is, SOAP, SMTP, HTTP, etc.) An organization using any protocols could implement the Web service interface described above. The content of a WSDL <binding> element ties down these abstract hooks to a Web protocol. The binding element has both a name attribute to identify it within the WSDL document and a type attribute that references the portType for which this element describes a binding. It also has an <operation> element for every <operation> element in the <portType> for which this is a binding. The <operation> element in turn has <input>/<output>/<fault> elements for those defined in its corresponding <operation> element. Elements describing a binding are nested within these descendents of the <binding> element in order to link messaging protocol specifics to generalities mentioned in the target portType. Following is an example with no messages defined [38].

```
<wsdl:binding name="SoapDVDRenting" type="tns:DVDRenting">
    <wsdl:operation name="tns:SubscribeToSpecialsAlertList">
      <wsdl:input>
      </wsdl:input>
    </wsdl:operation>

    <wsdl:operation name="tns:RentDVD">
      <wsdl:input>
      </wsdl:input>
```

```
    <wsdl:output>
    </wsdl:output>
    <wsdl:fault>
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>
```

The W3C recommends three bindings for Web services: SOAP over HTTP, HTTP GET/POST, SOAP/MIME. A short example for SOAP over HTTP is:

```
<wsdl:binding name="SoapDVDRenting" type="tns:DVDRenting">
  <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="SubscribeToSpecialsAlertList">
    <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body namespace="http://dvd.com" use="literal"/>
      </wsdl:input>
    </wsdl:operation>
```

The <soap:binding> element intimates the messaging style (rpc) of the soap communications (see Messaging Styles) and the intended transport protocol (HTTP) . The soapAction attribute of the <soap:operation> element gets translated into an HTTP header that declares the intention of the HTTP request that will be sent.

Finally, the <service> WSDL element associates a particular binding to one or more processes on the network that can service requests according to the portType that the binding implements. For SOAP over HTTP this is simply a URL pointing to that process [38]. Following is an example service declaration.

```
<wsdl:service name="DVDRentalServices">
  <wsdl:documentation>Here are some endpoints that support the DVDRental SOAP
    HTTPbinding
  </wsdl:documentation>
  <wsdl:port name="LPCDVDRentalService" binding="tns:SoapDVDRenting">
    <soap:address location="http://localhost:6080/soap/servlet/rpcrouter" />
  </wsdl:port>
  <wsdl:port name="ClassicsDVDRentals" binding="tns:SoapDVDRenting">
    <soap:address
location="http://www.classicsdvdrentals.com/soap/servlet/rpcrouter" />
  </wsdl:port>
</wsdl:service>
```

2.2 Semantic Web and Web Service Semantics

The World Wide Web has been made possible through a set of widely established standards which guarantee interoperability at various levels: the TCP/IP protocol has ensured that nobody has to worry about transporting bits over the wire anymore; similarly, HTTP and HTML have provided a standard way of retrieving and presenting hyperlinked text documents. Applications were able to use this common infrastructure and this has led to the WWW as we know it now.

The current Web can be characterised as the second generation Web: the first generation Web was characterised by handwritten HTML pages; the second generation made the step to machine generated and often active HTML pages. These generations of the Web were meant for direct human processing (reading, browsing, form-filling, etc.). The third generation Web aims to make Web resources more readily accessible to automated processes by adding meta-data annotations that describe their content [19]. This coincides with the "Semantic Web" vision of Tim Berners-Lee, which aims to bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users [24].

The WWW has also drastically changed the availability of electronically available information. However, this success and exponential growth makes it increasingly difficult to find, to access, to present, and to maintain the information of use to a wide variety of users. In reaction to this bottleneck many new research initiatives and commercial enterprises have been set up to enrich available information with machine processable semantics. Such support is essential for "bringing the web to its full potential" in areas such as knowledge management and electronic commerce. This semantic web will provide intelligent access to heterogeneous and distributed information enabling software products (agents) to mediate between the user needs and the available information sources [18].

The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation [24].

The first step in this direction is taken by RDF which define a syntactical convention and a simple data model for representing machine-processable semantics of data. The Resource Description Framework (RDF) [32] is a standard for Web meta data developed by the World Wide Web Consortium (W3C) [43]. A second step is taken by the RDF Schema (RDFS) [33] recommendation that defines basic ontological modeling primitives on top of RDF. RDFS in particular is recognisable as an ontology/knowledge representation language: it talks about classes and properties (binary relations), range and domain constraints (on properties), and subclass and subproperty (subsumption) relations.

RDFS is, however, a very primitive language, and more expressive power would clearly be necessary/desirable in order to describe resources in sufficient detail. Moreover, such descriptions should enable automated reasoning if they are to be used effectively by automated processes, e.g., to determine the semantic relationship between syntactically different terms [19].

As a response to these needs, third step is taken by OIL [29] that uses RDFS as a starting point and extends it to a full-fledged ontology modeling language. OIL unifies three important aspects provided by different communities: Rich modeling primitives as provided by the Frame community, formal semantics and efficient reasoning support as provided by Description Logics, and a standard proposal for syntactical exchange notations as provided by the Web community. Another candidate for such a web-based ontology modeling language is DAML-ONT [9] funded by DARPA [8]. In 1999 the DARPA Agent Markup Language (DAML) program was initiated with the aim of providing the foundations of a next generation Semantic Web. As a first step, it was decided that the adoption of a common ontology language would facilitate semantic interoperability across the various projects making up the program. RDFS was seen as a good starting point, and was already a proposed World Wide Web Con-

sortium (W3C) standard, but it was not expressive enough to meet DAML's requirements. A new language called DAML-ONT was therefore developed that extended RDF with language constructors from object-oriented and frame-based knowledge representation languages. Like RDFS, DAML-ONT suffered from a rather weak semantic specification, and it was soon realised that this could lead to disagreements, both amongst humans and machines, as to the precise meaning of terms in a DAML-ONT ontology [18]. The developers of DAML-ONT and OIL have combined their efforts to produce DAML+OIL [12]. The merged language has a formal (model theoretic) semantics that provides machine and human understandability, and a reconciliation of the language constructors from the two languages. The development of DAML+OIL has been undertaken by a committee largely made up of members of the two language design teams titled the Joint EU/US Committee on Agent Markup Languages. DAML+OIL will be presented in detail in Section 2.2.1. Recently World Wide Web Consortium has started the initiative to develop Semantic Web and a semantic markup language for publishing and sharing ontologies, namely Web Ontology Language (OWL) [30]. OWL is derived from DAML+OIL by incorporating learnings from the design and application use of DAML+OIL.

Among the most important Web resources are those that provide services. The Semantic Web should enable greater access not only to content but also to services on the Web. [11]. To exploit the Web Services in their full potential semantics of the Web Services should be defined so that users and software agents should be able to discover, invoke, compose, and monitor Web resources offering particular services and having particular properties automatically [16].

Web services, like their real life counterparts, may have many properties such as:

- The methods of charging and payment.
- The channels by which the service is requested and provided.
- Constraints on temporal and spatial availability.

- Service quality, security, trust and rights attached to a service and many more.

Web Service Description Language (WSDL) specifies only the technical interface of the Web services. To be able to describe the properties of the services, the semantics of the service should be defined in a machine processable and interoperable manner by using ontologies. In other words, all the necessary properties of services can easily be defined through an ontology language and domain specific ontologies can be developed by standard bodies [15].

In this respect, DAML-S [11] is an initiative by BBN Technologies, Carnegie Mellon University, Nokia, Stanford University, and SRI International to provide an ontology, within the framework of the DARPA Agent Markup Language (DAML), for describing Web services. DAML-S defines an upper ontology, that is, a generic "Service" class. In order to make use of DAML-S upper ontology, the lower levels of the ontology need to be defined [15]. DAML-S will be presented in detail in Section 2.2.2.

As the time of writing, DAML-S 0.9 Beta Specification is available which is expected to be the latest release based on DAML+OIL. Subsequent releases will be based upon the Web Ontology Language [30] developed by the Web-Ontology Working Group at the World Wide Web Consortium [10] and be named as OWL-S [35].

Defining languages for the semantic web is just the first step into this direction. Developing new tools, architectures, and applications is the real challenge afterwards. In this respect, this thesis provides an important basis since it covers both an extension to DAML-S Service Ontology and the usage of this semantic web constructs to implement a system architecture that lets semi-automatic web service composition.

2.2.1 DAML+OIL

DAML+OIL is an ontology language, and as such is designed to describe the structure of a domain. DAML+OIL takes an object oriented approach, with the

structure of the domain being described in terms of classes and properties. An ontology consists of a set of axioms that assert, e.g., subsumption relationships between classes or properties. Asserting that resources (pairs of resources) are instances of DAML+OIL classes (properties) is left to RDF, a task for which it is well suited [19]. DAML+OIL divides the universe into two disjoint parts [13]. One part consists of the values that belong to XML Schema datatypes. This part is called the datatype domain. The other part consists of (individual) objects that are considered to be members of classes described within DAML+OIL (or RDF). This part is called the object domain.

DAML+OIL is mostly concerned with the creation of classes that describe (or define) part of the object domain. Such classes are called object classes and are elements of daml:Class, a subclass of rdfs:Class. DAML+OIL also allows the use of XML Schema datatypes to describe (or define) part of the datatype domain. These datatypes are used within DAML+OIL simply by including their URIs within a DAML+OIL ontology. They are (implicitly) elements of daml:Datatype.

Relations between classes are defined with rdf:Property elements. Properties can be either instances of ObjectProperty, which relate objects to other objects; or datatype properties, DatatypeProperty which relate objects to datatype values.

From a formal point of view, DAML+OIL can be seen to be equivalent to a very expressive description logic, with a DAML+OIL ontology corresponding to a DL (Description Logics) terminology. As in a DL, DAML+OIL classes can be names (URIs) or expressions, and a variety of constructors are provided for building class expressions. The expressive power of the language is determined by the class (and property) constructors supported, and by the kinds of axiom supported. The constructors supported by DAML+OIL are as follows:

- intersectionOf
- unionOf

- complementOf
- disjointUnionOf
- oneOf
- toClass
- hasClass
- hasValue
- minCardinalityQ
- maxCardinalityQ
- cardinalityQ

The meaning of the first four constructors (intersectionOf, unionOf and complementOf) is relatively self-explanatory: they are just the standard boolean operators that allow classes to be formed from the intersection, union and negation of other classes. An example Class definition using the "intersectionOf" construct is as follows:

```
<daml:Class rdf:ID="Human">
 <daml:unionOf rdf:parseType="daml:collection">
  <daml:Class rdf:about="#Female"/>
  <daml:Class rdf:about="#Male"/>
 </daml:unionOf>
</daml:Class>
```

The oneOf constructor allows classes to be defined existentially, i.e., by enumerating their members. In DAML it is possible to define property restrictions. The rest of constructors enumerated in the preceding list are used to indicate the type of the restrictions. A property restriction is a special kind of class expression. It implicitly defines an anonymous class, namely the class of all objects that satisfy the restriction. There are two kinds of restrictions. The first kind, ObjectRestriction, works on object properties, i.e., properties that

relate objects to other objects. The second kind, DatatypeRestriction, works on datatype properties, i.e., properties that relate objects to datatype values. Both kinds of restrictions are created using the same syntax, with the usual difference being whether a class element or a datatype reference is used.

- toClass element defines the class of all objects for whom the values of property P all belong to the class expression.
- hasValue element defines the class of all objects for whom the property P has at least one value equal to the named object or datatype value (and perhaps other values as well).
- hasClass element defines the class of all objects for which at least one value of the property P is a member of the class expression or datatype.
- maxCardinality element defines the class of all objects that have at most N distinct values for the property P.
- minCardinality element defines the class of all objects that have at least N distinct values for the property P.
- cardinalityQ element defines the class of all objects that have exactly N distinct values for the property P that are instances of the class expression or datatype (and possibly other values not belonging to the class expression or datatype).

As already mentioned, besides the set of constructors supported, the other aspect of a language that determines its expressive power is the kinds of axiom supported. These axioms make it possible to assert subsumption or equivalence with respect to classes or properties, the disjointness of classes, the equivalence or nonequivalence of individuals (resources), and various properties of properties. The set of axioms supported by DAML+OIL are listed as follows:

- subClassOf
- sameClassAs

- subPropertyOf
- samePropertyAs
- disjointWith
- sameIndividualAs
- differentIndividualFrom
- inverseOf
- transitiveProperty
- uniqueProperty
- unambiguousProperty

2.2.2 DAML-S

DAML-S [11] is an attempt by BBN Technologies, Carnegie Mellon University, Nokia, Stanford University, and SRI International to provide an ontology, within the framework of the DARPA Agent Markup Language, for describing Web services. It enables users and software agents to automatically discover, invoke, compose, and monitor Web resources offering services, under specified constraints. It is an ontology for services written in DAML-the DARPA Agent Markup Language, which enables the creation of ontologies for any domain and the instantiation of these ontologies in the description of specific Web sites.

DAML-S supports both simple and complex services. It aims to manage the following automations:

Automatic Web service discovery: Automatic Web service discovery involves the automatic location of Web services that provide a particular service and that adhere to requested constraints. DAML-S provides declarative advertisements of service properties and capabilities that can be used for automatic service discovery.

Automatic Web service invocation: Automatic Web service invocation involves the automatic execution of an identified Web service by a computer program or agent.

Automatic Web service composition and interoperation: This task involves the automatic selection, composition and interoperation of Web services to perform some task, given a high-level description of an objective.

Automatic Web service execution monitoring: Individual services and, even more, compositions of services will often require some time to execute completely.

DAML-S provides an upper ontology for services. The class Service stands at the top of taxonomy of services, and its properties are the properties normally associated with all kinds of services. The upper ontology for services is silent as to what the particular subclasses of Service should be, or even the conceptual basis for structuring this taxonomy, but it is expected that the taxonomy will be structured according to functional and domain differences and market needs.

The structuring of the ontology of services is motivated by the need to provide three essential types of knowledge about a service as presented in Figure 2.2, each characterized by the question it answers:

- What does the service require from the user(s), or other agents, and provide for them? The answer to this question is given in the "profile". Thus, the class *Service* presents a *ServiceProfile*.

- How does it work? The answer to this question is given in the "model". Thus, the class *Service* is describedBy a *ServiceModel*.

- How is it used? The answer to this question is given in the "grounding". Thus, the class *Service* supports a *ServiceGrounding*.

The properties *presents, describedBy, and supports* are properties of Service. The classes *ServiceProfile, ServiceModel, and ServiceGrounding* are the respective ranges of those properties. The DAML-S definition of the service can be found in [14].

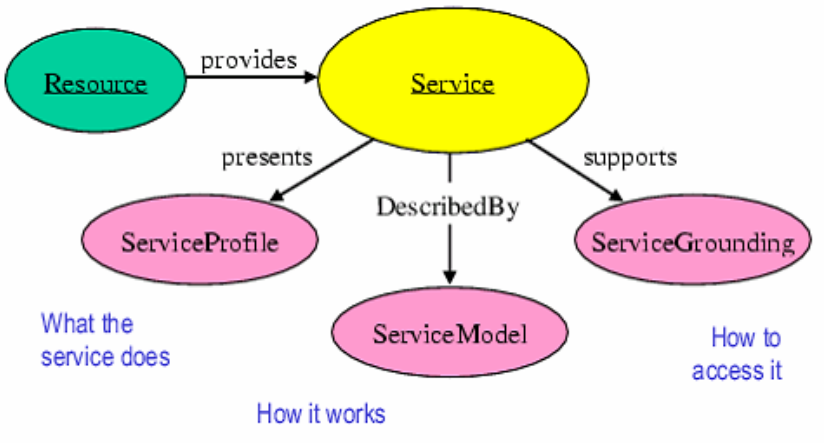

Figure 2.2: Top level of the service ontology

The service profile, tells, "what the service does"; that is, it gives the type of information needed by a service-seeking agent to determine whether the service meets its needs (typically such things as input and output types, preconditions and postconditions, and binding patterns). A service profile provides a high-level description of a service and its provider; it is used to request or advertise services with discovery/location registries. Service profiles consist of three types of information: a human readable description of the service; a specification of the functionalities that are provided by the service; and a host of functional attributes which provide additional information and requirements about the service that assist when reasoning about several services with similar capabilities.

Service functionalities are represented as a transformation from the inputs required by the service to the outputs produced. For example, a news reporting service would advertise itself as a service that, given a date, will return the news reported on that date. Functional attributes specify additional information about the service, such as what guarantees of response time or accuracy it provides, or the cost of the service.

While service providers use the service profile to advertise their services, service requesters use the profile to specify what services they need and what

they expect from such a service. The service profile contains only the information that allows registries to decide which advertisements are matched by a request. To this extent, the information in the profile is a summary of the information in the *process model* and *service grounding*.

In these thesis The DAML-S Service Profile is extended to include additional features. The specifics of this extension is given in Section 4.2, Hence the DAML-S Service Profile is presented in more detail later in this section.

The service model tells, "How the service works"; that is, it describes what happens when the service is carried out. For non-trivial services (those composed of several steps over time), this description may be used by a service-seeking agent in at least four different ways: (1) to perform a more in-depth analysis of whether the service meets its needs; (2) to compose service descriptions from multiple services to perform a specific task; (3) during the course of the service enactment, to coordinate the activities of the different participants; (4) to monitor the execution of the service. For non-trivial services, the first two tasks require a model of action and process; the last two involve, in addition, an execution model.

A service grounding ("grounding" for short) specifies the details of how an agent can access a service. Typically a grounding will specify a communications protocol, message formats, and service-specific details such as port numbers used in contacting the service. In addition, the grounding must specify, for each abstract type specified in the ServiceModel, an unambiguous way of exchanging data elements of that type with the service (that is, the marshaling serialization techniques employed). In DAML-S version 0.9 beta, service grounding is based on WSDL definitions of the services.

Generally speaking, *the ServiceProfile* provides the information needed for an agent to discover a service. Taken together, *the ServiceModel* and *Service-Grounding* objects associated with a service provide enough information for an agent to make use of a service.

DAML-S Service Profile Model

In this section, the details of the fields of the profile model will be presented in four sections: first section describes the properties that link the Service Profile class with the Service class and Process Model class; the second section describes the form of contact information and the Description of the profile this is information usually intended for human consumption; where in the third section , the functional representation and specifically the IOPEs are described; and finally, in the last section, the attributes of the Profile are described.

- *Service Profile:* The class ServiceProfile provides a superclass of every type of high-level description of the service. ServiceProfile does not mandate any representation of services, but it mandates the basic information to link any instance of profile with an instance of service. There is a two-way relation between a service and a profile, so that a service can be related to a profile and a profile to a service. These relations are expressed by the properties *presents* and *presentedBy*. *presents* describes a relation between an instance of service and an instance of profile, it basically says that the service is described by the profile. *presentedBy* is the inverse of *presents*; it specifies that a given profile describes a service.

- *Service Name, Contacts and Description:* Some properties of the profile provide human-readable information that is unlikely to be automatically processed. These properties include *serviceName*, *textDescription* and *contactInformation*. A profile may have at most one service name and text description, but as many items of contact information as the provider wants to offer. *serviceName* refers to the name of the service that is being offered. It can be used as an identifier of the service. *textDescription* provides a brief description of the service. It summarizes what the service offers, it describes what the service requires to work, and it indicates any additional information that the compiler of the profile wants to share with the receivers. *contactInformation* specifies a person or other entity that the provider of the service wants to share with the reader. Each item of

contact information is an instance of the class *Actor* described below.

- *Actor:* The class *Actor* provides information on the provider or the requester of the service; specifically, it provides the following information:

  name: The name property of *Actor* specifies the name of the actor. This could be either a person name or a company name.

  title: Title of the contact, a CEO, or Service Department or whatever is deemed appropriate.

  phone: A phone number that can be used to gather information on the service.

  fax: A fax number that can be used to gather information on the service.

  email: An e-mail address that can be used to gather information on the service.

  physicalAddress: A physical address that can be used to gather information on the service.

  webURL: A URL of the product or company Website.

- *Functionality Description:* An essential component of the profile is the specification of what functionality the service provides and the specification of the conditions that must be satisfied for a successful result. In addition, the profile specifies what conditions result from the service, including the expected and unexpected results of the service activity. The DAML-S Profile represents two aspects of the functionality of the service: the information transformation and the state change produced by the execution of the service. For example, to complete the sale, a book-selling service requires as input a credit card number and expiration date, but also the precondition that the credit card actually exists and is not overdrawn. The result of the sale is the output of a receipt that confirms the proper execution of the transaction, and as effect the transfer of ownership

and the physical transfer of the book from the warehouse of the seller to the address of the buyer. The information transformation produced by the service is represented by input and output properties of the profile. The *input* property specifies the information that the service requires to proceed with the computation. For example, a book-selling service could require the credit-card number and bibliographical information of the book to sell. The outputs specify what is the result of the operation of the service. For the book-selling agent the output could be a receipt that acknowledges the sale. The state change produced by the execution of the service is specified through the *precondition* and *effect* properties of the profile. *Precondition* presents logical conditions that should be satisfied prior to the service being requested. These conditions should have associated explicit *effects* that may occur as a result of the service being performed. Effects are the result of the successful execution of a service. The representation of preconditions and effects depends on the representation of rules in the DAML language. Currently, a working group is trying to specify rules in DAML, but no proposal has been put forward. For this reason, the fields *precondition* and *effect* are mapped to thing meaning that anything is possible, but this will have to be modified in future releases of the profile.

input: specifies one of the inputs of the service. It takes as value an instance of *ParameterDescription* (see below) that specifies an *id* of the input, a *value* and a *reference* to the corresponding *input* in the process model. The value of the property is an instance of *ParameterDescription* described below.

output: specifies one of the outputs of the service. It takes as value an instance of *ParameterDescription* (see below) that specifies an *id* of the output, a *value* and a *reference* to the corresponding *output* in the process model. The value of the property is an instance of *ParameterDescription* described below.

precondition: specifies one of the preconditions of the service. It takes as value an instance of *ParameterDescription* that specifies an *id* of the precondition, a *value* and a *reference* to the corresponding *precondition* in the process model. The value of the property is an instance of *Parameter-Description* described below.

effect: specifies one of the effects of the service. It takes as value an instance of *ParameterDescription* that specifies an *id* of the effect, a *reference* and a *reference* to the corresponding *effect* in the process model. The value of the property is an instance of *ParameterDescription* described below.

ParameterDescription: The class *ParameterDescription* provides values to inputs and outputs. It collects in one class the name of the input or *output* that can be used as an identifier, its *value* and a *reference* to the corresponding *input* or *output* in the process model.

parameterName: provides the name of the input or output, which could be just a literal, or perhaps the URI of the process parameter (a property).

restrictedTo: provides a restriction on the values of the input or output.

refersTo: provides a reference to the input or output in the process model.

- *Profile Attributes:* In the previous section we introduced the functional description of services, but there are other aspects of services of which users should be aware. These additional attributes include the quality guarantees that are provided by the service, possible classification of the service, and additional parameters that the service may want to specify.

  serviceParameter: is an expandable list of properties that may accompany a profile description. The value of the property is an instance of the class *ServiceParameter*.

serviceCategory: refers to an entry in some ontology or taxonomy of services. The value of the property is an instance of the class *ServiceCategory.*

QualityRating: is used to specify the rating of a service using some rating system. The rating of a service provides the potential client with information about the quality of the service provided.

- *ServiceParameter:*

  serviceParameterName: is the name of the actual parameter, which could be just a literal, or perhaps the URI of the process parameter (a property).

  sParameter: points to the value of the parameter within some DAML ontology.

- *QualityRating:*

  ratingName: points to the name of the rating service.

  rating: stores the value of the rating within a given rating service.

- *ServiceCategory:*

  ServiceCategory: describes categories of services on the basis of some classification that may be outside DAML-S and possibly outside DAML. In the latter case, they may require some specialized reasoner if any inference has to be done with it.

  categoryName: is the name of the actual category, which could be just a literal, or perhaps the URI of the process parameter (a property).

  taxonomy: stores a reference to the taxonomy scheme. It can be either a URI of the taxonomy, or a URL where the taxonomy resides, or the name of the taxonomy or anything else.

  value: points to the value in a specific taxonomy There may be more than one value for each taxonomy, so no restriction is added here.

codeTo: each type of service stores the code associated to a taxonomy.

2.3 BPEL4WS

The goal of the Web Services effort is to achieve universal interoperability between applications by using Web standards. Web Services use a loosely coupled integration model to allow flexible integration of heterogeneous systems in variety of domains including business-to-consumer, business-to-business and enterprise application integration. Currently specifications such as WSDL, DAML-S, Universal Description, Discovery and Integration of Web Services (UDDI) [36] or ebXML [17] and SOAP define the web service space.

Systems integration requires more than the ability to conduct simple interactions by using standard protocols. The full potential of Web Services as an integration platform will be achieved only when applications and business processes are able to integrate their complex interactions by using a standard process integration model.

Models for business interactions typically assume sequences of peer-to-peer message exchanges, both synchronous and asynchronous, within stateful, long-running interactions involving two or more parties. To define such business interactions, a formal description of the message exchange protocols used by business processes in their interactions is needed. The definition of such business protocols involves precisely specifying the mutually visible message exchange behavior of each of the parties involved in the protocol, without revealing their internal implementation. There are two good reasons to separate the public aspects of business process behavior from internal or private aspects. One is that businesses obviously do not want to reveal all their internal decision making and data management to their business partners. The other is that, even where this is not the case, separating public from private process provides the freedom to change private aspects of the process implementation without affecting the public business protocol.

Business protocols must clearly be described in a platform-independent manner and must capture all behavioral aspects that have cross-enterprise business significance. Each participant can then understand and plan for conformance to the business protocol without engaging in the process of human agreement that adds so much to the difficulty of establishing cross-enterprise automated business processes today [3].

Defining business protocols and defining executable business processes require very similar concepts. The concepts required for defining business protocols and those required for defining executable business processes form a continuum, and Business Process Execution Language for Web Services (BPEL4WS) [3] is designed to cover this continuum. BPEL4WS defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners. The interaction with each partner occurs through Web Service interfaces, and the structure of the relationship at the interface level is encapsulated in a service link. The BPEL4WS process defines how multiple service interactions with these partners are coordinated to achieve a business goal, as well as the state and the logic necessary for this coordination. BPEL4WS also introduces systematic mechanisms for dealing with business exceptions and processing faults. Finally, BPEL4WS introduces a mechanism to define how individual or composite activities within a process are to be compensated in cases where exceptions occur or a partner requests reversal.

BPEL4WS is layered on top of several XML specifications: WSDL 1.1, XML Schema 1.0, and XPath 1.0. WSDL messages and XML Schema type definitions provide the data model used by BPEL4WS processes. XPath provides support for data manipulation. All external resources and partners are represented as WSDL services. BPEL4WS provides extensibility to accommodate future versions of these standards, specifically the XPath and related standards used in XML computation [2].

BPEL4WS specification is positioned to become the Web services standard for composition. It allows developers to create complex processes by creating

and wiring together different activities that can, for example, perform Web services invocations, manipulate data, throw faults, or terminate a process. These activities may be nested within structured activities that define how they may be run, such as in sequence, or in parallel, or depending on certain conditions [4].

BPEL4WS represents the merging of IBM's Web Services Flow Language (WSFL) [40] and Microsoft's XLANG [44]. BPEL4WS combines the best of both WSFL (support for graph oriented processes) and XLANG (structural constructs for processes) into one cohesive package that supports the implementation of any kind of business process in a very natural manner. In addition to being an implementation language, BPEL4WS can be used to describe the interfaces of business processes as well – using the notion of abstract processes.

As an executable process implementation language, the role of BPEL4WS is to define a new Web service by composing a set of existing services. Thus, BPEL4WS is basically a language to implement such a composition. The interface of the composite service is described as a collection of WSDL portTypes, just like any other Web service.

The BPEL4WS process itself is basically a flowchart like expression of an algorithm. Each step in the process is called an activity. There are a collection of primitive activities: invoking an operation on some Web service (<invoke>), waiting for a message to operation of the service's interface to be invoked by someone externally (<receive>), generating the response of an input/output operation (<reply>), waiting for some time (<wait>), copying data from one place to another (<assign>), indicating that something went wrong (<throw>), terminating the entire service instance (<terminate>), or doing nothing (<empty>).

These primitive activities can be combined into more complex structures using any of the structure activities provided in the language. These are the ability to define an ordered sequence of steps (<sequence>), the ability to have branching using the now common "case-statement" approach (<switch>), the ability to define a loop (<while>), the ability to execute one of several alternative

paths (<pick>), and finally the ability to indicate that a collection of steps should be executed in parallel (<flow>). Within activities executing in parallel, one can indicate execution order constraints by using the links.

BPEL4WS allows the structured activities to be recursively combined to express arbitrarily complex algorithms that represent the implementation of the service.

As a language for composing together a set of services into a new service, BPEL4WS processes mainly consist of making invocations to other services and/or receiving invocations from clients. BPEL4WS calls these other services that interact with a process, partner.

BPEL4WS uses service link types to define partners. Basically, a partner is defined by giving it a name and then indicating the name of a service link type and identifying the role that the process will play from that service link type and the role that the partner will play. In order for it to work at runtime, the partner must resolve to an actual Web service. Thus, a partner is really eventually just a typed service reference, where the typing comes from the service link type and the roles. The BPEL4WS process itself does not indicate how a partner is bound to a specific service; that is considered a deployment time or runtime binding step that must be supported by the BPEL4WS implementation.

Developers need ways to handle and recover from errors in business processes. BPEL4WS has exceptions (faults) built into the language via the <throw> and <catch> constructs. The fault concept on BPEL4WS is directly related to the fault concept on WSDL and in fact builds on it.

In addition, BPEL4WS supports the notion of compensation, which is a technique for allowing the process designer to implement compensating actions for certain irreversible actions. For example, imagine a travel reservation process. Once a reservation has been confirmed, one must perform some explicit operations to cancel that reservation. Those actions are called "compensating actions" for the original action.

Fault handling and compensating is supported recursively in BPEL4WS by introducing the notion of a scope, which is essentially the unit of fault handling and/or compensation.

Following is a simple BPEL4WS document example. The document needs WSDL documents to reflect the process completely and executed.

```
<process name="loanApprovalProcess"
    targetNamespace="http://acme.com/simpleloanprocessing"
    xmlns="http://schemas.xmlsoap.org/ws/2002/07/business-process/"
    xmlns:lns="http://loans.org/wsdl/loan-approval"
    xmlns:loandef="http://tempuri.org/services/loandefinitions"
    xmlns:apns="http://tempuri.org/services/loanapprover">
```

BPEL4WS documents rely heavily on WSDL documents in order to refer to the messages being exchanged, the operations being invoked, and the *portTypes* these operations belong to. Above are the definitions of the namespaces that will allow the BPEL4WS document to refer to the required WSDL information.

```
<partnerLinks>
    <partner name="customer" partnerLinkType="lns:loanApprovalLinkType"
      myRole="approver"/>
    <partner name="approver" partnerLinkType="lns:loanApprovalLinkType"
      partnerRole="approver"/>
  </partnerLinks>
```

The next step is to declare the parties involved. Named partners are defined, each characterized by a WSDL *partnerLinkType*. *partnerLinkType*s are defined in respective WSDL documents.

```
  <variables>
    <variable name="request" messageType="loandef:CreditInformationMessage"/>
    <variable name="approvalInfo"  messageType="apns:approvalMessage"/>
  </variables>
```

Incoming messages should be stored where the BPEL activity can access it. In BPEL, data is written to and accessed from data variables which can hold instances of specific WSDL message types.

```
  <sequence>
    <receive name="receive1"
      partnerLink="customer" portType="apns:loanApprovalPT"
      operation="approve"
      variable="request"
      createInstance="yes">
    </receive>
    <invoke name="invokeapprover"
      partnerLink="approver"
      portType="apns:loanApprovalPT"
      operation="approve"
      inputVariable="request"
      outputVariable="approvalInfo">
    </invoke>
    <reply name="reply"
      partnerLink="customer"
      portType="apns:loanApprovalPT"
      operation="approve"
      variable="approvalInfo">
    </reply>
  </sequence>
</process>
```

The BPEL process is defined as a sequence in the example, consisting of simple receive, invoke and reply activities. The simple activities will be executed in sequence. When the process is deployed into a supporting engine, it will wait until somebody starts it through the receive activity.

IBM's alphaWorks made a BPEL4WS engine publicly available. The IBM Business Process Execution Language for Web Services Java Run Time (BPWS4J) [1] includes the following: a platform upon which business processes written using the BPEL4WS can be executed; a set of samples demonstrating the use of BPEL4WS; and a tool that validates BPEL4WS documents.

A workflow is created with the assumption that it will be executed repeatedly. A single-use workflow generation is not a feasible effort. Since the workflow resides in the engine for a period of time, it can be executed several times. The workflows can be executed with the same configuration, or with a new set of inputs every time it is executed. A web interface can be configured to take the new set of inputs. The process may also be configured as a web service. In this

case, protocols such as SOAP can be used to invoke the BPEL4WS workflow that has been composed and deployed on BPEL4J. A complex banking transaction; that involves several steps like sale of stocks or bonds, money transfer and payment; and which is executed repeatedly can be defined as a workflow and deployed on a BPEL4J engine. Similarly a workflow for frequent travellers which discovers hotels and flights, reserves rooms and flight tickets, makes payments can be defined so that the user can execute the process with different set of inputs.

2.4 J2ME

The ability to program custom applications that can work on limited devices like PDAs and cellular phones offers great potential. The increase in computing power, memory and display opportunities, combined with the ever increasing data communication bandwidth is the main reason behind mobile computing. It is estimated that over 20 million Java 2 Micro Edition (J2ME) enabled phones are manufactured in 2001, Japan only. [26]

J2ME is the final addition to Java 2 family with Java 2 Standard Edition (J2SE) and Java 2 Enterprise Edition (J2EE). J2ME is aimed at devices with low computing power; memory, display, bandwidth and power resources. J2SE support would be acceptable; however with the entire classes approaching a size of 1+ Megabytes, this is not feasible. A Java Virtual Machine (JVM) with a smaller footprint is needed.

However these devices vary greatly. It is not wise to compare a state-of-the-art PDA with a 400 MHz processor to a J2ME enabled cellular phone, although they are both supported by J2ME. It is obvious from the discussion that a single Java platform will not be able to cover the broad range of devices. Two concepts are introduced at this point in J2ME: configurations and profiles.

Configuration:

A configuration defines a Java platform for a broad range of devices. In fact a configuration defines the Java language features and the core Java libraries

of the JVM for that particular configuration. What a configuration applies is most of the time based on memory, display, network connectivity and processing power available.

Currently there are two configurations defined: Connected Device Configuration (CDC) [5] and Connected, Limited Device Configuration (CLDC) [6]. CDC applies to devices with:

- 512 Kb minimum memory for running Java
- 256 Kb minimum for runtime memory allocation
- Network connectivity, possibly persistent and high bandwidth

CLDC applies to devices with:

- 128 Kb memory for running Java
- 32 Kb memory for runtime memory allocation
- Restricted user interface
- Low power, typically battery powered
- Network connectivity, typically wireless, with low bandwidth and intermittent access

It should be noted that, as the technology progresses the overlap between the configurations will increase and the boundary between them will be less significant. Although devices can be classified as being a member of one configuration or the other, there is still the fact that a scarce resource for one device can be abundant to another. To provide for more flexibility Sun introduced the concept of a profile to the J2ME platform.

Profiles:

A profile is an extension to a configuration which provides libraries for a particular type of device. For example, the Mobile Information Device Profile

(MIDP) [25] defines APIs for user interface components, input and event handling, persistent storage, networking and timers, taking into the consideration the screen and memory limitations of mobile devices.

The configurations are developed by open industry working groups utilizing Sun's Java Community Process Program [20]. This way industries can decide for themselves what elements are necessary to provide a complete solution targeted at their industry [22].

Java Virtual Machines:

The engine behind any Java Platform is the JVM. The JVM is responsible for changing the compiled byte code into machine code. Also providing security, allocating and freeing memory and managing threads of execution are responsibilities of the JVM. The JVM for CDC has the same specification as J2SE, however for CLDC Sun developed a reference implementation of a smaller JVM, namely KVM (where K is from Kilobyte) which:

- Requires 40-80 kilobytes of memory
- Requires ony 20-40 kilobytes of dynamic memory (heap)
- Can run on 16-bit processors clocked at 25 Mhz.

In this thesis, an application for a J2ME enabled cellular phone is developed using the CLDC and MIDP. The generic architecture of J2ME and the specific architecture used in this thesis is given in Figure 2.3.

CLDC provides an assortment of vital classes from the J2SE including:

- System Classes in java.lang package: Class, Object, Runnable, Runtime, String, StringBuffer, System, Thread and Throwable
- Data Type Classes in java.lang package: Boolean, Byte, Character, Integer, Long, Short
- Collection Classes in java.util package: Enumeration, Hashtable, Stack, Vector

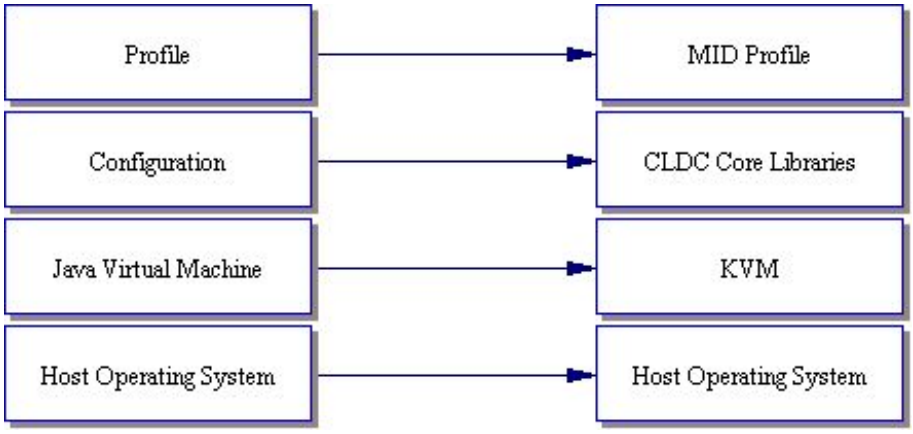

Figure 2.3: Generic and implementation specific J2ME architecture

- Input/output Classes in java.io.package.
- Calendar and Time Classes in java.util package: Calendar, Date, TimeZone
- Utility Classes: java.lang.Math, java.util.Random
- Exception Classes
- Error Classes
- Internalization classes

CLDC also includes classes specific to itself that are used to access storage and network systems. MIDP provides an API for specific devices that helps building user interfaces on small displays, support for HTTP Connection etc.

CHAPTER 3

System Architecture

The World Wide Web is developed for human use. Hyper Text Markup Language (HTML), one of the building blocks of the web is obviously designed for the humans; to format the text such that it is easier and more interesting to read.

As the web became a widely used communication medium, unstructured scatter of data and information in large amounts made discovery of information harder. Search engines appeared on the web that searched and cached every web page to create a structure.

This increase in amount of data led to two things. The first one is the concern for metadata, data about data, thus the appearance of XML. Second one is the deployment of web services previously published on dynamic web pages. Web Services are self-contained, self-describing, modular applications that can be published, located, and invoked through XML artefacts across the Web. Web services technologies can be applied to many kinds of applications, where they offer considerable advantages compared to the old world of product-specific APIs, platform-specific coding, and other "brittle" technology restrictions. XML and XML-based languages like RDF let the metadata to be described in an elegant way. Data can be published in a structured manner. Whereas technologies like WSDL and SOAP enabled platform independant definitions of web services so that web services were deployed and worked globally.

Currently there are millions of web services available on the web due to the increase in e-commerce business volume. The service information is stored in public registries so they are easy to find. Service interfaces are described by WSDL documents and service requesters use this information to configure their own programs that will use the service.

Web services can be discovered using public registries and invoked through respective interfaces. However to *automatically* find, compose, invoke and monitor the web services is still an issue. The automatic discovery, composition, invocation and monitoring of web services require that semantics are attached to service definitions. Computers can understand what the service is about in a consistent way with semantics through ontologies. This enables the automatic processing of web services.

The focus of this thesis is on the composition of web services. Currently the web service definitions are described in DAML-S ontology. DAML-S ontology define the service properties including inputs and outputs. One of the several approaches to the composition of web services is based on the fact that outputs of a given web service would provide a perfect or an approximate match to the inputs of another service, thus this service may succeed the current one [31]. The methodology is to extract the input and output properties of services from a DAML-S ontology and match them to come up with a sequence of services or a workflow.

The approach taken in this thesis is to let the service providers decide the possible succeeding services.A high-level service ontology is defined and stored in a DAML-S document which includes all the service types and their properties. There is no property defined for the service providers to state a succeeding service as the time of writing in DAML-S specification. In this thesis, the service ontology is extended to include the possible alternatives as the next service.

With this architecture the publishers of the service can decide whether a generic service and/or a specific (promoted) service follows their service. The provider can choose to state a generic service as succeeding in case it is known

that a service type is applicable as succeeding service to his service. This type of succeeding services are called Succeeding Generic Services (SGS). From a business view, there may also be cases where the provider would like to promote a specific service instance; a service that is known to be a perfect match; a service that belongs to the same enterprise; or a service for which an agreement between two organizations exists. This type of succeeding services are called Succeeding Service Instances (SSI). An example may be a book selling service that has an agreement with a postal service. The user is able to choose from alternatives and construct a workflow of services that meets her needs. As the services are chosen by the user, inputs for the services are requested from the user and stored.

The services, their properties and succeeding service alternatives are taken from a DAML-S ontology. The ontology is parsed and the service definitions are stored in a format that is easily accessible, like a registry or a relational database.

Service interface definitions are taken from the WSDL documents that belong to each service. This information is combined with the user's choices to build a composition of web services and service definitions from the DAML-S ontology into a flow. The flow at the end, is represented both graphically and textually. Textual representation is in the form of a BPEL4WS document, which employs a language that is XML-based and used to represent web service compositions. It is made up of simple activities (invoke, reply, receive etc.) combined into more complex activities representing sequences, concurrent flows, loops and conditional branches.

A BPEL4WS document can be fed into a BPEL4WS engine along with the respective WSDL documents to automatically execute the workflow. An alpha version BPEL4WS engine is available from IBM's "alphaWorks Emerging Technologies" [1].

A workflow is created with the assumption that it will be executed repeatedly. A single-use workflow generation is not a feasible effort. Since the workflow

resides in the engine for a period of time, it can be executed several times. The workflows can be executed with the same configuration, or with a new set of inputs every time it is executed. A web interface can be configured to take the new sset of inputs. The process may also be configured as a web service. In this case, protocols such as SOAP can be used to invoke the BPEL4WS workflow that has been composed and deployed on BPEL4J. A complex banking transaction; that involves several steps like sale of stocks or bonds, money transfer and payment; and which is executed repeatedly can be defined as a workflow and deployed on a BPEL4J engine. Similarly a workflow for frequent travellers which discovers hotels and flights, reserves rooms and flight tickets, makes payments can be defined so that the user can execute the process with different set of inputs.

In this thesis, BPEL4WS documents that can be deployed on a BPEL engine like BPEL4J are generated. The validity of the documents generated is checked by the BPEL4J Editor developed by IBM alphaWorks [1].

Time is an important asset for the people of the 21st century. Almost all technological research in IS/IT sector aims to shorten the time required to complete a certain activity (e.g. information access time, data gathering time, data processing time). As the technology lets computers to be smaller while more powerful and nomadic through wireless high bandwidth communication technologies, people can carry their computers with themselves. This provides the information accessible at any time to the user, thus the term mobile computing. Ranging from workstation replacement to ultra-light notebooks, Tablet PCs and PDAs; currently cellular phones with programmable memories and adequate processing power offer the extreme mobility. Integrated communication technologies like WAP, GPRS and UMTS; along with platform independency of Java makes information available anytime, anywhere.

It is the aim of this thesis to prove that even in the extreme mobility of state-of-the-art cellular phones it is possible to develop a web service composition utility with a Graphical User Interface (GUI). Information for alternative

services is presented to the user. As the user chooses some service, inputs of the service is requested from her. The user then chooses the workflow construct which identifies whether the service will be followed by concurrent execution, a sequence, a loop or a branch of services. At every step the workflow is shown on the screen of the mobile phone so that the user can follow her choices. The user can navigate on the flow and get information for any service in the flow.

The utility developed is not a device dependent solution. It is compatible with widely accepted standards through Java 2 Micro Edition (J2ME) and can work on many devices from different vendors through Connected Limited Device Configuration (CLDC) and Mobile Information Device Profile (MIDP) specifications.

The final system has a three-layered architecture as shown in the Figure 3.1. The top layer is the presentation layer where in this thesis a mobile device is chosen. A standard PC, a PDA or any kind of device can be used. This layer interacts with the user to present the results of her choice and take inputs from her to build a workflow. The middle layer is the application layer where data is gathered and the resulting workflow is produced in the form of a BPEL4WS file. The implementation can be in the form of a *servlet* residing on an application server or a standard application. In this thesis a standard Java application is developed. The bottom layer is the data layer where service ontologies, individual service definitions and service interface definitions are stored. The implementation can be done using plain DAML-S and WSDL documents; a relational database can be used; or a public registry like ebXML or UDDI can be used provided that the ontology constructs are successfully integrated into the registries as in [23]. In this thesis the DAML-S ontology is parsed and results are placed in a relational database.

The contributions of this thesis are as follows:

- A DAML-S extension including SGS and SSI as *Service Properties*.

- Service definitions in DAML-S document.

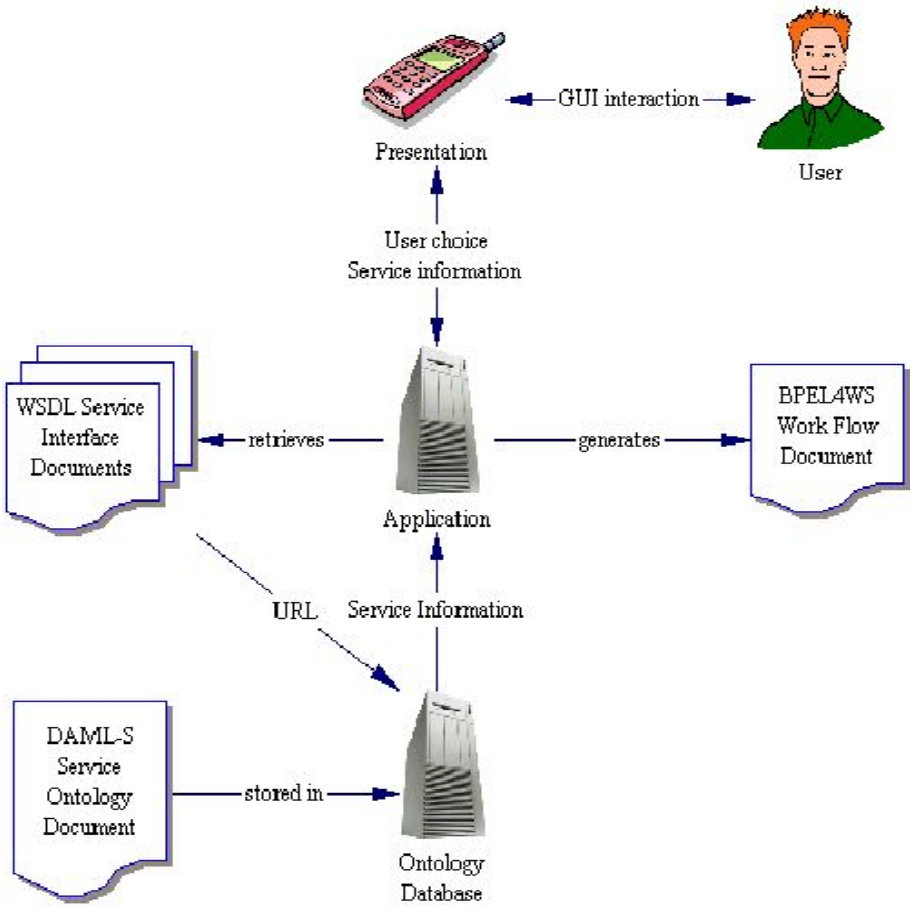

Figure 3.1: The Overall System Architecture

- Service interface specifications in WSDL format.

- A mobile application written in J2ME, in accordance with the CLDC and MIDP specifications. This application presents the service information to the user and takes user choices. It presents the resulting workflow through a GUI. The mobile application uses standard socket connection.

- An application that connects the mobile client to the service ontology. It parses the service definitions into a relational database. It queries this ontology database with respect to the user choices to combine that service ontology and service interface data to come up with a BPEL4WS document representation of the workflow produced, which is executable on a certain

engine.

- A messaging scheme that effectively connects the three layers.

CHAPTER 4

Design and Implementation

As stated in the Chapter 3, the system is implemented in three layers which are treated as three modules. These three modules are:

- Presentation (Mobile) Module
- Application Module
- Data Module

The design and implementation details of each layer are explained in detail in Sections 4.1 through 4.5.

4.1 Module Objectives

Presentation (Mobile) Module's main objective is to interact with the user. The others are to:

- Get user choices for services in the flow.
- Get service inputs from the user.
- Get workflow constructs from the user.
- Show the workflow graphically.

- Let the user navigate on the workflow and get information.

Application Module's main objective is to act like the bridge between the Mobile Module and the Ontology Database. It compiles user and service information to build a textual representation of the workflow as a BPEL4WS file. The other objectives are to:

- Store the DAML-S ontology in the database.
- Retrieve information about the services from the database and the corresponding WSDL files.
- Store user inputs.
- Construct a BPEL4WS representation of the workflow.

Data Module's main objective is to store service ontologies and return appropriate result sets to application module.

4.2 Ontology Processing

The details of the Mobile Module and the Application Module are available in Section 4.4 and Section 4.5 respectively. In this section the details of the ontology and how it is mapped to a relational database is given.

DAML-S service profile class is extended to include a succeding generic service class. A succeeding generic service of a service is also an instance of a profile class and therefore domain and range values are as follows:

```
<daml:ObjectProperty rdf:ID="succeedingGenericService">
  <daml:domain rdf:resource="#Profile" />
  <daml:range rdf:resource="#Profile" />
</daml:ObjectProperty>
```

Each DAML-S profile should have at least one succeeding generic service. If no succeeding generic service is applicable, a special generic service named "nullService" should be defined. Thus the cardinality constraint:

```
<daml:Class rdf:about="#Profile">
  <rdfs:subClassOf>
    <daml:Restriction daml:minCardinality="1">
      <daml:onProperty rdf:resource="#succedingGenericServices" />
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
```

Also the succeeding service instances are defined for each DAML-S service profile. There is no cardinality constraint for the following definition.

```
<daml:ObjectProperty rdf:ID="succeedingServiceInstance">
  <daml:domain rdf:resource="Profile" />
  <daml:range rdf:resource="Profile" />
</daml:ObjectProperty>
```

A complete DAML-S extended service profile definition (Profile.daml) is given in Appendix C.

A mini-ontology for tourism services is developed for this thesis using DAML-S specification. The namespace and class declarations are as follows:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!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 daml "http://www.daml.org/2001/03/daml+oil">
  <!ENTITY xsd "http://www.w3.org/2000/10/XMLschema.xsd">
  <!ENTITY time "http://www.ai.sri.com/daml/ontologies/time/Time.daml">
  <!ENTITY profile "http://www.daml.org/services/daml-s/0.7/Profile.daml">
  <!ENTITY DEFAULT "http://www.daml.org/services/damls/0.7/ProfileHierarchy.daml">
]>
<rdf:RDF
  xmlns:rdf= "&rdf;#"
  xmlns:rdfs= "&rdfs;#"
  xmlns:daml= "&daml;#"
  xmlns:xsd=  "&xsd;#"
  xmlns:profile= "&profile;#"
  xmlns= "&DEFAULT;#">

  <daml:Ontology>
    <daml:versionInfo>
      $MiniTravelOntology.daml v0.01 by K.Alpay Erturkmen $
    </daml:versionInfo>
```

```
    <daml:comment>
      A sample ontology developed by K.Alpay Erturkmen
    </daml:comment>
    <daml:imports rdf:resource="&rdf;" />
    <daml:imports rdf:resource="&daml;" />
    <daml:imports rdf:resource="&profile;" />
  </daml:Ontology>
```

The following is a generic "Hotel Reservation Service" declaration. Notice that the restriction on the *succedingGenericService* property. The restriction makes sure that the succeding generic service of any "Hotel Reservation Service" is only a "Payment Service". "Payment Service" should also be available in the same document or a different document should be defined including the definition.

```
<daml:Class rdf:ID="HotelReservationService">
    <rdfs:subClassOf rdf:resource="&profile;#Profile" />
    <daml:comment>
Generic Hotel Reservation Service
    </daml:comment>
    <rdfs:subClassOf>
      <daml:Restriction>
        <daml:onProperty rdf:resource="&profile;#succedingGenericService" />
        <daml:hasValue rdf:resource="#PaymentService" />
      </daml:Resctriction>
    </rdfs:subClassOf>
  </daml:Class>
```

The following is a property definition. This definition restricts the "HotelArrivalDate" to be an input of generic "Hotel Reservation Service" and its type to be "Arrival Date".

```
  <daml:DatatypeProperty rdf:ID="hotelArrivalDate">
    <daml:subPropertyOf rdf:resource="&profile;#input" />
    <daml:domain rdf:resource="#HotelReservationService" />
    <daml:range rdf:resource="#ArrivalDate"/>
  </daml:DatatypeProperty>
```

In the above example notice that the range of the property is defined as "ArrivalDate" which is a sub-class of a super parameter definition as follows:

```
  <daml:class rdf:ID="ArrivalDate">
    <daml:subClassOf rdf:resource="#StringParameter" />
  </daml:class>
```

The *StringParameter* super parameter is defined as follows:

```
  <daml:class rdf:ID="StringParameter">
  </daml:class>
  <daml:DatatypeProperty rdf:ID="value">
    <daml:domain rdf:resource="#StringParameter" />
    <daml:range rdf:resource="&xsd;#String" />
  </daml:DatatypeProperty>
```

The complete DAML-S tourism mini-ontology developed for the thesis is available in Appendix D.

After the DAML-S profile class is extended and a tourism mini-ontology is developed, individual service's DAML-S definitions can be prepared. Following is the DAML-S definition for an instance of "Hotel Reservation Service":

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!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 daml "http://www.daml.org/2001/03/daml+oil">
  <!ENTITY service "http://www.daml.org/services/daml-s/0.9/Service.daml">
  <!ENTITY profile "http://www.daml.org/services/daml-s/0.9/Profile.daml">
  <!ENTITY minitronto "http://www.ii.metu.edu.tr/~kalpaye/daml-s/mini-tronto.daml">
  <!ENTITY process "http://www.daml.org/services/daml-s/0.9/Process.daml">
  <!ENTITY four_reasons_service "http://www.ii.metu.edu.tr/~kalpaye/daml-s/fourReasonsService.daml">
  <!ENTITY four_reasons_process "http://www.ii.metu.edu.tr/~kalpaye/daml-s/fourReasonsProcess.daml">
  <!ENTITY DEFAULT "http://www.ii.metu.edu.tr/~kalpaye/daml-s/fourReasonsProfile.daml">
]>

<rdf:RDF
  xmlns:rdf=     "&rdf;#"
  xmlns:rdfs=    "&rdfs;#"
  xmlns:daml=    "&daml;#"
  xmlns:service= "&service;#"
  xmlns:process= "&process;#"
  xmlns:profile= "&profile;#"
  xmlns:mini-tronto= "&minitronto;#"
  xmlns=         "&DEFAULT;#">
```

```
<daml:Ontology>
  <daml:versionInfo>
    $Id: fourReasinsProfile.daml,v 0.01 $
  </daml:versionInfo>
  <rdfs:comment>
    Four Reasons Hotel Reservation Service Profile
  </rdfs:comment>
  <daml:imports rdf:resource="&rdf;" />
  <daml:imports rdf:resource="&rdfs;" />
  <daml:imports rdf:resource="&daml;" />
  <daml:imports rdf:resource="&service;" />
  <daml:imports rdf:resource="&profile;" />
  <daml:imports rdf:resource="&process;" />
  <daml:imports rdf:resource="&four_reasons_service;" />
  <daml:imports rdf:resource="&four_reasons_process;" />
  <daml:imports rdf:resource="&minitronto;" />
</daml:Ontology>

<minitronto:HotelReservationService rdf:ID="Four_Reasons_Reservation_Service">
  <service:presentedBy rdf:resource="&four_reasons_service;#Four_Reasons_Reservation_Service"/>
  <profile:has_process rdf:resource="&four_reasons_process;#Four_Reasons_Reservation_Service_Process"/>
  <profile:serviceName>Four_Reasons_Reservation_Service</profile:serviceName>
  <profile:textDescription>
    This service provides reservation to Four Reasons Hotel. The service inherits its inputs
    (fromDate, toDate, noOfPersons) and its outputs (HotelReservationID) from the mini travel
    ontology.
    The succeeding generic service is restricted to only "Payment Service". However the service
    can declare succeeding service instances.
  </profile:textDescription>
  <profile:succedingServiceInstance>PHdCard</profile:succeedingServiceInstance>
</minitronto:HotelReservationService>
```

A sub-module (Ontology Database Creator Sub-Module) of the application module parses the DAML-S service ontology and individual service profile definitions using HP's Jena Toolkit [21], which is used to parse RDF and offers a specialized API for DAML processing. The parsed ontology is loaded into the ontology database. The ontology is parsed and stored in a relational database every time the application is initialized.

The database of choice is MySQL [27] since its free, offers high performance and uses standard SQL. It offers a visual administration tool called MySQL Control Center [28] to manipulate the database. The tables created in MySQL

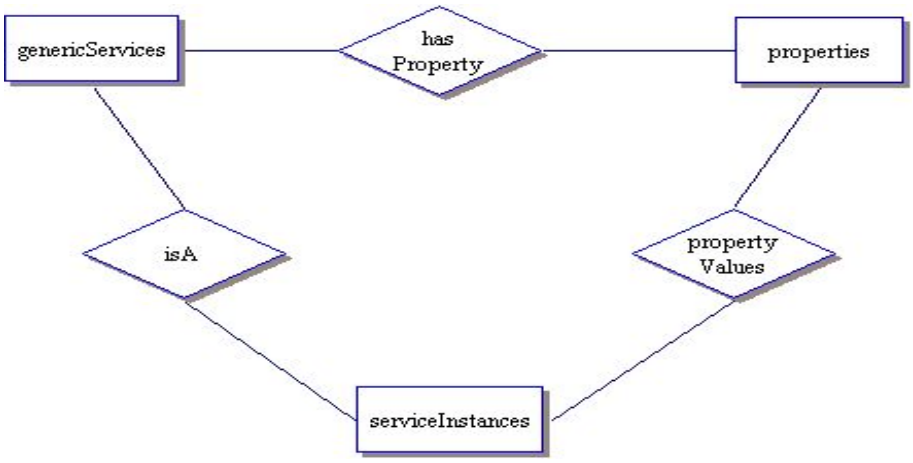

Figure 4.1: Logical Relationships in Ontology Database

are given in the Table 4.1 with the fields, field types and keys. MySQL does not force or implement relationships, however relationships given in Figure 4.1 built when the ontology is parsed and stored in the database, through the application module.

Generic services are stored in the "genericServices" table with unique IDs associated with them. In the DAML-S ontology, generic services are defined as sub-classes of the *Profile* itself. Thus whenever the parser encounters a sub-class of the *Profile* a new entry in the "genericServices" table is created.

In the DAML-S ontology, the inputs and the outputs of a generic service are defined as properties having their domain as the generic service itself and the range as a sub-class of a super parameter class. This super parameter class represent the data type of the property value. The sub-classes correspond to properties in the "properties" table. The type of the super parameter defines the data type of the property in the table (i.e. string in the previous example).

Values for these properties are stored in "propertyValues" table.

"hasProperty" table shows the input and output properties for generic services. The entries in this table are created from the ontology when a property has a generic service as its domain value. The super property of the property shows whether an input or output relation exists.

Table 4.1: Ontology Database Tables

Table Name	Field Name	Field Type	Key
genericServices	GenericServiceID	Varchar(100)	Yes
	GenericServiceDescriptions	Varchar(100)	
hasProperty	GenericServiceID	Varchar(100)	Yes
	PropertyID	Varchar(100)	Yes
	Type	Varchar(100)	
isA	ServiceInstanceID	Varchar(100)	Yes
	GenericServiceID	Varchar(100)	
properties	PropertyID	Varchar(100)	Yes
	PropertyDataType	Varchar(100)	
	PropertyDescription	Varchar(100)	
propertyValues	PropertyID	Varchar(100)	Yes
	Value	Varchar(100)	Yes
serviceInstances	ServiceInstanceID	Varchar(100)	Yes
	ServiceDescription	Varchar(100)	
succeeds	serviceID	Varchar(100)	Yes
	succeedingServiceID	Varchar(100)	Yes

Service instances are stored in a table with their unique IDs and they are associated with their generic service types through the "isA" relation table. They are defined as instances of generic services in the ontology.

Succeeding Generic Services and Succeeding Service Instances are stored in a different table called "succeeds". SGSs are described using a restriction on SGS property in Generic Service definitions. SSIs are described using a SSI property in the service instance definitions.

The summary of this mapping is available Table 4.2. This mapping is used by the Ontology Database Creator Sub-Module to store the DAML-S ontology in the Ontology Database.

4.3 Messaging Scheme

As the three modules of the system can be physically apart as they are logically, a reliable messaging scheme is needed.

The Application Module and the Data Module communicate over a JDBC connection. Application Module sends SQL queries through the connection and

Table 4.2: DAML-S Ontology - Ontology Database Mapping

Description in Ontology	**Description in Ontology Database**
subClassOf #profile	"genericServices" table
<daml:*genericServiceID*>	"serviceInstances" table
subClassOf #*Datatype*Parameter	"properties" table with type *Datatype*
subPropertyOf #input	"hasProperty" with type input
subPropertyOf #output	"hasProperty" with type output
Restriction on Property SGS	"succeeds" table
Value of Property SSI	"succeeds" table

Data Module returns the results as *ResultSet* Java objects.

There are five types of SQL queries that are sent through the JDBC connection. These are:

- *getGenericServices*: this query is used to retrieve all the generic services. The result is shown to the user to start the workflow selection.
- *getInstances*: this query is used to retrieve all the instances of a generic service.
- *getProperties*: this query is used to retrieve the input fields of a service.
- *getSucceedingService*: this query is used to retrieve all the succeeding services, both generic services and service instances, of a given service instance.
- *findType*: this query is used to find out whether a given ID belongs to a service instance or a generic service.

The results of the above stated queries are processed and converted into a string representation very similar to the technique known as URL rewriting for which examples can be found in Section 4.4. This helps grouping and pairing the results in the long result string made up of multiple entries and pairs.

For the fundamentals of the messaging scheme to be clear the first few steps of it is explained in detail. The messaging scheme is summarized in Figure 4.2 through Figure 4.5. The following example clearly shows that the communication between the Application and Mobile Module is actually synchronous.

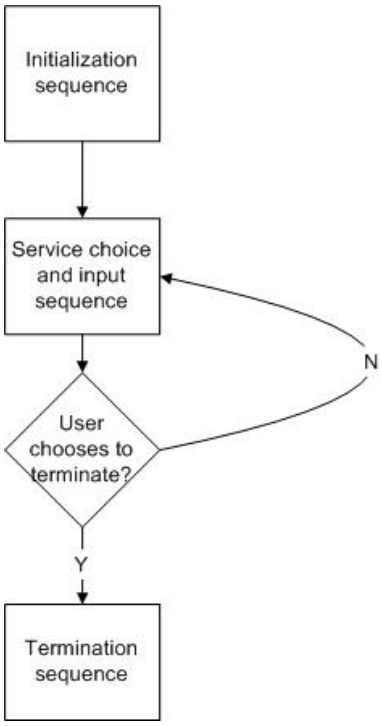

Figure 4.2: Overall System Flow Chart

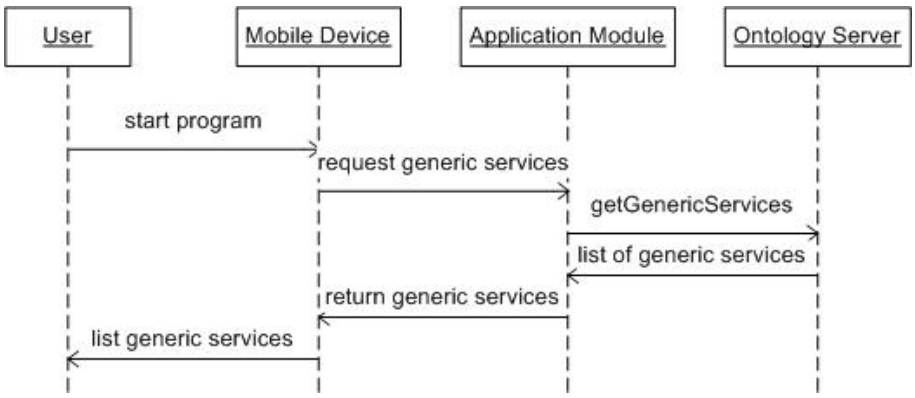

Figure 4.3: Sequence Diagram for Initialization

The process is initiated when the mobile module sends an "initialConnection" string through a socket to the Application Module. Upon receipt of this special string, application module retrieves all the generic services from the data module using the *getGenericServices* query. The list of *genericServiceID*s and *genericServiceDescriptions* are sent to the Mobile Module in the following form:

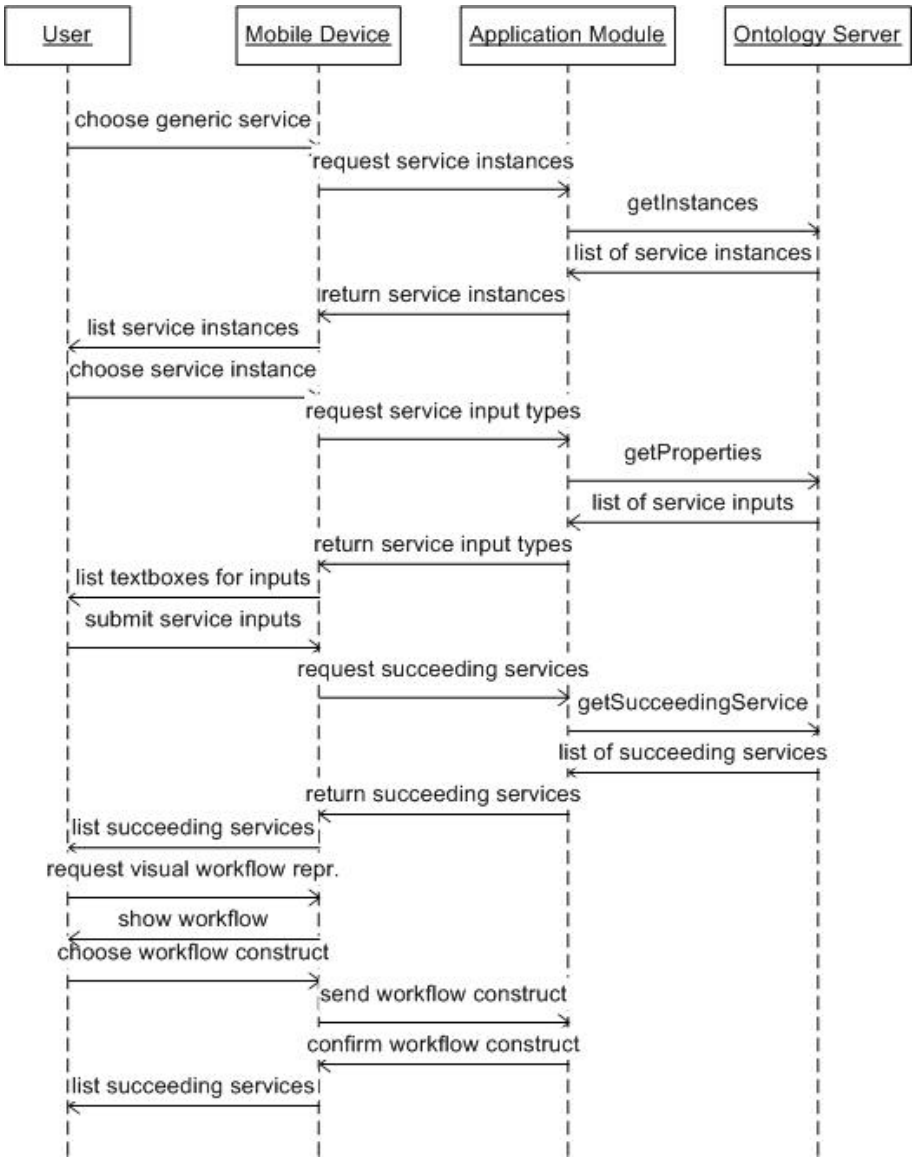

Figure 4.4: Sequence Diagram for Service Selection

genericServices?id=genericServiceID&
description=genericServiceDescription?id=...?

Mobile Module parses the incoming message string. The first phrase of the message string indicates that the following pairs are generic service IDs and descriptions. It lists the *genericServiceDescriptions* to the user. The user chooses one of them and the module sends back only the *genericServiceID* of the selected generic service.

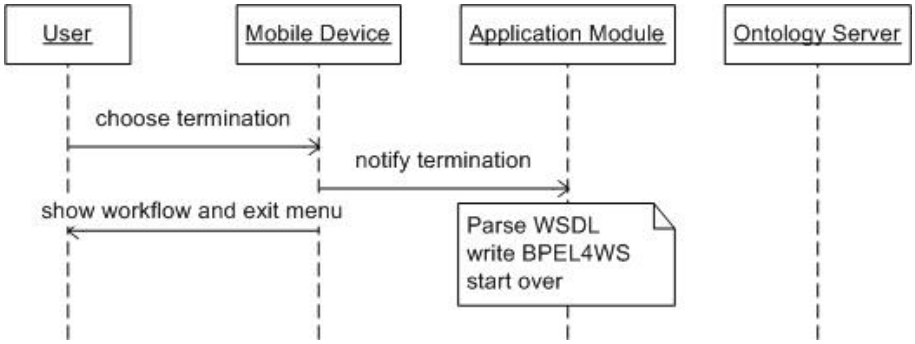

Figure 4.5: Sequence Diagram for Termination

If no initial phrase exists, application module understands that the number is in fact an ID: either a *genericServiceID* or a *serviceInstanceID*. The application module proceeds accordingly.

The requests and respective replies the two modules generate are listed in Table 4.3. The two modules generate the replies whenever they receive the corresponding request, however it should be noted that; if the expected order of the requests are not followed, generated flow and BPEL4WS file may be invalid.

4.4 Presentation (Mobile) Layer

The presentation layer is implemented using the Java 2 Micro Edition (J2ME) platform. The code written is in accordance with the CLDC and MIDP specifications thus, the application runs on even the most limited, J2ME-enabled mobile devices.

The user interface is designed in grayscale and menus are employed for the user to make choices. The user interface map is given in Figure 4.6. Text boxes are used for the user to enter free text service inputs. When the workflow is created and drawn on the screen, the user can navigate on the workflow and get information about the services.

The main *MobileWare* class which is instantiated by the application manager of the cellular phone is a sub-class of *MIDlet* class in MIDP API. It has methods like *startApp, pauseApp* and *destroyApp* for MIDlet state changes. The *Mobile-*

Table 4.3: Messaging Scheme

ID	**Request**	**Requesting Module**	**Reply To**
1	initialConection	Mobile	-
2	GenericServices?id=*genericServiceID*& description=*genericServiceDescription*?...?	Application	1
3	*genericServiceID*	Mobile	2,8
4	*ServiceInstances*?*ServiceInstanceID*=*serviceInstanceID*& ServiceInstanceDescription=*serviceInstanceDescription*?...?	Application	3
5	*serviceInstanceID*	Mobile	6
6	ServicePropertyDescriptions?PropertyID=propertyID& PropertyDataType=*propertyDataType*& PropertyDescription=*propertyDescription*?...?	Application	5
7	userInfo?*propertyID*=*propertyValue*&...?	Mobile	6
8	SucceedingServices?value=*genericServiceID*&*genericServiceDesc* ription=*genericServiceDescription*?*value*=*serviceInstanceID*&*servi* ceDescription=*serviceInstanceDescription*?...?	Application	7,9
9	wfc=*wfcName*	Mobile	8
10	wfcOK	Application	9

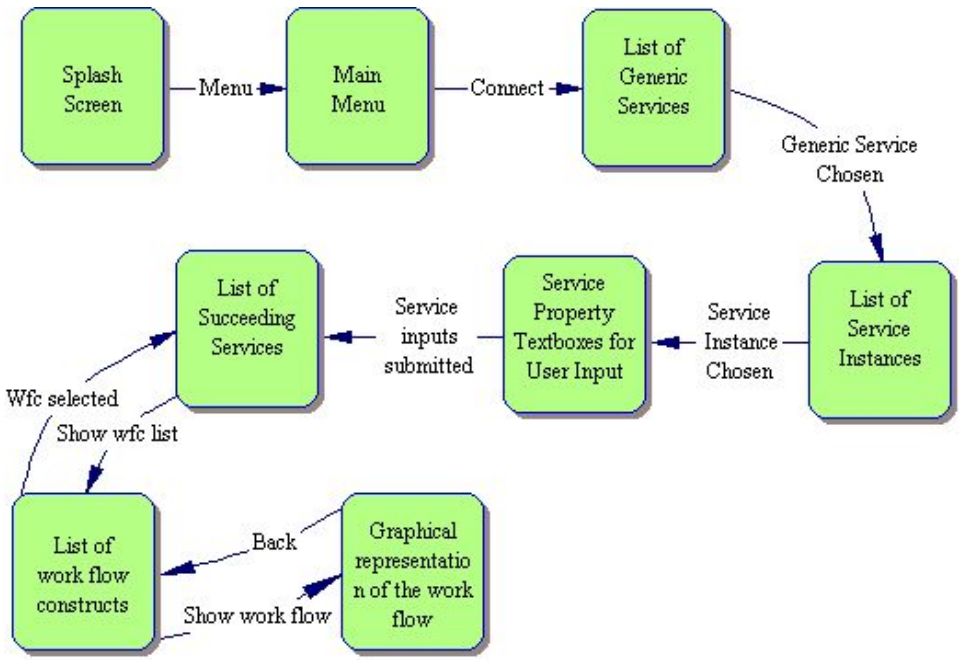

Figure 4.6: User Interface Map for the Mobile Module

Ware class also implements the *CommandListener* interface to get commands from the user in the form of menu selection, text box input, soft button com-

Figure 4.7: Mobile Module Welcome Message Screen

mands etc. The actions performed by the user call the *commandAction* method. User commands are identified for the program to perform respective actions.

All classes, their methods and important fields for the mobile module of the classes are listed in Appendix A.

Following is a detailed view of the steps the system takes in the mobile module in the order of execution. The steps can also be followed from Figure 4.2 through Figure 4.5.

1. As the MIDlet starts its execution the constructor of the *MobileWare* class is called which initiates the screen and data objects.

2. A welcome message screen as in Figure 4.7 with two commands, *Menu* and *Exit* is displayed. *Exit* quits the program and *Menu* brings up a main

Figure 4.8: Mobile Module Main Menu

menu (Figure 4.8) that has links to more information about the program, help text and a *Connect* option that starts actual execution.

3. When the *Connect* option is selected, the *connectNow* method with a string parameter "initialConnection" is called. This method opens a standard socket connection to application module, sends the request and starts a timer that waits while the application module to gather data, process it and return a result. The timer triggers the *connectingMIDLET* a sub-class of *TimerTask* in MIDP API. This class opens an *InputStream* that reads the response from the application module into a string. The string is parsed using the *parseResponse* method which separates the message header from the contents and stores the contents into a two dimensional

Figure 4.9: Mobile Module List of Generic Services

array of strings named *serviceArray*. The MIDlet then proceeds depending on the header of the message. From now on the process of connecting to the application module, sending a message, getting a response and parsing it is called "submitting the request containing a message to the application module".

4. From Table 4.3, the response for the "initialConnection" string is the IDs and descriptions of all generic services. The *listServices* method is called with the *serviceArray* parameter. A list of generic services appears on the screen (Figure 4.9). The user will choose one of them and *a request containing the ID of the selected generic service is submitted to the application module.*

Figure 4.10: Mobile Module List of Service Instances

5. The application module's response contains IDs and descriptions of *all instances* of the generic service selected in Step 4. It should be noted that it is possible to apply a filter that reduces the number of instances returned by forcing a condition on some service attribute. This can be quality of service, geographical location, service provider etc. However this filtering mechanism is out of the scope of this thesis. The service instances are listed on the screen through the *listServices* method as in Figure 4.10 and the user will choose one of them. A *request containing the ID of the selected service instance is submitted to the application module.* The service instance information is used to create a *WFServiceInstance* object and stored in memory. The data structure contains a series of objects,

Figure 4.11: Mobile Module Input Properties Text Boxes

which are either service instances represented by a *WFServiceInstance* object or flow objects represented by *WFFlow* objects. *WFFlow* objects are actually arrays any of which can contain sequences of *WFFlow* and *WFServiceInstance* objects.

6. The application module's response contains all the property IDs, descriptions and data types of the service instance selected in step 5. Text boxes corresponding to each property is shown on the screen (Figure 4.11). The user will fill the text boxes in free text and a *request containing the property IDs and user input corresponding is submitted to the application module.* The format of the message is:
   userInfo?propertyID=propertyValue&...?

Figure 4.12: Mobile Module List of Workflow Constructs

7. The application module's response contains the IDs of SGSs and SSIs of the service instance selected in step 5. They are listed on the screen (Figure 4.14) through the *listServices* method. The user should first give the *selectWFC* command to choose a workflow construct. A list of available workflow constructs is shown in Figure 4.12 which includes: a new flow command, a branch completed command and a sequence (next) command. If a "new flow" command is issued, the MIDlet creates as much *WFFLow* objects as the number of branches and pushes them into a stack. If a "branch completed" command is issued the *WFFlow* object is popped. If a "next" command is issued the flow continues with another service. The user will choose one of them and a request containing the workflow con-

Figure 4.13: Mobile Module Graphical Representation of the Workflow

struct name is submitted to the application module. The format of the message is:

wfc=wfcName

The user can see the workflow upto that moment by giving the *showWF* command. This commands sets the displayed object to an instance of *workFlowCanvas* object which is a sub-class of *Canvas*. The paint method of this object will traverse the data structure representing the flow and create a visual representation of services as boxes and lines connecting consecutive services (Figure 4.13). The user can use cellular phone dependent "soft" buttons to navigate through the flow and choose any of the services to get information about it.

Figure 4.14: Mobile Module List of Succeeding Services

8. If the workflow construct chosen in step 7 does not imply any changes in the succeeding services the response is "wfcOK". If the selected workflow construct implies a different set of succeeding services, the new set of succeeding services is received from the application module. This kind of a change can be necessary if a flow ends and the next service could be the union of last services of the ending branches. This policy of combining succeeding service sets could be changed; e.g. to be the intersection of service sets.

9. The user sees the list of the succeeding services again (Figure 4.14) and user will choose one of them. A *request containing the ID of the selected service instance or generic service is submitted to the application module.* The

execution will go back to step 6 or step 7 depending on whether a service instance or a generic service is selected until a special "terminate" service is chosen. In this case, the finalized flow is shown and the application module is notified.

4.5 Application Layer

The application layer is implemented as a standard application using Java 2 Standard Edition. It communicates with the Data Module through a JDBC Connection; and Presentation (Mobile) Module through a standard socket connection. All classes, methods and important fields of the Application Module is available in Appendix B.

Application Module has five sub-modules with respect to functionality. These are:

- Controller: this sub-module makes decisions about the actions of the Application Module. It calls other sub-modules to fulfill the requests from the Mobile Module; and reply them correctly. This module is explained in detail in Section 4.5.1.

- Messenger: this sub-module is responsible for the communication between the Application Module and the Mobile Module. Two Java classes make up the messenger sub-module.

  MessageReceiver listens to the port 5678 and has an infinite loop to accept connections. When a connection is made, it opens a *BufferedInputStream* to accept the Mobile Module request. The request is stored in a string object. A new instance of the *Controller* class is generated and its *run* method is called with the mobile module request as the parameter. The *run* method returns the message to be sent as a string. The returned string is used to call the *MessageSender* class.

  MessageSender is the other class that makes up the Messenger sub-module. It is also called by the *MessageReceiver* and replies the Mobile Module

through the same socket connection. The connection is closed afterwards.

- Ontology Database Creator: This sub-module is once executed when the Application Module is initialized. Its responsibility is to populate the Ontology Database using the DAML-S ontology.

  The DAML-S ontology is parsed using HP's Jena RDF Parser. Using the mapping given Table 4.2, the database tables in Table 4.1 are created and are populated with data. Details of the process is given in Section 4.2.

- BPEL4WS Generator: This sub-module is executed after the "terminate" signal from the Mobile Module is received. The workflow structure stored in the memory, that has been generated from the service instance and workflow construct choices of the user, is combined with the service interface definitions from the WSDL files. This information is used to construct an executable BPEL4WS instance. Details of BPEL4WS document generation are given in Section 4.5.2.

- Ontology Database Retriever: This sub-module is executed each time the application module needs to get some data from the Ontology Database. The *OntologyDatabaseRetriever* class has methods for the generation of five types of SQL queries stated in the Section 4.3: *getGenericServices, getInstances, getProperties getSucceedingServices*, and *findType*. The queries generated by these methods are fed into the *runSQL* method. *runSQL* method returns *ResultSet* Java objects which are converted into strings by the *getStringRepresentation* method. The string representation is sent to the Mobile Module by the Messenger Sub-Module.

4.5.1 Controller Sub-Module

Controller Sub-Module organizes the actions performed by the Application Module. The sub-module is created and called by the *MessageReceiver* class of the Messenger sub-module with the Mobile Module request as a parameter to the *run* method.

The *run* method of the Controller Sub-Module employs an "if..else if.." structure with the header of the Mobile Module request as the condition. This structure is used to construct a string that is sent back to the Mobile Module. Generally the header is added by the *run* method and the payload of the message is created by a method of the Controller Sub-Module or Ontology Database Retriever Sub-Module.

The Mobile Module request "initialConnection" means this is the first request of the session. Controller Sub-Module sets the header of the message as "GenericServices" and calls *getGenericServices* method of the Ontology Database Retriever Sub-Module, which generates the necessary string including generic service ID and dexcriptions.

If the request starts with "userInfo", this means that the Mobile Module is submitting the inputs of a service that the user supplied. *userInfoParser* method of Controller Sub-Module is called to parse the values supplied and the succeeding services are retrieved using the *getServiceInstance* method of the Ontology Database Retriever Sub-Module.

Mobile Module sends a workflow construct with a header "wfc". If such a request arrives to Controller Sub-Module, the workflow construct is checked to see whether it is a special construct implying a new set of succeeding services. A new set of succeeding services (using the *getSucceedingServices* method of the Ontology Database Retriever Sub-Module) or a "wfcOK" message is sent depending on the workflow construct type.

The Controller Sub-Module possesses the same data structures to hold the workflow in memory as the one built for the Mobile Module. The same actions as in step 7 of Section 4.4 are executed.

If a "terminate" message is received from the Mobile Module, the execution is stopped and BPEL4WS Generator Sub-Module is initiated.

If none of the above are found in the incoming string, then the Mobile Module has obviously submitted an ID. Controller Sub-Module calls the *findType* method of the Ontology Database Retriever Sub-Module to see whether the ID belongs

to a generic service or a service instance. If the ID belongs to a generic service, instances of the generic service are retrieved using the *getInstances* method of Ontology Database Retriever Sub-Module. If the ID belongs to a service instance, inputs of the service are retrieved using the *getProperties* method. Respective headers are added and sent to the Mobile Module.

When a service instance is selected by the user and its inputs are sent to the Mobile Module by the Application Module, Controller Sub-Module stores the service instance information into a data structure that is the same as the Mobile Module. The same actions as in step 5 of Section 4.4 are executed.

4.5.2 BPEL4WS Generator Sub-Module

The BPEL4WS Generator Sub-Module is called by the Controller Sub-Module when the Mobile Module signals that the user has chosen to end the workflow via the "terminate" service.

BPEL4WS, XML representation of the generated workflow relies heavily on WSDL service interface definitions. Thus, during the BPEL4WS generation process, WSDL documents for services that are available are parsed when needed.

A BPEL4WS document automatically generated using the system developed in this thesis is available in Appendix E. WSDL documents used for the generation of the BPEL4WS documents are given in Appendix F.

This process combines the flow of services created by the user with the interface definitions of the services chosen. The service flow structure resides in the memory in a custom designed data structure which has been explained in Chapter 3. WSDL files are parsed using an XML parser.

```
<message name="approvalMessage">
  <part name="accept" type="xsd:string"/>
</message>

<portType name="loanApprovalPT">
  <operation name="approve">
    <input message="loandef:creditInformationMessage"/>
    <output message="tns:approvalMessage"/>
    <fault name="loanProcessFault"
```

```
            message="loandef:loanRequestErrorMessage"/>
    </operation>
  </portType>
```

From the above WSDL definition of a service the following *partnerLinkType* and *partnerLinks* are generated:

```
 <slnk:partnerLinkType name="loanApprovalLinkType">
    <slnk:role name="approver">
      <portType name="apns:loanApprovalPT"/>
    </slnk:role>
 </slnk:partnerLinkType>

<partnerLinks>
    <partnerLinks name="approver"
      partnerLinkType="lns:loanApprovalLinkType"
      partnerRole="approver"/>
</partners>
```

The BPEL4WS workflow structure uses these declarations as follows:

```
<sequence>
  <receive name="receive1" partner="customer"
    portType="apns:loanApprovalPT"
    operation="approve" variable="request"
    createInstance="yes">
  </receive>
</sequence>
```

The *variable* in the example above is declared as the following code section; from the *property* entries in the Ontology Database where the *messageType* declarations are taken from the WSDL documents' *message* declarations:

```
<variables>
  <variable name="request" messageType="loandef:CreditInformationMessage"/>
  <variable name="approvalInfo"  messageType="apns:approvalMessage"/>
</variables>
```

A BPEL4WS document consists of two sections. The "declaration" section includes: namespace declarations; *partnerLinkType* and *partnerLink* declarations for each service; and *variable* declarations for data exchange between partners.

The “activity” section includes regular activities and structure activities that define the flow of work.

The data structure representing the flow is traversed once and above stated sections are generated concurrently.

Each service in the flow has its own *partnerLinkType* and *partnerLink* definition. *variables* correspond to the *propertyValues* table in Ontology Database. For every entry in the table a BPEL4WS *variable* is created. If the variable is input by the user, the value is copied using an Xpath expression to the respective variable. If the variable value is generated by a service as an output, the value in the appropriate variable is assigned directly.

For every service in the flow the above declarations are given. When the flow branches and several services or service sequences are executed concurrently, a <flow> structure activity is written. <sequence> activity structure is used for activities that need to to be executed in sequence. The other structure activities like <switch>, <while> and <pick> are not implemented in this thesis.

CHAPTER 5

Conclusions and Future Work

The first revolution was the introduction of the Personal Computer (PC). The second revolution was the Web. It is claimed that the third will be the realization of the Semantic Web.

For the first two revolutions, the social impact was so huge that the economical and business opportunities followed the social impacts.

Semantic Web coupled with the current e-Commerce technologies like WSDL, SOAP, ebXML etc. unleash great opportunities involving the automatic discovery, composition, invocation and monitoring of web services for e-Business. However with the current level of technology only some of these four activities can be performed to some extent, since the involvement of semantics in the process is minor.

Of all the four activities, web service composition is currently the most challenging since it needs a full semantic description of the services and an infrastructure capable of carrying the semantics. DAML-S, especially designed and evolved for semantic web service definitions, is the main building block of this thesis.

It seems that current technology is not able to perform full automatic service composition. In this thesis, we have realized a semi automatic web service composition architecture in which the user has the control of the composition

and service semantics aids her in service discovery and automation.

BPEL4WS on the other hand is the natural choice for the description of the service composition generated since it is designed for this specific purpose. BPEL4WS documents generated in this thesis are validated by the BPEL4J Validator by IBM alphaWorks. The scope of this thesis includes the generation of valid BPEL4WS documents. These documents along with the respective WSDL documents of the services in the flow can be fed into a BPEL engine, an implementation of which is available from IBM alphaWorks BPEL4J. When the process represented by the BPEL4WS document is deployed into the engine, it waits for a specific message from a specific port type to be executed.

Building of a workflow is justified if the workflow is executed many times. The workflows can be executed with the same configuration, or with a new set of inputs every time it is executed. A web interface can be configured to take the new set of inputs. The process may also be configured as a web service. In this case, protocols such as SOAP can be used to invoke the BPEL4WS workflow that has been composed and deployed on BPEL4J. A complex banking transaction; that involves several steps like sale of stocks or bonds, money transfer and payment; and which is executed repeatedly can be defined as a workflow and deployed on a BPEL4J engine. Similarly a workflow for frequent travellers which discovers hotels and flights, reserves rooms and flight tickets, makes payments can be defined so that the user can execute the process with different set of inputs.

It should be noted that, realization of such a complex activity on a cellular phone and the seamless integration with different platforms and the level of flexibility provided by Java 2 Micro Edition is impressive.

This thesis will be used as a part of the IST-1-002103-STP ARTEMIS:A Semantic Web Service-based P2P Infrastructure for the Interoperability of Medical Information Systems Project. The architecture provided and realized in this thesis can be taken as a basis for other projects also, to develop semi automatic web service composition systems in mobile environments. Even an inference engine

to perform matchmaking can be used to let the service providers free of deciding succeeding services to the services they provide.

In this thesis we have proved that it is possible to compose semantically enriched web services in a mobile environment. A state-of-the-art and comprehensive implementation of this thesis is left as a future work, which includes: The implementation of other work flow constructs like loops and branches; the implementation of the application layer in an application server; the realization of the Ontology Database in a public registry like UDDI or ebXML; and the filtering of service instances with respect to any service attribute. The deployment of executable BPEL4WS documents into a BPEL engine like BPEL4J and their invocation with different input configurations constitutes an extension to this thesis.

REFERENCES

[1] IBM Business Process Execution Language for Web Services Java Run Time (BPWS4J), http://www.alphaworks.ibm.com/tech/bpws4j, last updated April 30, 2003.

[2] Business Process Execution Language for Web Services (BPEL4WS) Specification Version 1.0, http://dev2dev.bea.com/technologies/webservices/BPEL4WS.jsp

[3] Business Process Execution Language for Web Services (BPEL4WS) Specification Version 1.1, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, last updated May 5, 2003.

[4] Business Process with BPEL4WS: Learning BPEL4WS, IBM developerWorks Tutorial, http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol1/, last updated August 1, 2002.

[5] Connected Device Configuration (CDC), http://java.sun.com/products/cdc/

[6] Connected Limited Device Configuration (CLDC), http://java.sun.com/products/cldc/

[7] M. Colan, "Web Services: What's It All For?", http://ibm.com/developerworks/webservices

[8] Defense Advanced Research Projects Agency (DARPA), http://www.darpa.mil/

[9] DARPA Agent Markup Language (DAML), http://www.daml.org, last updated August 19, 2003.

[10] DAML-Services, http://www.daml.org/services/daml-s/0.9/, last updated May 7, 2003.

[11] The DAML Services Coalition: Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A.

McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne and Katia Sycara. "DAML-S: Web Service Description for the Semantic Web." In The Proceedings of the First International Semantic Web Conference (ISWC), 2002

[12] DAML+OIL, http://www.daml.org/2001/03/daml+oil-index.html, last updated March, 2001.

[13] DAML+OIL (March 2001) Reference Description, http://www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218

[14] The DAML-S Definition of the Service, http://www.daml.org/services/daml-s/0.9/Service.daml, last updated May 7, 2003.

[15] A. Dogac, "A Tutorial on Exploiting Semantic of Web Services through ebXML Registries", eChallenges 2003, October 2003, Bologna, Italy

[16] A. Dogac, Y. Kabak, G. B. Laleci,"A Semantic-Based Web Service Composition Facility for ebXML Registries", 9th International Conference of Concurrent Enterprising, Espoo, Finland, June 2003

[17] Electronic Business using eXtensible Markup Language (ebXML), http://www.ebxml.org

[18] D. Fensel, "The Semantic Web and its Languages", IEEE Computer Society NOVEMBER/DECEMBER 2000

[19] I. Horrocks, "DAML+OIL: a Reason-able Web Ontology Language", In. Proceedings of EDBT 2002, March 2002

[20] Java Community Process Program, http://www.jcp.org

[21] Jena Semantic Web Toolkit, HP, http://www.hpl.hp.com/semweb/jena.htm

[22] J2ME Frequently Asked Question (FAQ), http://java.sun.com/j2me/faq.html

[23] Kabak, Y., "Exploiting Web Service Semantics through ebXML Registries", MS Thesis, Middle East Technical University, 2003

[24] T. B. Lee, J. Hendler, O. Lassile, "The Semantic Web, A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities", Scientific American, May 2001.

[25] Mobile Information Device Profile (MIDP), http://java.sun.com/products/midp/

[26] core J2ME Technology & MIDP, John W.MUCHOW, Sun Microsystems Press, 2002

[27] MySQL Relational Open Source Database, http://www.mysql.com/

[28] MySQL Control Center, http://www.mysql.com/products/mysqlcc

[29] OIL, http://www.daml.org/2001/03/daml+oil-index.html, last updated March 2001.

[30] Web Ontology Language, http://www.w3.org/TR/owl-ref, last updated August 18, 2003.

[31] Paolucci, M., Kawamura, T., Payne, T., Sycara, K., "Semantic Matching of Web Services Capabilities", in Proc. of Intl. Semantic Web Conference, Sardinia, Italy, June 2002.

[32] Resource Description Framework (RDF), http://www.w3.org/RDF/

[33] Resource Description Framework Schema (RDFS),// http://www.w3.org/TR/rdf-schema/, last updated September 05, 2003.

[34] Simple Object Access Protocol (SOAP), http://www.w3.org/TR/soap12-part1/, last updated June 24, 2003.

[35] Web Ontology Language (OWL) Reference Version 1.0,// http://www.daml.org/2002/06/webont/owl-ref-proposed, last updated August 18, 2003.

[36] Universal Description, Discovery and Integration of Web Services (UDDI), http://www.uddi.org

[37] IBM WebSphere SDK for Web Services (WSDK) Version 5.0.1, Overview, http://www-106.ibm.com/developerworks/webservices/wsdk/

[38] Describing Web services: WSDL, IBM Developerworks tutorial, http://www-106.ibm.com/developerworks/edu/ws-dw-ws-dewsdl-i.html

[39] Web Services Description Language (WSDL) Version 1.2, http://www.w3.org/TR/wsdl12/, last updated June 11, 2003.

[40] Web Services Flow Language Specification version 1.0, http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[41] Web Services - The Web's next revolution, http://www-106.ibm.com/developerworks/ edu/ws-dw-wsbasics-i.html

[42] Web Services Toolkit Tutorial, IBM, http://ibm.com/developerworks/webservices

[43] World Wide Web Consortium (W3C), http://www.w3.org/

[44] XLANG, Web Services for Business Process Design, http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

[45] XML Schema, http://www.w3.org/XML/Schema

APPENDIX A

Classes of Mobile Module

```
public class MobileWare extends javax.microedition.midlet.MIDlet implements CommandListener
  public MobileWare()
  public void startApp()
  public void pauseApp()
  public void destroyApp(boolean unconditional)
  public void commandAction(Command c, Displayable s)
  public void listServices(String[][] serviceArray, Displayable currentDisplay, boolean isSucceeding)
  public void userInput(String[][] serviceArray)
  public void connectNow(String str)

  public class screenObject
    public screenObject(String ID, String description, int x, int y)

  public class drawingStackObject
    public drawingStackObject(WFFlow flowObject, int branch, int leftInterval, int rightInterval, int y)

  public class workFlowCanvas extends Canvas implements CommandListener
    public void paint(Graphics g)
    public void commandAction(Command c, Displayable s)
    public int getMinX(int cursorY)
    public int getMaxX(int cursorY)
    public void keyPressed(int keyCode)
    public workFlowCanvas(MobileWare midlet)

  public class WFServiceInstance
    public WFServiceInstance()

  public class WFFlow
```

```
  public WFFlow(int size)

public class connectingMIDLET extends TimerTask
  public void run()

public String[][] parseResponse(String response)
```

APPENDIX B

Classes of Application Module

```
public class ebXMLMessenger
  public String getPropertyValue(String propertyID)
  public String checkPropertyValues(String serviceInstanceID)
  public String getServiceDescription(String serviceInstanceID)
  public void insertPropertyValues(String propertyID, String value)
  public String getServiceOutputs(String serviceInstanceID)
  public String getSucceedingServices(String serviceInstanceID)
  public String getProperties(String serviceInstanceID)
  public String getGenericServices()
  public String getInstances(String genericServiceID)
  public String findType(String id)
  public ResultSet runSQL(String sql)
  public String getStringRepresentation(ResultSet rs)
  public ebXMLMessenger()

public class MessageReceiver
  public static void main(String[] args)

public class MessageSender
  public MessageSender(String str)

public class MiddleControl
  public String userInfoParser(String mobileRequest)
  public boolean wfcChecker(String wfc)
  public String run(String mobileRequest)
  public void MiddleControl()

class ServiceOutputVectorObject
```

```
  public ServiceOutputVectorObject(String propertyID, String messageName)

public class WFFlow
  public WFFlow(int size)

public class WFServiceInstance
  public WFServiceInstance()

public class WritingStackObject
  public WritingStackObject(WFFlow flowObject, int branch)

public class WSDLContentHandler implements ContentHandler
  public void setDocumentLocator(Locator locator)
  public void startDocument()
  public void endDocument()
  public void startPrefixMapping(String prefix, String url)
  public void endPrefixMapping(String prefix)
  public void startElement(String namespaceURI, String localname, \\ String qname, Attributes atts)
  public void endElement(String namespaceURI, String localname, String qname)
  public void characters(char[] ch, int start, int length)
  public void ignorableWhitespace(char[] ch, int start, int length)
  public void processingInstruction(String target, String data)
  public void showCharacters(char[] ch, int start, int length)
  public void skippedEntity(String name)

public class WSDLParser
  public WSDLServiceObject run(String filename)
  public WSDLParser()

public class WSDLServiceObject
  public WSDLServiceObject()
```

APPENDIX C

Extended DAML-S Profile

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!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 daml "http://www.daml.org/2001/03/daml+oil">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY service "http://www.daml.org/services/daml-s/0.9/Service.daml">
  <!ENTITY process "http://www.daml.org/services/daml-s/0.9/Process.daml">
  <!ENTITY DEFAULT "http://www.daml.org/services/daml-s/0.9/Profile.daml">
  <!ENTITY country "http://www.daml.org/services/daml-s/0.9/Country.daml">
  <!-- <!ENTITY country "http://www.daml.ri.cmu.edu/ont/Country.daml"> -->
]>

<rdf:RDF
  xmlns:rdf= "&rdf;#"
  xmlns:rdfs= "&rdfs;#"
  xmlns:daml= "&daml;#"
  xmlns:xsd=  "&xsd;#"
  xmlns:service= "&service;#"
  xmlns:process= "&process;#"
  xmlns= "&DEFAULT;#">

  <daml:Ontology>
    <daml:versionInfo>
      $Id: Profile.daml,v 1.22 2003/07/25 23:22:14 martin Exp $
    </daml:versionInfo>
    <daml:comment>
      Process and Service Coalition
```

```
      First cut at DAML ontology for Advertisements (i.e. Profiles) based
      upon the Service Model.
      Version of Profile for DAML-S 0.9 Release
      Created by Terry Payne (terryp@cs.cmu.edu).
      Modified by Massimo Paolucci (paolucci@cs.cmu.edu)

    </daml:comment>
    <daml:imports rdf:resource="&rdf;" />
    <daml:imports rdf:resource="&daml;" />
    <daml:imports rdf:resource="&service;" />
    <daml:imports rdf:resource="&process;" />
  </daml:Ontology>

  <daml:Class rdf:ID="Profile">
    <daml:label>Profile</daml:label>
    <rdfs:subClassOf rdf:resource="&service;#ServiceProfile" />
    <daml:comment>
      Definition of Profile
    </daml:comment>
  </daml:Class>

<!--*************************************************************-->
<!--*************** Extended by K.Alpay Erturkmen ***************-->
<!--*************************************************************-->
  <daml:ObjectProperty rdf:ID="succeedingGenericService">
    <daml:domain rdf:resource="#Profile" />
    <daml:range rdf:resource="#Profile" />
  </daml:ObjectProperty>

  <daml:Class rdf:about="#Profile">
    <rdfs:subClassOf>
      <daml:Restriction daml:minCardinality="1">
        <daml:onProperty rdf:resource="#succedingGenericServices" />
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:ObjectProperty rdf:ID="succeedingServiceInstance">
    <daml:domain rdf:resource="Profile" />
    <daml:range rdf:resource="Profile" />
  </daml:ObjectProperty>
<!--*************************************************************-->
<!--*************************************************************-->
<!--*************************************************************-->

  <daml:DatatypeProperty rdf:ID="serviceName">
```

```
    <daml:domain rdf:resource="#Profile"/>
    <daml:range rdf:resource="&xsd;#string"/>
  </daml:DatatypeProperty>

  <daml:Class rdf:about="#Profile">
    <rdfs:comment>
    A profile can have only one name
    </rdfs:comment>
    <rdfs:subClassOf>
      <daml:Restriction daml:cardinality="1">
<daml:onProperty rdf:resource="#serviceName"/>
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:DatatypeProperty rdf:ID="textDescription">
    <daml:domain rdf:resource="#Profile"/>
    <daml:range rdf:resource="&xsd;#string"/>
  </daml:DatatypeProperty>

  <daml:Class rdf:about="#Profile">
    <rdfs:comment>
    A profile can have only one text description
    </rdfs:comment>
    <rdfs:subClassOf>
      <daml:Restriction daml:cardinality="1">
<daml:onProperty rdf:resource="#textDescription"/>
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:ObjectProperty rdf:ID="contactInformation">
    <daml:domain rdf:resource="#Profile"/>
    <daml:range rdf:resource="#Actor"/>
  </daml:ObjectProperty>

  <daml:UniqueProperty rdf:ID="has_process">
    <daml:domain rdf:resource="#Profile"/>
    <daml:range rdf:resource="&process;#ProcessPowerSet"/>
  </daml:UniqueProperty>

  <daml:Property rdf:ID="serviceCategory">
    <daml:domain rdf:resource="#Profile"/>
    <daml:range rdf:resource="#ServiceCategory"/>
  </daml:Property>
```

```
<daml:Property rdf:ID="serviceParameter">
  <daml:domain rdf:resource="#Profile"/>
  <daml:range rdf:resource="#ServiceParameter"/>
</daml:Property>

<daml:Property rdf:ID="qualityRating">
  <daml:domain rdf:resource="#Profile"/>
  <daml:range rdf:resource="#QualityRating"/>
</daml:Property>

<daml:ObjectProperty rdf:ID="parameter">
  <daml:domain rdf:resource="#Profile"/>
  <daml:range rdf:resource="#ParameterDescription"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="input">
  <daml:subPropertyOf rdf:resource="#parameter" />
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="output">
  <daml:subPropertyOf rdf:resource="#parameter" />
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="precondition">
  <daml:subPropertyOf rdf:resource="#parameter" />
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="effect">
  <daml:subPropertyOf rdf:resource="#parameter" />
</daml:ObjectProperty>

<daml:Class rdf:ID="ParameterDescription"/>

<daml:DatatypeProperty rdf:ID="parameterName">
  <daml:domain rdf:resource="#ParameterDescription"/>
  <daml:range rdf:resource="&xsd;#string"/>
</daml:DatatypeProperty>

<daml:Property rdf:ID="restrictedTo">
  <daml:domain rdf:resource="#ParameterDescription"/>
  <daml:comment>
    Range is left unspecified, to allow for both DAML classes and
    XSD datatypes.
  </daml:comment>
  <!-- <daml:range  rdf:resource="&daml;#Class"/> -->
</daml:Property>
```

```
  <daml:Class rdf:about="#ParameterDescription">
    <daml:comment>
    a Parameter is restricted to refer to only one concept in some
    ontology
    </daml:comment>
    <rdfs:subClassOf>
      <daml:Restriction daml:cardinality="1">
<daml:onProperty rdf:resource="#restrictedTo"/>
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:ObjectProperty rdf:ID="refersTo">
    <daml:domain rdf:resource="#ParameterDescription"/>
    <daml:range  rdf:resource="&process;#parameter"/>
  </daml:ObjectProperty>

  <daml:Class rdf:about="#ParameterDescription">
    <daml:comment>
    a Parameter is restricted refer to only one parameter in the
      process model
    </daml:comment>
    <rdfs:subClassOf>
      <daml:Restriction daml:cardinality="1">
<daml:onProperty rdf:resource="#refersTo"/>
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:Class rdf:ID="ServiceCategory"/>

  <daml:DatatypeProperty rdf:ID="categoryName">
    <daml:domain rdf:resource="#ServiceCategory"/>
    <daml:range rdf:resource="&xsd;#string"/>
  </daml:DatatypeProperty>

  <daml:Class rdf:about="#ServiceCategory">
    <daml:comment>
    a ServiceCategory is restricted to refer to only onename
    </daml:comment>
    <rdfs:subClassOf>
      <daml:Restriction daml:cardinality="1">
<daml:onProperty rdf:resource="#categoryName"/>
      </daml:Restriction>
    </rdfs:subClassOf>
```

```
  </daml:Class>

  <daml:DatatypeProperty rdf:ID="taxonomy">
    <daml:domain rdf:resource="#ServiceCategory"/>
    <daml:range  rdf:resource="&xsd;#string"/>
  </daml:DatatypeProperty>

  <daml:Class rdf:about="#ServiceCategory">
    <daml:comment>
    a ServiceCategory is restricted to refer to only one taxonomy
    </daml:comment>
    <rdfs:subClassOf>
      <daml:Restriction daml:cardinality="1">
<daml:onProperty rdf:resource="#taxonomy"/>
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:DatatypeProperty rdf:ID="value">
    <daml:domain rdf:resource="#ServiceCategory"/>
    <daml:range  rdf:resource="&xsd;#string"/>
  </daml:DatatypeProperty>

  <daml:Class rdf:about="#ServiceCategory">
    <daml:comment>
    a ServiceCategory is restricted to refer to only one taxonomy
    </daml:comment>
    <rdfs:subClassOf>
      <daml:Restriction daml:cardinality="1">
<daml:onProperty rdf:resource="#value"/>
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:DatatypeProperty rdf:ID="code">
    <daml:domain rdf:resource="#ServiceCategory"/>
    <daml:range rdf:resource="&xsd;#string"/>
  </daml:DatatypeProperty>

  <daml:Class rdf:about="#ServiceCategory">
    <daml:comment>
    a ServiceCategory is restricted to refer to only one taxonomy
    </daml:comment>
    <rdfs:subClassOf>
      <daml:Restriction daml:cardinality="1">
<daml:onProperty rdf:resource="#code"/>
```

```
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:Class rdf:ID="ServiceParameter"/>

  <daml:DatatypeProperty rdf:ID="serviceParameterName">
    <daml:domain rdf:resource="#ServiceParameter"/>
    <daml:range rdf:resource="&xsd;#string"/>
  </daml:DatatypeProperty>

  <daml:Class rdf:about="#ServiceParameter">
    <daml:comment>
    A ServiceParameter should have at most 1 name (more precisely only
      one serviceParameterName)
    </daml:comment>
    <rdfs:subClassOf>
      <daml:Restriction daml:cardinality="1">
<daml:onProperty rdf:resource="#serviceParameterName"/>
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:ObjectProperty rdf:ID="sParameter">
    <daml:domain rdf:resource="#ServiceParameter"/>
    <daml:range  rdf:resource="&daml;#Thing"/>
  </daml:ObjectProperty>

  <daml:Class rdf:about="#ServiceParameter">
    <daml:comment>
    a Parameter is restricted to refer to only one concept in some
    ontology
    </daml:comment>
    <rdfs:subClassOf>
      <daml:Restriction daml:cardinality="1">
<daml:onProperty rdf:resource="#sParameter"/>
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:Class rdf:ID="QualityRating"/>

  <daml:DatatypeProperty rdf:ID="ratingName">
    <daml:domain rdf:resource="#QualityRating"/>
    <daml:range rdf:resource="&xsd;#string"/>
  </daml:DatatypeProperty>
```

```
<daml:ObjectProperty rdf:ID="rating">
  <daml:domain rdf:resource="#QualityRating"/>
  <daml:range  rdf:resource="&daml;#Thing"/>
</daml:ObjectProperty>

<daml:Class rdf:ID="Actor">
  <daml:label>Actor</daml:label>
  <rdfs:subClassOf rdf:resource="&daml;#Thing" />
  <daml:comment>
    Actor represents a Requester or Provider who might request or offer a service.
  </daml:comment>
</daml:Class>

<daml:DatatypeProperty rdf:ID="name">
  <daml:domain rdf:resource="#Actor"/>
  <daml:range rdf:resource="&xsd;#string"/>
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="title">
  <daml:domain rdf:resource="#Actor"/>
  <daml:range rdf:resource="&xsd;#string"/>
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="phone">
  <daml:domain rdf:resource="#Actor"/>
  <daml:range rdf:resource="&xsd;#string"/>
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="fax">
  <daml:domain rdf:resource="#Actor"/>
  <daml:range rdf:resource="&xsd;#string"/>
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="email">
  <daml:domain rdf:resource="#Actor"/>
  <daml:range rdf:resource="&xsd;#string"/>
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="physicalAddress">
  <daml:domain rdf:resource="#Actor"/>
  <daml:range rdf:resource="&xsd;#string"/>
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="webURL">
  <daml:domain rdf:resource="#Actor"/>
```

```
    <daml:range rdf:resource="&xsd;#string"/>
  </daml:DatatypeProperty>

  <daml:Class rdf:ID="NAICS">
    <daml:comment>
      Hook to the NAICS taxonomy
    </daml:comment>
    <rdfs:subClassOf rdf:resource="#ServiceCategory"/>
    <rdfs:subClassOf>
      <daml:Restriction>
<daml:onProperty rdf:resource="#categoryName"/>
<daml:hasValue>
  NAICS
</daml:hasValue>
      </daml:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <daml:Restriction>
<daml:onProperty rdf:resource="#taxonomy"/>
<daml:hasValue>
  www.naics.com
</daml:hasValue>
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:Class rdf:ID="UNSPSC">
    <daml:comment>
      Hook to the UNSPSC taxonomy
      </daml:comment>
    <rdfs:subClassOf rdf:resource="#ServiceCategory"/>
    <rdfs:subClassOf>
      <daml:Restriction>
<daml:onProperty rdf:resource="#categoryName"/>
<daml:hasValue>
   UNSPSC
</daml:hasValue>
      </daml:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <daml:Restriction>
<daml:onProperty rdf:resource="#taxonomy"/>
<daml:hasValue>
  www.un-spsc.net
</daml:hasValue>
      </daml:Restriction>
```

```
    </rdfs:subClassOf>
  </daml:Class>

  <daml:Class rdf:ID="MaxResponseTime">
    <rdfs:subClassOf rdf:resource="#ServiceParameter"/>
    <rdfs:subClassOf>
      <daml:Restriction>
<daml:onProperty rdf:resource="#sParameter"/>
<daml:toClass rdf:resource="#Duration"/>
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:Class rdf:ID="AverageResponseTime">
    <rdfs:subClassOf rdf:resource="#ServiceParameter"/>
    <rdfs:subClassOf>
      <daml:Restriction>
<daml:onProperty rdf:resource="#sParameter"/>
<daml:toClass rdf:resource="#Duration"/>
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:Class rdf:ID="Duration" />

  <daml:Class rdf:ID="GeographicRadius">
    <rdfs:subClassOf rdf:resource="#ServiceParameter"/>
    <rdfs:subClassOf>
      <daml:Restriction>
<daml:onProperty rdf:resource="#sParameter"/>
<daml:toClass rdf:resource="&country;#Country"/>
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:Class rdf:ID="DAndBRating">
    <rdfs:subClassOf rdf:resource="#QualityRating"/>
    <rdfs:subClassOf>
      <daml:Restriction>
<daml:onProperty rdf:resource="#ratingName"/>
<daml:hasValue>
          <xsd:string rdf:value="Dun and Bradstreet Rating"/>
</daml:hasValue>
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>
```

```
<daml:Class rdf:ID="OfferedService">
  <daml:label>OfferedService</daml:label>
  <rdfs:subClassOf rdf:resource="&service;#ServiceProfile"/>
</daml:Class>

<daml:Class rdf:ID="NeededService">
  <daml:label>NeededService</daml:label>
  <rdfs:subClassOf rdf:resource="&service;#ServiceProfile"/>
</daml:Class>

<daml:Class rdf:ID="ServiceRequester">
  <daml:label>ServiceRequester</daml:label>
  <rdfs:subClassOf rdf:resource="#Actor" />
  <daml:comment>
    ServiceRequester provides general contract details such as address, fax etc.
  </daml:comment>
</daml:Class>

 <daml:Class rdf:ID="ServiceProvider">
  <daml:label>ServiceProvider</daml:label>
  <rdfs:subClassOf rdf:resource="#Actor" />
  <daml:comment>
    ServiceProvider provides general contract details such as address, fax etc.
  </daml:comment>
</daml:Class>

<daml:Property rdf:ID="serviceType">
  <daml:domain rdf:resource="#Profile"/>
  <daml:range rdf:resource="&daml;#Thing"/>
</daml:Property>

<daml:Property rdf:ID="intendedPurpose">
  <daml:domain rdf:resource="#Profile"/>
  <daml:range rdf:resource="&daml;#Thing"/>
</daml:Property>

<daml:ObjectProperty rdf:ID="role">
  <daml:domain rdf:resource="#Profile"/>
  <daml:range rdf:resource="#Actor"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="requestedBy">
  <daml:subPropertyOf rdf:resource="#role" />
  <daml:range rdf:resource="#ServiceRequester"/>
</daml:ObjectProperty>
```

```
  <daml:ObjectProperty rdf:ID="providedBy">
    <daml:subPropertyOf rdf:resource="#role" />
    <daml:range rdf:resource="#ServiceProvider"/>
  </daml:ObjectProperty>

  <daml:Property rdf:ID="domainResource">
    <daml:domain rdf:resource="#Profile"/>
    <daml:range rdf:resource="&daml;#Thing"/>
  </daml:Property>

  <daml:ObjectProperty rdf:ID="geographicRadius">
    <daml:domain rdf:resource="#Profile"/>
    <daml:range rdf:resource="#Location"/>
  </daml:ObjectProperty>

  <daml:Class rdf:ID="Location">
    <daml:label>Location</daml:label>
    <rdfs:subClassOf rdf:resource="&daml;#Thing" />
    <daml:comment>
This class represents the scope or availability
of a service to some area.
    </daml:comment>
  </daml:Class>

  <daml:Property rdf:ID="degreeOfQuality">
    <daml:domain rdf:resource="#Profile"/>
    <daml:range rdf:resource="&daml;#Thing"/>
  </daml:Property>

  <daml:Property rdf:ID="qualityGuarantee">
    <daml:domain rdf:resource="#Profile"/>
    <daml:range rdf:resource="&daml;#Thing"/>
  </daml:Property>

  <daml:Property rdf:ID="communicationThru">
    <daml:domain rdf:resource="#Profile"/>
    <daml:range rdf:resource="&daml;#Thing"/>
  </daml:Property>

</rdf:RDF>
```

APPENDIX D

DAML-S Tourism Mini-Ontology

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!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 daml "http://www.daml.org/2001/03/daml+oil">
  <!ENTITY xsd "http://www.w3.org/2000/10/XMLschema.xsd">
  <!ENTITY time "http://www.ai.sri.com/daml/ontologies/time/Time.daml">
  <!ENTITY profile "http://www.daml.org/services/daml-s/0.7/Profile.daml">
  <!ENTITY DEFAULT "http://www.daml.org/services/damls/0.7/ProfileHierarchy.daml">
]>
<rdf:RDF
  xmlns:rdf= "&rdf;#"
  xmlns:rdfs= "&rdfs;#"
  xmlns:daml= "&daml;#"
  xmlns:xsd=  "&xsd;#"
  xmlns:profile= "&profile;#"
  xmlns= "&DEFAULT;#">

  <daml:Ontology>
    <daml:versionInfo>
      $MiniTravelOntology.daml v0.01 by K.Alpay Erturkmen $
    </daml:versionInfo>
    <daml:comment>
      A sample ontology developed by K.Alpay Erturkmen
    </daml:comment>
    <daml:imports rdf:resource="&rdf;" />
    <daml:imports rdf:resource="&daml;" />
    <daml:imports rdf:resource="&profile;" />
```

```
  </daml:Ontology>

  <daml:Class rdf:ID="HotelReservationService">
    <rdfs:subClassOf rdf:resource="&profile;#Profile" />
    <daml:comment>
Generic Hotel Reservation Service
    </daml:comment>
    <rdfs:subClassOf>
      <daml:unionOf>
        <daml:Restriction>
          <daml:onProperty rdf:resource="&profile;#succedingGenericService" />
          <daml:hasValue rdf:resource="#PaymentService" />
        </daml:Restriction>
        <daml:Restriction>
          <daml:onProperty rdf:resource="&profile;#succedingGenericService" />
          <daml:hasValue rdf:resource="#FlightDiscoveryService" />
        </daml:Restriction>
      </daml:unionOf>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:DatatypeProperty rdf:ID="hotelArrivalDate">
    <daml:subPropertyOf rdf:resource="&profile;#input" />
    <daml:domain rdf:resource="#HotelReservationService" />
    <daml:range rdf:resource="#ArrivalDate"/>
  </daml:DatatypeProperty>

  <daml:class rdf:ID="ArrivalDate">
    <daml:subClassOf rdf:resource="#StringParameter" />
  </daml:class>

  <daml:DatatypeProperty rdf:ID="hotelDepartureDate">
    <daml:subPropertyOf rdf:resource="&profile;#input" />
    <daml:domain rdf:resource="#HotelReservationService" />
    <daml:range rdf:resource="#DepartureDate"/>
  </daml:DatatypeProperty>

  <daml:class rdf:ID="DepartureDate">
    <daml:subClassOf rdf:resource="#StringParameter" />
  </daml:class>

  <daml:DatatypeProperty rdf:ID="hotelNoOfPeople">
    <daml:subPropertyOf rdf:resource="&profile;#input" />
    <daml:domain rdf:resource="#HotelReservationService" />
    <daml:range rdf:resource="#NoOfPeople"/>
  </daml:DatatypeProperty>
```

```
  <daml:class rdf:ID="NoOfPeople">
    <daml:subClassOf rdf:resource="#StringParameter" />
  </daml:class>

  <daml:DatatypeProperty rdf:ID="hotelReservationId">
    <daml:subPropertyOf rdf:resource="&profile;#output" />
    <daml:domain rdf:resource="#HotelReservationService" />
    <daml:range rdf:resource="#ReservationId"/>
  </daml:DatatypeProperty>

  <daml:class rdf:ID="ReservationId">
    <daml:subClassOf rdf:resource="#StringParameter" />
  </daml:class>

  <daml:Class rdf:ID="HotelDiscoveryService">
    <rdfs:subClassOf rdf:resource="&profile;#Profile" />
    <daml:comment>
Generic Hotel Discovery Service
    </daml:comment>
    <rdfs:subClassOf>
      <daml:unionOf>
        <daml:Restriction>
          <daml:onProperty rdf:resource="&profile;#succedingGenericService" />
          <daml:hasValue rdf:resource="#HotelReservationService" />
        </daml:Restriction>
        <daml:Restriction>
          <daml:onProperty rdf:resource="&profile;#succedingGenericService" />
          <daml:hasValue rdf:resource="#HotelInfoService" />
        </daml:Restriction>
      </daml:unionOf>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:DatatypeProperty rdf:ID="hotelDiscoveryArrivalDate">
    <daml:subPropertyOf rdf:resource="&profile;#input" />
    <daml:domain rdf:resource="#HotelDiscoveryService" />
    <daml:range rdf:resource="#ArrivalDate"/>
  </daml:DatatypeProperty>

  <daml:DatatypeProperty rdf:ID="hotelDiscoveryDepartureDate">
    <daml:subPropertyOf rdf:resource="&profile;#input" />
    <daml:domain rdf:resource="#HotelDiscoveryService" />
    <daml:range rdf:resource="#DepartureDate"/>
  </daml:DatatypeProperty>
```

```
  <daml:DatatypeProperty rdf:ID="hotelDiscoveryNoOfPeople">
    <daml:subPropertyOf rdf:resource="&profile;#input" />
    <daml:domain rdf:resource="#HotelDiscoveryService" />
    <daml:range rdf:resource="#NoOfPeople"/>
  </daml:DatatypeProperty>

  <daml:DatatypeProperty rdf:ID="hotelDiscoveryCity">
    <daml:subPropertyOf rdf:resource="&profile;#input" />
    <daml:domain rdf:resource="#HotelDiscoveryService" />
    <daml:range rdf:resource="#City"/>
  </daml:DatatypeProperty>

  <daml:DatatypeProperty rdf:ID="discoveryHotelId">
    <daml:subPropertyOf rdf:resource="&profile;#output" />
    <daml:domain rdf:resource="#HotelDiscoveryService" />
    <daml:range rdf:resource="#HotelId"/>
  </daml:DatatypeProperty>

  <daml:class rdf:ID="HotelId">
    <daml:subClassOf rdf:resource="#StringParameter" />
  </daml:class>

  <daml:class rdf:ID="City">
    <daml:subClassOf rdf:resource="#StringParameter" />
  </daml:class>

  <daml:Class rdf:ID="HotelInfoService">
    <rdfs:subClassOf rdf:resource="&profile;#Profile" />
    <daml:comment>
Generic Hotel Info Service
    </daml:comment>
    <rdfs:subClassOf>
        <daml:Restriction>
          <daml:onProperty rdf:resource="&profile;#succedingGenericService" />
          <daml:hasValue rdf:resource="#HotelReservationService" />
        </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:DatatypeProperty rdf:ID="infoHotelId">
    <daml:subPropertyOf rdf:resource="&profile;#input" />
    <daml:domain rdf:resource="#HotelInfoService" />
    <daml:range rdf:resource="#HotelId"/>
  </daml:DatatypeProperty>

  <daml:DatatypeProperty rdf:ID="infoHotelReservationServiceId">
```

```
    <daml:subPropertyOf rdf:resource="&profile;#output" />
    <daml:domain rdf:resource="#HotelInfoService" />
    <daml:range rdf:resource="#HotelReservationServiceId"/>
  </daml:DatatypeProperty>

  <daml:class rdf:ID="HotelReservationServiceId">
    <daml:subClassOf rdf:resource="#StringParameter" />
  </daml:class>

  <daml:DatatypeProperty rdf:ID="infoHotelInfo">
    <daml:subPropertyOf rdf:resource="&profile;#output" />
    <daml:domain rdf:resource="#HotelInfoService" />
    <daml:range rdf:resource="#HotelInfo"/>
  </daml:DatatypeProperty>

  <daml:class rdf:ID="HotelInfo">
    <daml:subClassOf rdf:resource="#StringParameter" />
  </daml:class>

  <daml:Class rdf:ID="PaymentService">
    <rdfs:subClassOf rdf:resource="&profile;#Profile" />
    <daml:comment>
Generic Payment Service
    </daml:comment>
    <rdfs:subClassOf>
        <daml:Restriction>
          <daml:onProperty rdf:resource="&profile;#succedingGenericService" />
          <daml:hasValue rdf:resource="#PostalService" />
        </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:DatatypeProperty rdf:ID="paymentReservationId">
    <daml:subPropertyOf rdf:resource="&profile;#input" />
    <daml:domain rdf:resource="#PaymentService" />
    <daml:range rdf:resource="#ReservationId"/>
  </daml:DatatypeProperty>

  <daml:DatatypeProperty rdf:ID="paymentBillId">
    <daml:subPropertyOf rdf:resource="&profile;#output" />
    <daml:domain rdf:resource="#PaymentService" />
    <daml:range rdf:resource="#BillId"/>
  </daml:DatatypeProperty>

   <daml:class rdf:ID="BillId">
    <daml:subClassOf rdf:resource="#StringParameter" />
```

```
  </daml:class>

  <daml:Class rdf:ID="PostalService">
    <rdfs:subClassOf rdf:resource="&profile;#Profile" />
    <daml:comment>
Generic Postal Service
    </daml:comment>
  </daml:Class>

  <daml:DatatypeProperty rdf:ID="postalBillId">
    <daml:subPropertyOf rdf:resource="&profile;#input" />
    <daml:domain rdf:resource="#PostalService" />
    <daml:range rdf:resource="#BillId"/>
  </daml:DatatypeProperty>

  <daml:DatatypeProperty rdf:ID="postalDestinationAdress">
    <daml:subPropertyOf rdf:resource="&profile;#input" />
    <daml:domain rdf:resource="#PostalService" />
    <daml:range rdf:resource="#DestinationAddress"/>
  </daml:DatatypeProperty>

  <daml:class rdf:ID="DestinationAddress">
    <daml:subClassOf rdf:resource="#StringParameter" />
  </daml:class>

  <daml:class rdf:ID="StringParameter">
  </daml:class>
  <daml:DatatypeProperty rdf:ID="value">
    <daml:domain rdf:resource="#StringParameter" />
    <daml:range rdf:resource="&xsd;#String" />
  </daml:DatatypeProperty>
```

APPENDIX E

Generated BPEL4WS Document

```
<process name="process"
             targetNamespace="http://euclid.ii.metu.edu.tr/~kalpaye/bpel/process.bpel"
             xmlns:tns="http://euclid.ii.metu.edu.tr/~kalpaye/bpel/process.bpel"
             xmlns:slt="http://schemas.xmlsoap.org/ws/2003/03/service-link/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
             xmlns:Seek="http://euclid.ii.metu.edu.tr/~kalpaye/wsdl/SeekHotels.com.wsdl"
             xmlns:Worl="http://euclid.ii.metu.edu.tr/~kalpaye/wsdl/WorldofWonders.com.wsdl"
             xmlns:Heal="http://euclid.ii.metu.edu.tr/~kalpaye/wsdl/Health-onHotel.wsdl"
             xmlns:Ph.D="http://euclid.ii.metu.edu.tr/~kalpaye/wsdl/Ph.DCard.wsdl"
             xmlns:WHOO="http://euclid.ii.metu.edu.tr/~kalpaye/wsdl/WHOOPSParcelService.wsdl"
>
  <slt:serviceLinkType name="SeekHotels.comSLT">
    <slt:role name="service">
      <slt:portType name="Seek:SeekHotels.comPT"/>
    </slt:role>
  </slt:serviceLinkType>
  <slt:serviceLinkType name="WorldofWonders.comSLT">
    <slt:role name="service">
      <slt:portType name="Worl:WorldofWonders.comPT"/>
    </slt:role>
  </slt:serviceLinkType>
  <slt:serviceLinkType name="Health-onHotelSLT">
    <slt:role name="service">
      <slt:portType name="Heal:Health-onHotelPT"/>
    </slt:role>
  </slt:serviceLinkType>
  <slt:serviceLinkType name="Ph.DCardSLT">
    <slt:role name="service">
```

```
      <slt:portType name="Ph.D:Ph.DCardPT"/>
    </slt:role>
  </slt:serviceLinkType>
  <slt:serviceLinkType name="WHOOPSParcelServiceSLT">
    <slt:role name="service">
      <slt:portType name="WHOO:WHOOPSParcelServicePT"/>
    </slt:role>
  </slt:serviceLinkType>
  <partners>
    <partner name="SeekHotels.comPartner" serviceLinkType="SeekHotels.comSLT"/>
    <partner name="WorldofWonders.comPartner" serviceLinkType="WorldofWonders.comSLT"/>
    <partner name="Health-onHotelPartner" serviceLinkType="Health-onHotelSLT"/>
    <partner name="Ph.DCardPartner" serviceLinkType="Ph.DCardSLT"/>
    <partner name="WHOOPSParcelServicePartner" serviceLinkType="WHOOPSParcelServiceSLT"/>
  </partners>
  <variables>
    <variable name="SeekHotels.comInput" messageType="Seek:SeekHotels.comRequest"/>
    <variable name="SeekHotels.comOutput" messageType="Seek:SeekHotels.comReply"/>
    <variable name="WorldofWonders.comInput" messageType="Worl:WorldofWonders.comRequest"/>
    <variable name="WorldofWonders.comOutput" messageType="Worl:WorldofWonders.comReply"/>
    <variable name="Health-onHotelInput" messageType="Heal:Health-onHotelRequest"/>
    <variable name="Health-onHotelOutput" messageType="Heal:Health-onHotelReply"/>
    <variable name="Ph.DCardInput" messageType="Ph.D:Ph.DCardRequest"/>
    <variable name="Ph.DCardOutput" messageType="Ph.D:Ph.DCardReply"/>
    <variable name="WHOOPSParcelServiceInput" messageType="WHOO:WHOOPSParcelServiceRequest"/>
    <variable name="WHOOPSParcelServiceOutput" messageType="WHOO:WHOOPSParcelServiceReply"/>
  </variables>
<sequence name="sequence">
<sequence name="sequence0">
  <receive name="receive0" partner="SeekHotels.comPartner" portType="Seek:SeekHotels.comPT"
   operation="Seek:SeekHotels.comOper" variable="SeekHotels.comInput" createInstance="yes"/>
  <invoke name="invoke0" partner="SeekHotels.comPartner" portType="Seek:SeekHotels.comPT"
   operation="Seek:SeekHotels.comOper" inputVariable="SeekHotels.comInput"
outputVariable="SeekHotels.comOutput"/>
</sequence>
<flow>
<sequence>
<sequence name="sequence1">
  <receive name="receive1" partner="WorldofWonders.comPartner" portType="Worl:WorldofWonders.comPT"
   operation="Worl:WorldofWonders.comOper" variable="WorldofWonders.comInput" />
  <invoke name="invoke1" partner="WorldofWonders.comPartner" portType="Worl:WorldofWonders.comPT"
  operation="Worl:WorldofWonders.comOper" inputVariable="WorldofWonders.comInput"
   outputVariable="WorldofWonders.comOutput"/>
</sequence>
</sequence>
<sequence>
```

```
<sequence name="sequence2">
  <receive name="receive2" partner="Health-onHotelPartner" portType="Heal:Health-onHotelPT"
   operation="Heal:Health-onHotelOper" variable="Health-onHotelInput" />
  <invoke name="invoke2" partner="Health-onHotelPartner" portType="Heal:Health-onHotelPT"
   operation="Heal:Health-onHotelOper" inputVariable="Health-onHotelInput"
outputVariable="Health-onHotelOutput"/>
</sequence>
</sequence>
</flow>
<sequence name="sequence3">
  <receive name="receive3" partner="Ph.DCardPartner" portType="Ph.D:Ph.DCardPT"
   operation="Ph.D:Ph.DCardOper" variable="Ph.DCardInput" />
  <invoke name="invoke3" partner="Ph.DCardPartner" portType="Ph.D:Ph.DCardPT"
   operation="Ph.D:Ph.DCardOper" inputVariable="Ph.DCardInput" outputVariable="Ph.DCardOutput"/>
</sequence>
<sequence name="sequence4">
  <receive name="receive4" partner="WHOOPSParcelServicePartner" portType="WHOO:WHOOPSParcelServicePT"
   operation="WHOO:WHOOPSParcelServiceOper" variable="WHOOPSParcelServiceInput" />
  <invoke name="invoke4" partner="WHOOPSParcelServicePartner" portType="WHOO:WHOOPSParcelServicePT"
   operation="WHOO:WHOOPSParcelServiceOper" inputVariable="WHOOPSParcelServiceInput"
        outputVariable="WHOOPSParcelServiceOutput"/>
</sequence>
</sequence>
</process>
```

APPENDIX F

WSDL Documents

F.1 SeekHotels.com.wsdl

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types>
  <xsd:schema>
    <xsd:simpleType name="StringType">
      <xsd:restriction base="xsd:string">
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:schema>
</wsdl:types>
  <wsdl:message name="SeekHotels.comRequest">
    <wsdl:part name="City" type="#StringType"/>
    <wsdl:part name="FromDate" type="#StringType"/>
    <wsdl:part name="ToDate" type="#StringType"/>
    <wsdl:part name="#ofPersons" type="#StringType"/>
  </wsdl:message>
  <wsdl:message name="SeekHotels.comReply">
    <wsdl:part name="HotelID" type="#StringType"/>
  </wsdl:message>
  <wsdl:portType name="SeekHotels.comPT">
    <wsdl:operation name="SeekHotels.comOper">
      <wsdl:input message="#SeekHotels.comRequest" />
      <wsdl:output message="#SeekHotels.comReply" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SoapSeekHotels.com" type="#SeekHotels.comPTi">
```

```
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http">
      <wsdl:operation name="SeekHotels.comOper">
      <soap:operation soapAction=""/>
        <wsdl:input>
          <soap:body namespace="http://SeekHotels.com.com" use="literal"/>
        </wsdl:input>
      </wsdl:operation>
    </soap:binding>
  </wsdl:binding>
</wsdl:definitions>
```

F.2 WorldofWonders.com.wsdl

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types>
  <xsd:schema>
    <xsd:simpleType name="StringType">
      <xsd:restriction base="xsd:string">
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:schema>
</wsdl:types>
  <wsdl:message name="WorldofWonders.comRequest">
    <wsdl:part name="HotelID" type="#StringType"/>
  </wsdl:message>
  <wsdl:message name="WorldofWonders.comReply">
    <wsdl:part name="ServiceInstanceID" type="#StringType"/>
    <wsdl:part name="HotelInfo" type="#StringType"/>
  </wsdl:message>
  <wsdl:portType name="WorldofWonders.comPT">
    <wsdl:operation name="WorldofWonders.comOper">
      <wsdl:input message="#WorldofWonders.comRequest" />
      <wsdl:output message="#WorldofWonders.comReply" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SoapWorldofWonders.com" type="#WorldofWonders.comPTi">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http">
      <wsdl:operation name="WorldofWonders.comOper">
      <soap:operation soapAction=""/>
        <wsdl:input>
          <soap:body namespace="http://WorldofWonders.com.com" use="literal"/>
        </wsdl:input>
      </wsdl:operation>
    </soap:binding>
```

```
  </wsdl:binding>
</wsdl:definitions>
```

F.3 Health-onHotel.wsdl

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types>
  <xsd:schema>
    <xsd:simpleType name="StringType">
      <xsd:restriction base="xsd:string">
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:schema>
</wsdl:types>
  <wsdl:message name="Health-onHotelRequest">
    <wsdl:part name="FromDate" type="#StringType"/>
    <wsdl:part name="ToDate" type="#StringType"/>
    <wsdl:part name="#ofPersons" type="#StringType"/>
  </wsdl:message>
  <wsdl:message name="Health-onHotelReply">
    <wsdl:part name="HotelReservationID" type="#StringType"/>
  </wsdl:message>
  <wsdl:portType name="Health-onHotelPT">
    <wsdl:operation name="Health-onHotelOper">
      <wsdl:input message="#Health-onHotelRequest" />
      <wsdl:output message="#Health-onHotelReply" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SoapHealth-onHotel" type="#Health-onHotelPTi">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http">
      <wsdl:operation name="Health-onHotelOper">
      <soap:operation soapAction=""/>
        <wsdl:input>
          <soap:body namespace="http://Health-onHotel.com" use="literal"/>
        </wsdl:input>
      </wsdl:operation>
    </soap:binding>
  </wsdl:binding>
</wsdl:definitions>
```

F.4 Ph.DCard.wsdl

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

```
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types>
  <xsd:schema>
    <xsd:simpleType name="StringType">
      <xsd:restriction base="xsd:string">
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:schema>
</wsdl:types>
  <wsdl:message name="Ph.DCardRequest">
    <wsdl:part name="ReservationID" type="#StringType"/>
  </wsdl:message>
  <wsdl:message name="Ph.DCardReply">
    <wsdl:part name="BillID" type="#StringType"/>
  </wsdl:message>
  <wsdl:portType name="Ph.DCardPT">
    <wsdl:operation name="Ph.DCardOper">
      <wsdl:input message="#Ph.DCardRequest" />
      <wsdl:output message="#Ph.DCardReply" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SoapPh.DCard" type="#Ph.DCardPTi">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http">
      <wsdl:operation name="Ph.DCardOper">
      <soap:operation soapAction=""/>
        <wsdl:input>
          <soap:body namespace="http://Ph.DCard.com" use="literal"/>
        </wsdl:input>
      </wsdl:operation>
    </soap:binding>
  </wsdl:binding>
</wsdl:definitions>
```

F.5 WHOOPSParcelService.wsdl

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types>
  <xsd:schema>
    <xsd:simpleType name="StringType">
      <xsd:restriction base="xsd:string">
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:schema>
</wsdl:types>
```

```
  <wsdl:message name="WHOOPSParcelServiceRequest">
    <wsdl:part name="BillID" type="#StringType"/>
    <wsdl:part name="DestinationAddress" type="#StringType"/>
  </wsdl:message>
  <wsdl:message name="WHOOPSParcelServiceReply">
  </wsdl:message>
  <wsdl:portType name="WHOOPSParcelServicePT">
    <wsdl:operation name="WHOOPSParcelServiceOper">
      <wsdl:input message="#WHOOPSParcelServiceRequest" />
      <wsdl:output message="#WHOOPSParcelServiceReply" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SoapWHOOPSParcelService" type="#WHOOPSParcelServicePTi">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http">
      <wsdl:operation name="WHOOPSParcelServiceOper">
      <soap:operation soapAction=""/>
        <wsdl:input>
          <soap:body namespace="http://WHOOPSParcelService.com" use="literal"/>
        </wsdl:input>
      </wsdl:operation>
    </soap:binding>
  </wsdl:binding>
</wsdl:definitions>
```