University of Alberta

FUSION OF SEMANTIC WEB TECHNOLOGY WITH THE FUZZY INFERENCE ENGINE: WEB SERVICES AND ENTERPRISE APPLICATIONS

by

DengMing Li 

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**

Department of Electrical and Computer Engineering

Edmonton, Alberta

Spring 2005

UNIVERSITY OF ALBERTA

Library Release Form

Name of Author: DengMing Li

Title of Thesis: Fusion of Semantic Web Technology with the Fuzzy Inference Engine: Web Services and Enterprise Applications

Degree: Master of Science

Year this Degree Granted: 2005

_____

Date: April 06, 2005

Abstract

A paradigm of Semantic Web provides a new way to represent information on the web. This new approach is based on application of ontology and ontology languages. At the same time web services are the standard approach of accessing interactive application through the web. The idea of this thesis is to build an intelligent web service in the Semantic Web environment using a concept of fuzziness. To do this, fuzzy ontologies are defined to reflect the "human-like" vague statements and facts. These ontologies together with service-related information are used to reason about "goodness" of sevices. Approximate reasoning is used in this process. A number of different tools are used to implement the proposed approach: Protégé is used to build the ontologies, FuzzyJ to build the reasoner, and DAMLJessKB and Jena to parse the ontologies.

In the thesis, the proposed idea is illustrated by creating a fuzzy hotel reservation service. End user can submit the request to the hotel reservation service; the hotel reservation service will then response to the end user by returning hotels that are the best fit to his/her preferences.

The concept of fuzzy-based Semantic Web Services can be also used in the enterprise application. This thesis presents the vision on how to integrate those applications using innovated model-driven approach, called Collaborative Ontology Enterprise Planning (COEP). The ultimate goal of this solution is to achieve seamless integration of enterprise front-end and back-end systems over the web.

Acknowledgements

This thesis was instructed by Dr. Reformat, who gave me both academic and methodological guidance. The direct instruction from him is the foundation of the thesis. My wife, Diana Wang gave me a lot of supports while I'm composing the thesis. I'm also thankful for her love and support. I will express my thanks to Adaptrust Solutions Inc, who provides me a testing environment for metadata and other applications used in the thesis.

TABLE OF CONTENTS

TABLE OF FIGURES

A LIST OF ABBREVIATIONS

AP: Accounting Payable, an accounting term

AR: Accounting Receivable, an accounting term

B2B: Business to Business, a term of E-Commerce

B2C: Business to Consumer, a term of E-Commerce

BPEL: Business Process Execution Language

BPEL4WS: BPEL to Web Services

CCR: Cached Consulting Result, a term only used in the thesis

CLIPS: C Language Integrated Production System

COEP: Collaborative Ontology Enterprise Planning

COM: Comprehensive operation management

COTS: Commercial Of The Shelf

CPC: Collaborative Presenting Component, a term only used in the thesis

CRM: Customer Relationship Management

CU: Control Unit

DAML: DARPA Agent Markup Language

DAML+OIL: the combination of DAML and OIL

DAML-S: DAML enabled services

DL: Description Logic

ERP: Enterprise Resource Planning

FACT: Fast Classification of Terminologies, a DL classifier used for modal logic

HIA: Hotel Information Agent, a term only used in the thesis

HTML: Hyper-Text Markup Language

JESS: Java Expert System Shell

KPC: Knowledge Parsing Component, a term only used in the thesis

LISP: List Processor, an AI programming language created by John McCarthy in 1965

OIL: Ontology Inference Layer

OLAP: Online Analytical Process

OWL: Web Ontology Language

PM: Project Management

POP: Purchase Order Processing, an accounting term

RDF: Resource Description Framework

RDFS: RDF (see above) Schema

RPC: Remote Procedure Call

RSC: Reasoning Service Component, a term only used in the thesis

SCM: Supply Chain Management

SOAP: Simple Object Access Protocol, is the part of Web Services protocol

SOP: Sales Order Processing, an accounting term

UDDI: Universal Description, Discovery and Integration, is the part of Web Services protocol

UDU: User Database Unit, a term only used in the thesis

W3C: World Wide Web Consortium

WSDL: Web Services Description Language

XBRL: eXtensible Business Reporting Language

XML: eXtensible Markup Language

Chapter 1

Introduction

Section 1.1 World Wide Web: Today's Status

We are now "living" in the WWW (World Wide Web) society. There are enormous numbers of different resources visible and accessible in the society. While the number of WWW users is increasing, web resources are also increasing dramatically. Not only is the Internet used as the information repository, but also as a community where knowledge can be discovered and shared.

However, fast growth of the Internet is making the information "in a mess". Such a problem is caused by the nature of current web. Although the information on each web site is represented using the same HTML format, the content itself is arbitrary. This means, that people are facing with difficulties while extracting useful information from the enormous number of web pages. So far, the web has been designed only for human beings – an automatic discovery and utilization of web-stored information is yet to come.

The good news is that the Internet technology is on its way of revolution. Researchers from both academic and industry organizations are now exploring the new vision of web in order to have the Internet more usable to all different types of programs. The

enhancement does not only make Internet search quicker and more accurate, but also creates capabilities for interaction between the multiple devices, and integration between heterogeneous applications.

Section 1.2 World Wide Web: Future Trends

As explained in section 1.1, the next generation web technology aims at making web resources more readily accessible. The future web will be presented in explicit self-described model, changing its role from merely content presentation to object-oriented information web house. This approach is done by adding metadata (the data of data, see Chapter 2, section 2.1 for detail explanation) annotations that can help to explicitly describe the web contents.

By annotation, web resources will be more meaningful to software programs. Web contents can be shared by the explicitly defined expression within a specific domain. The knowledge can be presented in the way that machine can understand it. This simple change will make it possible to use computer software as delegate to perform web browsing and exploring knowledge in the certain domain. In order to do this, an inference engine has to be built as the core of the exploration services. There are many ways of building such service engine. We use the fuzzy-based method as its backbone-reasoning technique. In my thesis, a light-weigh framework of such reasoning services is prototyped. It utilizes the Semantic Web (the academic name of the next generation web technology, Chapter 3 provides the detail explanation of the Semantic Web.) as the environment.

Section 1.3 Motivation

The motivation of my work is the increasing demands of using the Semantic web. In order to make maximum use of the web, some services are built upon it. Those services can be used by both software agent and humans. The individual web service can be built as an inference machine.

We use fuzzy logic as the principle method of building such an inference engine. This allows for processing such vague statements as "I'd like to book a hotel which is not too far from the business center." Fuzzy rules can be inserted into service process model. Those web services are capable of providing real request-response service to the users. These users could be either human being or another type of such a web service.

Section 1.4 Contributions

The thesis focuses on the concept of the intelligent software infrastructure utilizing semantic web technology. The thesis represents a fairly early work of applying fuzzy techniques for building semantic web agents. We use DAML (DARPA Agent Markup Language) web services combined with fuzzy logic. A simple hotel reservation system is used to demonstrate the process of booking a hotel room using service agent with fuzziness. The booking is done based on user's information and the preferred profile. Thanks to that, the response of the system points to hotels that are best matches against the inquiry. In the case presented, the user of hotel reservation system is a human being, but it can be any other agent as well.

We proposed COEP (Collaborative Ontology Enterprise Planning) in order to utilize the concepts of ontology and fuzziness into enterprise applications. The principle of COEP is to compose the large business process software applications with built-in fuzzy-based reasoner and treat those applications as the semantic web services. Although in most cases, COEP model should be applied in the intranet of a company for security and operation reason, the components could be organized hierarchically to perform the global complex business process across the Internet as well.

Section 1.5 Thesis's Organization

The thesis starts with a short description of the concept of metadata and the definition of ontology, Chapter 2. Chapter 3 defines the concepts of Semantic Web, Web Services. Chapter 4 explains the core technologies needed to develop a prototype of an agent with

an inference engine – fuzziness and approximate reasoning. Next, Chapter 5 explores the combination of Semantic Web Services with fuzziness. With above background and knowledge, a prototype of reasoner is being built. In Chapter 6, a simple hotel reservation system is proposed from semantic matching point of view. Chapter 7 and Chapter 8 expand the concept to enterprise applications area. COEP (Collaborative Ontology Enterprise Planning) is proposed to address the interoperability of enterprise applications as web services. Conclusions are drawn in Chapter 9. Appendix gives the abbreviations used in the thesis and DAML specifications.

Chapter 2

Matadata

This chapter introduces the metadata concept. This is an important concept since metadata is the foundation of describing the property of data. Besides the description of the metadata itself, all other related technologies are explained in this chapter as well.

Section 2.1 Metadata

Metadata is familiar to most of librarians. Metadata is defined as data of data. It extents the general data structure, provides description about digital or non-digital resources for wide areas of operations, consists of complex constructs which are usually hard to maintain, such as dictionary management, database schema, analytical dimension in data warehouse etc [1].

So far metadata has a limited use on the Internet. However, the current trends indicate its extensive use in the future. An example of metadata is the Dublin Core. The Dublin Core is a type of metadata to represent the digital resources. It is widely used in author-generated description for Web pages. The following characteristics are the key of the Dublin Core:

1. Simplicity
2. Semantic interoperability

3. International consensus
4. Extensibility
5. Modularity

The Dublin Core targets resource description in wide range industries. It can express semantic information [2].

The following code is from www.adaptrust.com homepage using metadata with two components: name and content. It uses Dublin Core in description of Creator, DateCreated and Relation.

```
<META name="description" content="Semantic   Web, Web Service, KM,
Knowledge Management, Crystal Report, Web Report, Artificial
Intelligence, CRM, eCommerce">

<META name="keywords" content="Knowledge Management, AI,
ASP.NET,Semantic Web, Web Service, KM,Crystal Report, Web Report,
Artificial Intelligence, CRM, CTI, Message, Mobile, Mobility, SCM, BI,
Reasoner, Fuzzy Logic, eCommerce">

<META name="DC.Creator" content="Ronnie, Li" >
<META name="DC.Date.Created" content="2003-12-18" scheme="ISO8601"/>
<META name="DC.Relation.isPartOf"content="http://www.adaptrust.com"/>
```

Here is the explanation of the above code. There are five metadata definitions, each metadata includes two parts: Name and Content. "Description" is what the context is about; Keywords indicates the terms related to the content, it is used for searching index; DC.Creator is name of the creator; DC.Date.Created means the creation Date, which uses ISO8601 code; DC.Relation.IsPartOf indicates that this document is a part of the resource, in this case the resource is http://www.adaptrust.com web site [3].

Section 2.2 Ontology

2.2.1 Definition of Ontology

After years of research on knowledge engineering, ontology-based approach has been adopted as the most suitable representation of knowledge across the web. It is a formal, explicit specification of a shared conceptualization [4]. Ontology is a set of well-defined classes to describe data models in the specific domain. Ontology instances are working as knowledge characters to express the individual facts [5]. Ontologies are efficient to form

web knowledge bases by their indigenous ability to present interrelated resources. Thus their use is widely applied in web applications [6].

The term originates from philosophy, where Ontology is a systematic indication of existence. For AI systems, "exists" means things can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects, and the describable relationships among them, are reflected in the representational vocabulary inside knowledge-based system. Thus, in the context of AI, we can describe the ontology of a program by defining a set of representational terms. In such ontology, definitions associate the names of entities in the universe of discourse (e.g., classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms.

Figure 2.1 indicates different levels of ontology. Basic ontology is the most generic and common, this lowest level is managing the basic concept or things which do not specify the context. Domain ontology is more concrete, it deals with the certain type of domain that needs knowledge engineers to fulfill. Application ontology is the highest level that requires both knowledge engineers and system engineers to construct them.

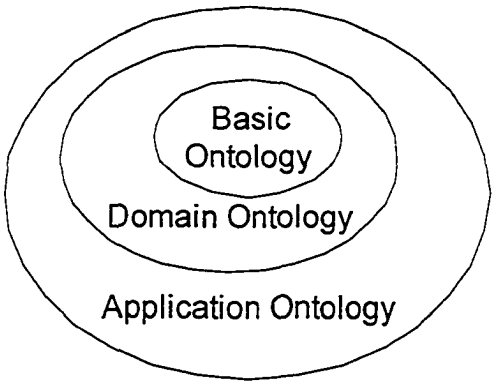

Figure 2.1: Hieratical Ontology Layers

2.2.2 Ontology as Knowledge Representation

Knowledge Representation is one of fundamental components in an expert system. It enables an entity to determine consequences by thinking rather than acting. Recently knowledge workers made big progress on modeling the knowledge. There is extensive involvement by some leading technical companies of variety of industries. In order to turn data into shared knowledge over the web, how to model the data efficiently and semantically is becoming more important than how to display them nicely. As we know that knowledge can be represented by natural language, computer descriptive resources also can model it. Theses resources include verbal, graphics or data grid.

Ontology plays an important role in knowledge representation. Ontology providing shared and common domain models is a key component of the semantic web. For example, ontology-based reasoning services can use semantics to provide transparently connected services. By defining the relationship between the form and the content of information, ontologies assist people and machines in assessing, processing and communicating information.

Section 2.3 Ontology Language

2.3.1 XML

XML (eXtensible Markup Language) is all about metadata and the idea that certain groups of people have similar needs for describing and organizing the data they use. Like HTML, XML is a set of tags and declarations. However, it focuses on providing information about the data itself and how it relates to other data rather than concerns with formatting information on a page,

Some data types are quite common, such as Name, PhoneNumber and Title. Others are industry or organization-oriented, such as in retail management, UnitPrice, Markup , ProductCategory are used frequently. XML allows each of these data types to be easily formatted for both engineer and users.

XML differs from HTML in three major respects:

1. Information providers can define new tag and attribute names at will.
2. Document structures can be nested to any level of complexity.
3. Any XML document can contain an optional description of its grammar for use by applications that need to perform structural validation.

XML is not backwards compatible with existing HTML documents, but documents conforming to HTML3.2 can easily be converted to XML, as can generic SGML documents and documents generated from databases [7].

XML has been used to encode many different types of information and one of its strengths lies in the ability of any XML-aware software to process any XML file. However, for some applications it is important that additional information is provided to allow for other relationships between data to be established.

XML is a subset of SGML (Standard Generalized Markup Language, an international standard for the definition of device-independent, system-independent methods of representing texts in electronic form) constituting a particular text markup language for interchange of structured data. XML was invented and maintained by the World Wide Web Consortium [8].

2.3.2 RDF/RDFS

Resource Description Framework (RDF) is a standard for describing resources and information on the web. Resource Description Framework Schema (RDFS) is used as an ontology language supporting exchange of knowledge over the web.

There is a need to modal relationships between atomic entities. It will form the basis of a processing model that, it is hoped, will lead to better machine processing of information in a networked environment.

RDF is a foundation for processing metadata; it provides interoperability between applications that exchange machine-understandable information on the Web. RDF uses

XML to exchange descriptions of Web resources but the resources being described can be of any type, including XML and non-XML resources. RDF emphasizes facilities to enable automated processing of Web resources. RDF can be used in a variety of application areas, for example: in resource discovery to provide better search engine capabilities, in cataloging for describing the content and content relationships available at a particular Web site, page, or digital library, by intelligent software agents to facilitate knowledge sharing and exchange, in content rating, in describing collections of pages that represent a single logical "document", for describing intellectual property rights of Web pages, and for expressing the privacy preferences of a user as well as the privacy policies of a Web site. RDF with digital signatures will be key to building the "Web of Trust" for electronic commerce, collaboration, and other applications.

Let's have a look at how a relationship is expressed normally. Obviously the common way is by the natural language used by human, for instance, "subject→predicate→ object (SPO)" mode

An example of this is the English sentences: "An author has a name and also has an email address". Figure 2.2 shows normal expression with SPO in diagraph

1. Author→Has→Name
2. Author→Has→Email Address

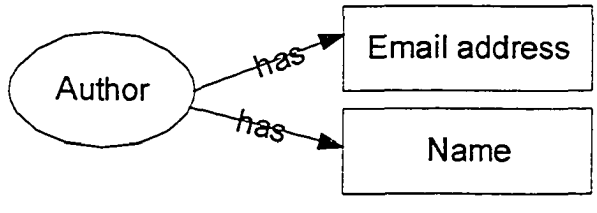

Figure 2.2: Normal expression with SPO in digraph

A more detailed example is the description of web store. This is a simple term net with semantic architecture. Figure 2.3 is a web store diagraph with indication of relationship between eCommerce

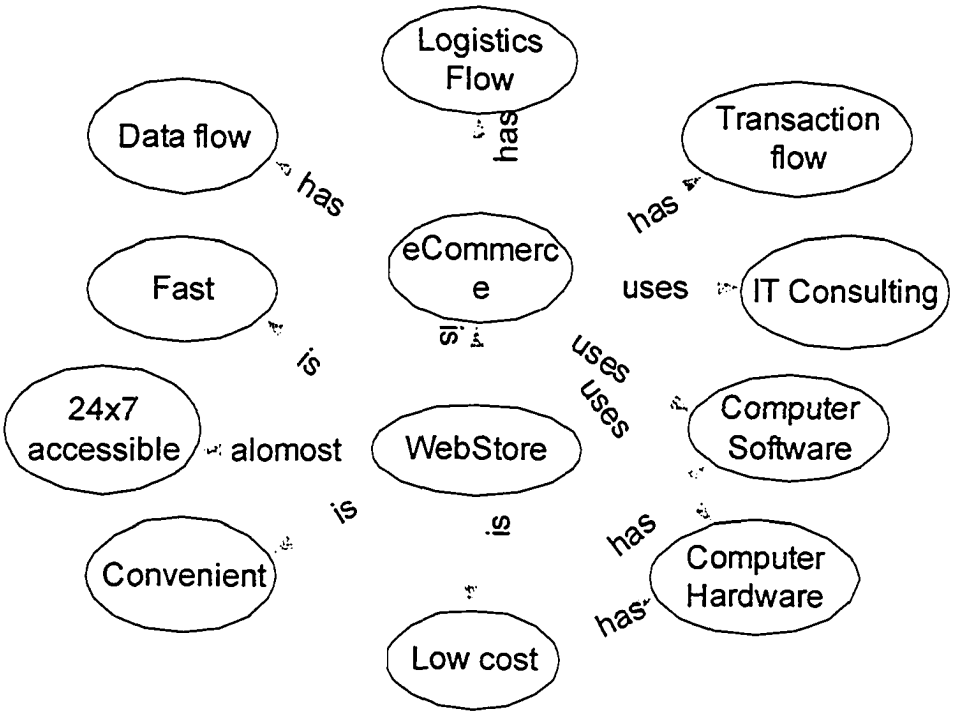

Figure 2.3: A simple web store term net

The way of representing labeled digraphs in a computer is with RDF (Resource Description Framework) [9]. It is recommended by the W3C. W3C is the abbreviation of World Wide Web Consortium; this organization is responsible for activities related standardization of the World Wide Web. The sample relationship shown in Figure 2-2 can be specified in RDF format in the following way [10].

```
<rdf:RDF>
<rdf:Description about="http://www.adaptrust.com/Home/RonnieLi">
<s:Creator
rdf:resource="http://www.adaptrust.com/humanresources/employeeid/ee_002
"/>
</rdf:Description>
<rdf:Description
about="http://www.adaptrust.com/humanresources/employeeid/ee_002">
<v:Name>Ronnie Li</v:Name>
<v:EmailAddress>ronnie-li@usa.net</v:EmailAddress >
</rdf:Description>
</rdf:RDF>
```

2.3.3 DAML+OIL

A combination of DARPA Agent Markup Language (DAML) and Ontology Inference Layer (OIL) enables the creation of ontologies for any domain and the instantiation of these ontologies in the description of specific web sites. DAML+OIL enhances and extends RDFS with richer modeling primitives [11]. DAML+OIL is an ontology description logic language. DAML+OIL described web resources more accessible to automated processes and integrated with Web Services.

2.3.4 OWL

The latest web resource ontology language is Web Ontology Language (OWL), which has been proposed as the recommendation by W3C [12]. OWL is not only used to represent information on the web, but also improves the capability to process the information and increases the interoperability among software agents [13]. DAML+OIL is used as the language to build fuzzy service ontology which is upgradeable to OWL with a corresponding parser.

2.3.5 Ontology editing tools

2.3.5.1 Protégé 2000

Protégé is an open source ontology editor built in Java, targeting at the easy use of building ontology or class schema in knowledge management system [14].

Protégé 2000 is a tool that can be used to:

1. Construct a domain ontology
2. Customize data entry forms
3. Enter data / or class

Protégé 2000 is useful in constructing knowledge-based systems. It provides a visual tool to build any kind of ontology, reducing much time in validating ontology and relationship between classes. Protégé now provides support for OWL ontology editing. For detail information about protégé 2000, please reference the its official website from Stanford University [15].

2.3.5.2 OilED

OilEd is another ontology editor which allows user to build ontologies in DAML+OIL format. OilED uses FaCT as built-in reasoner [16]. OilED was developed by Sean Bechhofer and and Gary Ng of the University of Manchester. For detail information about OilED, please reference the its official website from University of Manchester [17].

Chapter 3

Semantic Web

This chapter introduces the Semantic Web related terms. Section 1 provides an overview of Semantic web, Section 2 introduces the Web Services and finally Section 3 explains the combination of Section 1 and Section 2: Semantic Web Services.

Section 3.1 Semantic Web

Advances in Artificial Intelligence in the area of knowledge representation have led to the formation of the Semantic Web which is the representation of resources on the World Wide Web [18]. The Semantic Web is virtually a hub of linked information that can be accessed and operated by programs. These programs can be in a form of software agents or any other applications which are capable of handing the semantics. The concept of Semantic Web was introduced in May 2001 in Scientific American by Tim Berners-Lee, James Hendler, and Ora Lassila [19]. The initiative of propagating the Semantic Web is trying to solve the problem of increasing demand on information sharing, eCommerce, and intelligent searching over the Internet.

The structure of the Semantic Web is shown in Figure 3.1 [20]. The current web contains Hyper Text Markup Language (HTML) files with data hidden in them. Extensible

Markup Language (XML) is a good start of representing data on the web; XML's namespace is represented with XSD (XML Schema Definition). XML mainly works on the syntax level. Web data often have different meaning in different contexts. To address this issue, the Semantic Web uses RDF and RDFS as the basic methodology of expressing web resources in the form of triples: resource, subject, and property. The higher level ontology languages DAML+OIL and OWL are introduced upon RDF(S). They are more powerful to represent the semantics of resources and information. Within the Semantic Web, a search process will become more intelligent since the entities involved in it will know what the search means. There are other elements of the Semantic Web such as Proof, Trust and Digital Signature to provide security mechanisms within the framework of the Semantic Web.

There has been a growing interest for the Semantic Web in Europe and the U.S. in recent years. The area is now emerging as an academic field and there is considerable interest from industry, such as IBM, HP, BEA, SAP, Microsoft and so on.

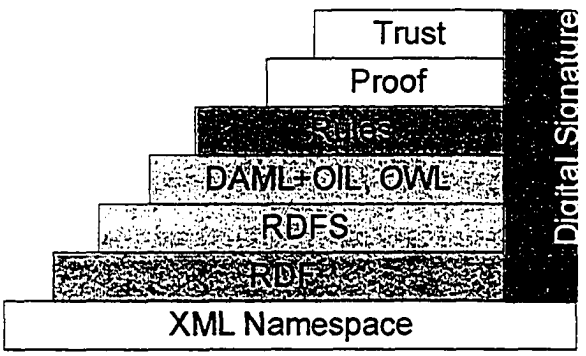

Figure 3.1: Structure of Semantic Web

Section 3.2 Services Description

What's the "services"? Generally speaking, services usually involve just two components: service requester and service provider.

In case of web, services can be provided in text, multimedia, and raw data format. The following example (see Figure 3.2) shows the traditional structure of a travel information

website that provides hotel reservation, car rental and shopping over the Internet. Firstly web user knows where those services can be found in certain website. Then he/she sends request to the provider by filling the form or simply performs the keyword searches. After that, he gets several results from the website according to his query.

The traditional services assume that they deal with human beings only. There is a gap between request-response talk, which was handled mainly by the user. In this scenario, user is more active than the services provider.

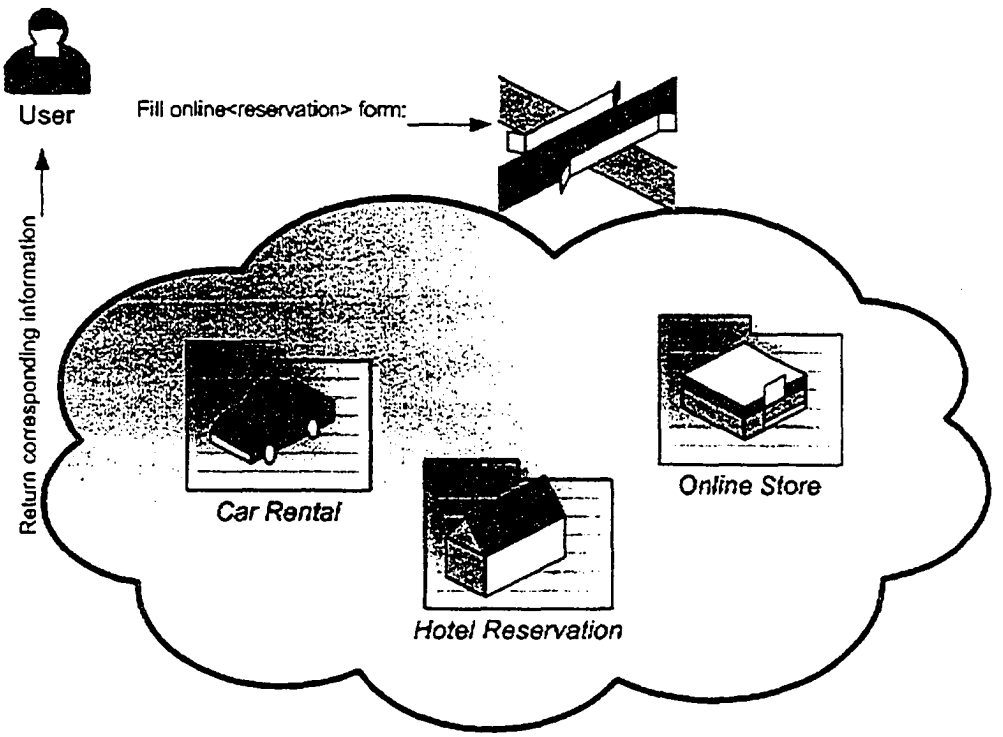

Figure 3.2: Travel information website

To improve the interoperability, many B2B, B2C applications are used as the routing bridge between user and provider (Figure 3.3). Those applications are so called web crawler because they know the structure of the certain web pages. The crawler understands what the user wants and what the website can provide, therefore they work perfectly in the static annotation webpage inside which contents are mostly represented in the fixed style. There is drawback in this solution. <ABC Online Crawler> will not work if the website changes its structure. This is because it does not understand the request and service at the semantic level.

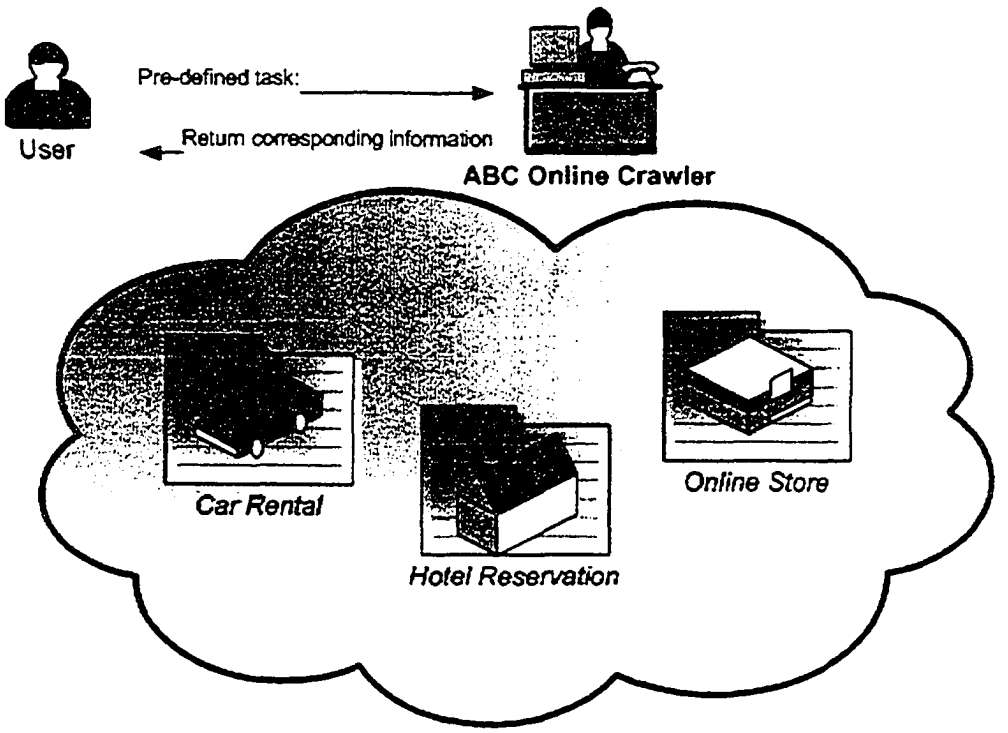

Figure 3.3: Travel information website with crawler

In order to fully achieve the interoperability, we have to use Semantic Web Services in which the resource, property, object and interface are encoded in an explicit way that make those information and communication understandable by machines (Figure 3.4). We explained the XML Web Services above; Semantic Web Services is addressing the markup of not only the content but also the Web Services on website to make it available for software agent exploitation. Semantic web service focus on represent the service in semantic web environment. This is a combination of web services with knowledge representation, utilizing ontology concept. The current work on Semantic Web Services is DMAL-S.

Section 3.3 Web Services

Web services are new breed of web application. It is also called XML Web Services. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes. Once a Web service is

deployed, other applications (and other Web services) can discover and invoke the deployed service.

The basic platform is XML plus HTTP. HTTP is a ubiquitous protocol, running practically everywhere on the Internet. XML provides a metalanguage in which you can write specialized languages to express complex interactions between clients and services or between components of a composite service. Behind the facade of a web server, the XML message gets converted to a middleware request and the results converted back to XML.

Web Services include three protocols: SOAP, UDDI, and WSDL. Below is a brief description of the platform elements. It should be noted that while service providers try to present the emergent web services platform as coherent, it's really a series of in-development technologies. Often at the higher levels there are, and may remain, multiple approaches to the same problem.

- SOAP (remote invocation)

  SOAP – is the abbreviation of Simple Object Access Protocol, SOAP is a communications protocol which present message in XML format (SOAP Message), similar to Remote Procedure Calls (RPC). The main idea of RPC is that the software programmer can transparently call a function from a machine in a network, as invoke on the local machine. SOAP protocol indicates how to send and receive SOAP messages. SOAP messages usually contain functions or methods with parameter and return values.

  E.g. a [sendmail] web service SOAP

  The following is a sample SOAP request and response. The placeholders shown need to be replaced with actual values.

```
POST /WebService/MailService/MailSrv.asmx HTTP/1.1
Host: www.adaptrust.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "DOEP.WS/SendMail"

The above code shows SOAP content header. MailSrv.asmx is the
service name, the service resides in server www.adaptrust.com. It
```

```
indicates the service action is to send an email using DOEP.WS
namespace.

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <SendMail xmlns="DOEP.WS">
      <From>string</From>
      <To>string</To>
      <Subject>string</Subject>
      <Message>string</Message>
      <fileName>string</fileName>
    </SendMail>
  </soap:Body>
</soap:Envelope>
```

The above XML document shows the wrapped envelope using namespace: http://schemas.xmlsoap.org/soap/envelope/. The envelope has function called SendMail. SendMail has five elements:

1. From
2. To
3. Subject
4. Message
5. FileName

The following code shows the return type of the function.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <SendMailResponse xmlns="DOEP.WS">
      <SendMailResult>string</SendMailResult>
    </SendMailResponse>
  </soap:Body>
</soap:Envelope>
```

- UDDI (Universal Description, Discovery and Integration) UDDI specifies the technical foundation for discovery of Web services across the Internet.

- WSDL (Web Services Description Language) WSDL is written in XML format, WSDL is capable of describing the content of an Internet services.

Web services have ability to extend programs to perform their task with less human intervention. In my research work I will combine semantic web with web service. Using a combination of web pointers, web markup, and ontology languages, service descriptions can be enriched by including a machine-readable description of how the service runs and some explicit logic describing the consequences of consuming the service.

Example: [SendMail] WSDL

Web service URL: (web service file name is MailSvr.asmx , it is hosted on server: www.adaptrust.com, it can be accessed via http from any Internet browser, MailSvr.asmx?WSDL means we will display the WSDL file content) http://www.adaptrust.com/XMLWebServices/MailService/MailSrv.asmx?WSDL

```
<?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:s0="DOEP.WS"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    targetNamespace="DOEP.WS"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  - <types>
    - <s:schema elementFormDefault="qualified"
        targetNamespace="DOEP.WS">
      - <s:element name="SendMail">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1"
                name="Message" type="s:string" />
              <s:element minOccurs="0" maxOccurs="1"
                name="fileName" type="s:string" />
            </s:sequence>
          </s:complexType>
        </s:element>
      - <s:element name="SendMailResponse">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1"
                name="SendMailResult" type="s:string"
                />
            </s:sequence>
          </s:complexType>
        </s:element>
      </s:schema>
    </types>
  - <service name="OntoWeb">
```

```
    <documentation>DOEP's public webservices</documentation>
  - <port name="OntoWebSoap" binding="s0:OntoWebSoap">
      <soap:address
        location="http://www.adaptrust.com/WebService/Mail
        Service/MailSrv.asmx" />
    </port>
  </service>
</definitions>
```

The above introduces the basic explanation of Web services and its three components: SOAP, UDDI and WSDL. Next section I will go into the research topic: build such Web Service that can provide online reasoning I order to help people exploring the Internet content.

Section 3.4 Semantic Web Services

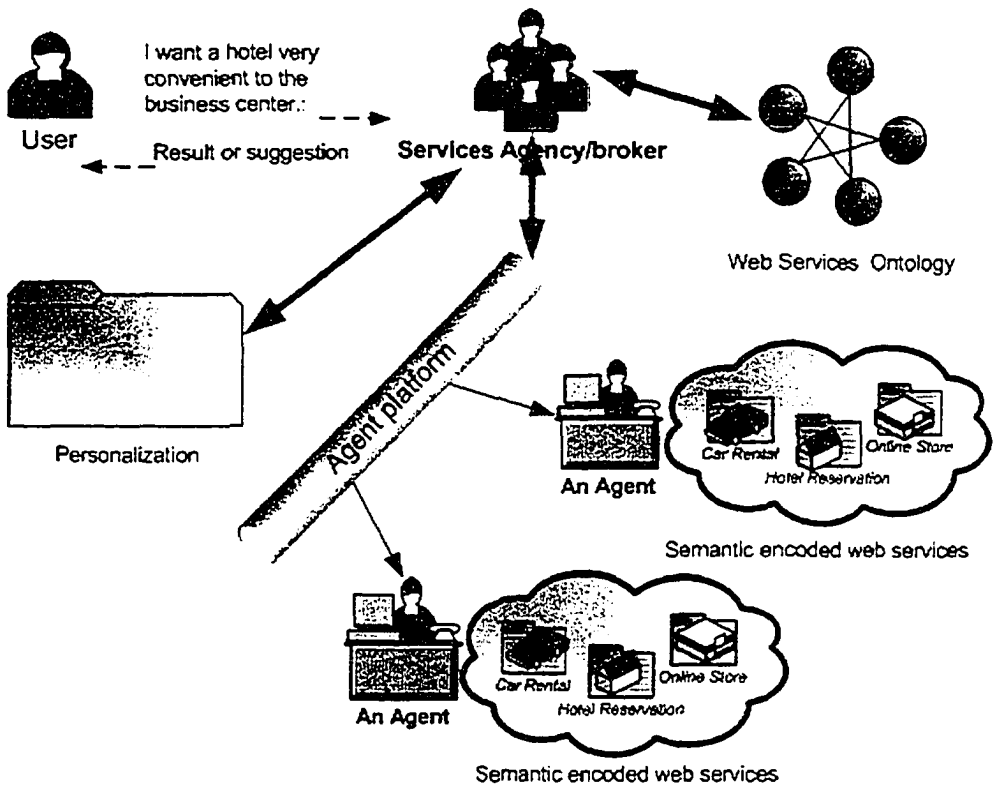

Figure 3.4: Travel information website with Semantic web service

The Semantic Web should enable users to locate, select, employ, compose, and monitor Web-based services automatically [21]. Semantic Web Services is built on some ontology language such DAML+OIL, OWL because of those languages' inner capability of representing complex relationship between entities inside webpages. The fact of building

Semantic Web Services is the process of rendering services using ontology. Figure 3.5 shows the Top level of the semantic web 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.

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.

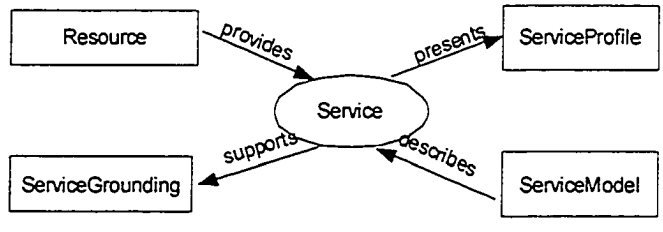

Figure 3.5: Top level of the semantic web service

A service grounding (``grounding'' for short) specifies the details of how an agent can access a service. Typically grounding will specify a communications protocol (e.g., RPC, HTTP-FORM, CORBA IDL, SOAP, Java RMI, OAA ACL [22]), 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). The likelihood is that a relatively small set of groundings will come to be widely used in conjunction with DAML services. Groundings will be specified at various well-known URIs (URIs is the abbreviation of

Uniform Resource Identifiers. URIs contain strings that identify the path of resources in the web, these resources include text files, images, or executables and so on.

In general, the *ServiceProfile* provides the information needed for an agent to discover a service. Taken together, the *ServiceModel* and *ServiceGrounding* objects associated with a service provide enough information for an agent to make use of a service.

DAML-S (DAML-S is the abbreviation of DAML-Services) is an attempt to provide ontology, within the framework of the DARPA Agent Markup Language, for describing Web services. It will enable users and software agents to automatically discover, invoke, compose, and monitor Web resources offering services, under specified constraints. The initial version of DAML-S was released at the DARPA [23].

Web resources can be presented more precisely and intensively. The contents on the website are accessible and understandable not only by both machine and human, but also by web services.

The semantic encoding of resources, properties, objects and interfaces makes them understandable by machines. The introduction of XML Web Services has greatly enhanced an interaction between distributed applications. However, human beings still need to discover the related Web Services and know their profile before using them. As shown in Figure 3.6 (extends figure 3.5, with indication of functions of service components) [24], the Semantic Web Services is addressing description - ServiceProfile, process - ServiceModel, composition, and grounding of Web Services what makes the service available for software agent exploitation. This is a combination of web services with knowledge representation, utilizing the ontology concept. DAML Service (DAML-S) can describe the Semantic Web Services by using DAML+OIL. The current work on Semantic Web Services is OWL-based Web Service Ontology (OWL-S) as enhanced version of DAML-S.

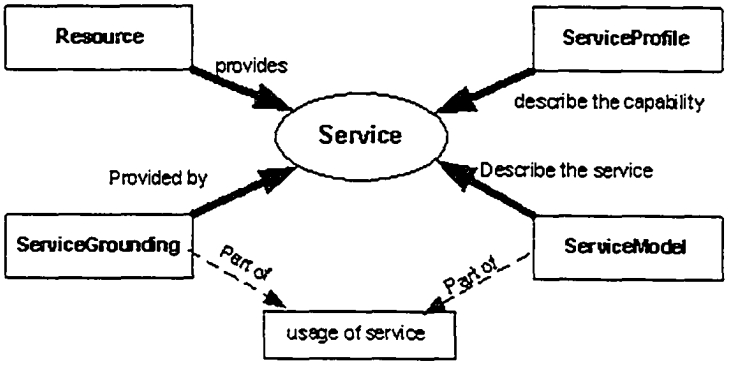

Figure 3.6: Upper Ontology of Services

In Figure 3.6, each item is defined as follows:

- Resource: any entity to be represented on the web
- ServiceProfile: provides a high-level description of a service and its provider
- ServiceModel: describes how a service works
- ServiceGrounding: describes how a service can be accessed

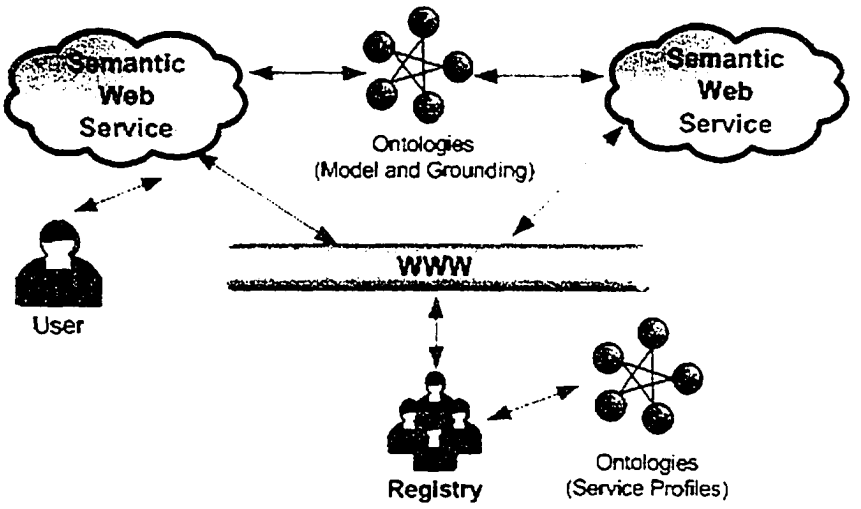

Figure 3.7: Interaction between Semantic Web Services

Figure 3.7 shows the interaction between Semantic Web Services. A single Semantic Web Service can locate others services based-on their ServiceProfiles in the registry. After that, all services are able to interact with each other through the ServiceModel and ServiceGrounding.

Chapter 4

Fuzzy Logic, Sets and Systems

As Fuzzy approach was used in the thesis as the inference method. In this chapter, the basic Fuzzy Logic and Fuzzy Set concept has been explored in Section 1. Section 2 introduces the Approximate Reasoning.

Section 4.1 Basic Concepts

Being one of three components of SoftComputing, fuzzy sets play an important role in representing ambiguous and inexact information. Especially, they are ideal for expressing individual's imprecise opinions related to making decisions regarding acceptance or not of results of different services [25].

Pointed out by Fuzzy theory creator Dr.Lofti Zadeh, crisp Boolean values do not work well in many cases when people try to express things naturally, especially when there are incomplete or imprecise raw data. Fuzzy logic and fuzzy sets have been theorized and widely used in academic and industry wherever that imprecise information is held. Fuzzy logic is used for knowledge representation and approximate reasoning. In a number of engineering domain, for example control, expert system and modeling [26].

Fuzzy logic is widely used in rule-based system. Rule-based systems are largely applied in decision-making, computer control systems budgeting and forecasting in the scenario with much of the imprecise, uncertain, and ambiguous and inexact knowledge.

People may feel that fuzziness is quite far away from them. However, "fuzzy" information is often involved in human thinking and reasoning because human beings have the ability of matching similar rather than identical experience in their patterns. In the case of classical logic (two-valued logic), it is very difficult to answer some questions that they do not have completely true answers. Humans, on the other hand, could handle that kind of questions pretty well.

Fuzziness

Fuzziness applies to where the border of related information is not clear-cut. For example, concepts such as *salty*, *well-done*, or *high* are in the fuzzy world. There is no defined quantitative value that represents the term *high temperature*. For example, for some people, 28°C is *high*, and for others, temperature 22°C is also *high*. As matter of fact the concept *temperature* has no clean border. However we know that generally temperature -10°C is definitely *low* and temperature 39°C is definitely high; however, temperature 16°C has some possibility of being *high* and some possibility of being low depending on the context. Unlike classical set theory where one deals with objects whose membership to a set can be clearly described, in fuzzy set theory membership of an element to a set can be partial, i.e., an element belongs to a set with a certain grade (possibility) of membership. More formally a fuzzy set A in a universe of discourse U is characterized by a membership function $mA : U \rightarrow [0,1]$ which associates with each element x of U a number mA(x) in the interval [0,1] which represents the grade of membership of x in the fuzzy set A.

For example, the fuzzy term ***high temperature*** might be defined by fuzzy set in Table 1.

Table 1: Fuzzy Term *high temperature*

Temperature (°C)	Grade of Membership
3	0.0

5	0.1
7	0.2
9	0.3
13	0.4
16	0.5
18	0.6
25	0.7
29	0.8
33	0.9
36	1.0

We can apply the formula to above fuzziness in format as: mHighTemperature (36) = 1, mhighTemperature (33) = 0.9, mhighTemperature (5) = 0.1... , mhighTemperature (3) = 0

Grade of membership values constitute a possibility distribution of the term highTemperature. The table can also be shown by diagram, see Figure 4.1.

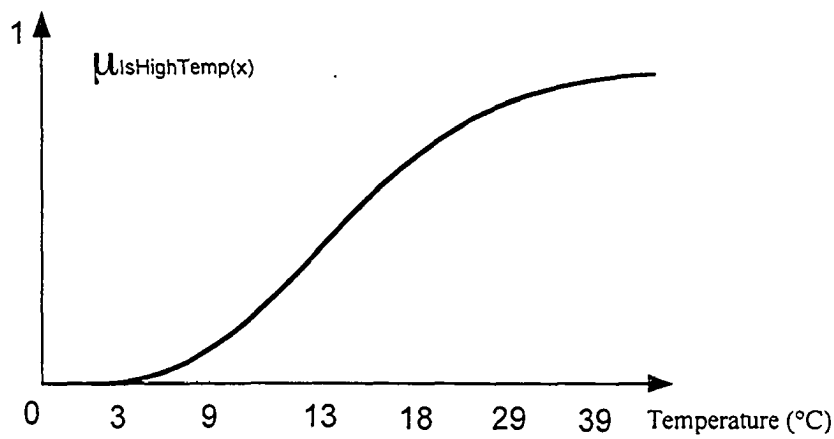

Figure 4.1: Grade of membership of HighTemperature

Uncertainty and fuzzy set

Uncertainty happens when one is not absolutely certain about some information. The degree of uncertainty can be represented by a crisp numeric value from {0,1}, where a

certainty factor of 1 means that the system is very certain that a fact is true, and a certainty factor of 0 mean that it is very uncertain that a fact is true. In fuzzy logic, instead of using set {0,1}, the degree of uncertainty can be expressed as a set membership function that can take on values in the interval [0,1].

Section 4.2 Approximate Reasoning

Approximate Reasoning is used in such scenario that incomplete or inaccurate information are given. This feature makes Approximate Reasoning a good candidate to facilitate the knowledge base engineering. In this circumstance, instead of response to the request is not a form of yes or no, an approximate answer will be given as response. This is different with the classical query-answer in software engineering. An important advantage is that approximate answers can be more "precise" than the classical yes-no answers.

Fuzzy logic deals with things that are vague, uncertain or probabilistic, as they inherently exist in the real world. Fuzzy system works more like human mind rather than two-value way, it applies the certain value of such a system that true or false answer can't be determined simply by its input. So fuzzy system focus on inexact concepts and similarity matching. Classical set system focus on scientific calculation, enterprise information system, banking transactions etc, in which exact precondition can be provided and corresponding rules can be fired precisely. Whereas fuzzy set systems have its vast applications on intelligent control unit, reasoning, customer behavior predicting, speech recognition, semantic searching and so on.

The approximate reasoning methodology is applied as the fundamental part of the reasoning service. As we mentioned in beginning of this thesis, fuzzy set, neural networks and probabilistic reasoning forms the soft computing. Variety of expert system can be built upon fuzzy logic. Fuzzy system has the capability of draw conclusion or

suggestion base on incomplete and inexact information by presenting users with certain degree.

The main difference between fuzziness and probability is that fuzziness is deterministic uncertainty while probability is non-deterministic. Fuzziness is more concerned with the degree to which events occur instead of likelihood of occurrence [27].

Fuzzy Rules

Rule is the basic element in rule-based expert system. Rule is the manipulation of predicates. Usually the rule is defined by domain knowledge experts. The definition of rule should reflect the business or process as accurate as possible. Rules are made up of a series of antecedents connected by logical operator (AND).

The number of rules depends on how many fuzzy terms you have in the system. In theory it is the combination of values of those factors. For example, the valve operation relates with two factors: Temperature and Humidity. Temperature has three linguist terms: High | Medium | Low, Humidity has two linguist terms: High | Low. So we will have the following rules:

Rule 1: IF Temperature is High AND Humidity is High THEN OpenValve

Rule 1: IF Temperature is Medium AND Humidity is High THEN OpenValve

Rule 1: IF Temperature is Low AND Humidity is High THEN OpenValve

Rule 1: IF Temperature is High AND Humidity is Low THEN CloseValve

Rule 1: IF Temperature is Medium AND Humidity is Low THEN CloseValve

Rule 1: IF Temperature is Low AND Humidity is Low THEN CloseValve

So the number of rule = NoOfMem(F1)* NoOfMem(F2)* NoOfMem(F2)* NoOfMem(F....)* NoOfMem(FN) , NoOfMem(Fi) is the number of values for F

Fuzzification and Defuzzification

Execution of a set of fuzzy rules includes sequence of fuzzification and defuzzification. Figure 4.3 shows the process of fuzzy inference (including both fuzzification and defuzzification). The crisp inputs values are firstly fuzzified to fulfill the FuzzyValue

antecedents (LHS) before producing a set of FuzzyValue outputs (RHS). The outputs can be processed as of defuzzification to get non-fuzzy values to perform the real control action.

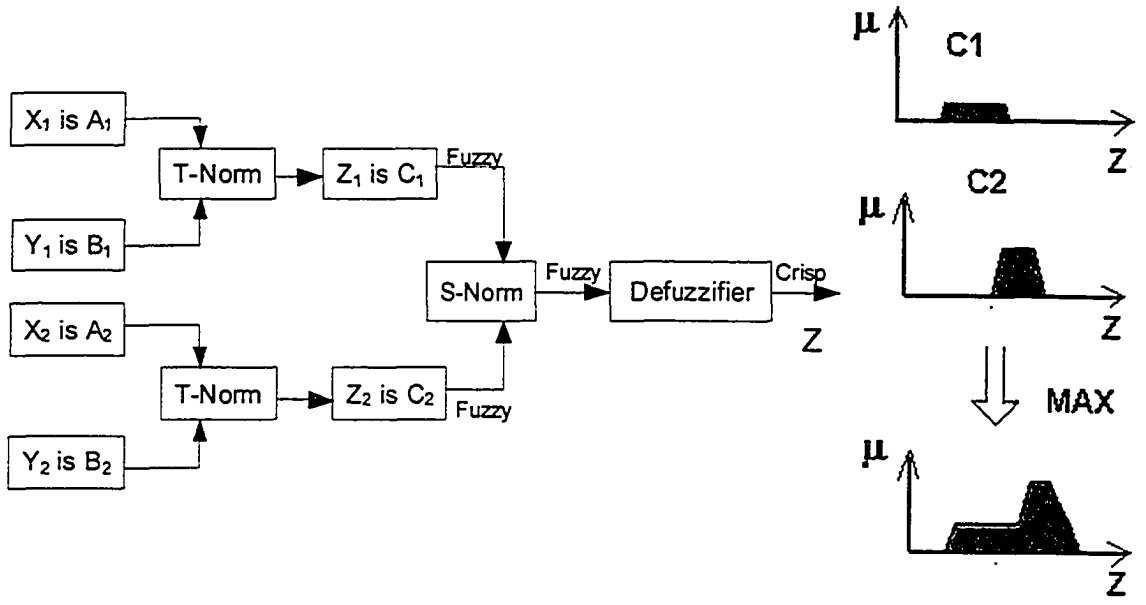

Figure 4.2: Fuzzy inference process

There are mainly two common defuzzification techniques are the Centre of Gravity (also called CENTROID, centroid of area) and MEANOFMAX methods. In the CENTROID method, the crisp value of the output variable is computed by finding the variable value of the center of gravity of the membership function for the fuzzy value. CENTROIDI method is shown in figure 4.4. In the MEANOFMAX method, the crisp output is the average of the maximizing values at which the membership function reaches the maximum degree.

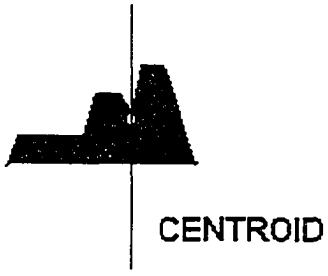

Figure 4.3: Centre of gravity defuzzification

More regarding fuzziness and execution will be given in Chapter 5: Semantic Web Services with Fuzziness.

Chapter 5

Semantic Web Services with Fuzziness

In this chapter, we will explain the concept and architecture of a semantic web service that supports fuzziness. The semantic web service was built as an agent who understands user's preference and work on behalf of the user automatically.

Section 5.1 Concept

The initiative of representing Web Services in the way that they are machine-understandable creates a new era for software agent* interoperability. A recent introduction of a concept of Semantic Web makes it possible to automatically locate, discover, composite, and execute the services. User agent works on behalf of the user, and knows user's preferences. As there might be many service providers on the web, finding the best one which will match user's preference is critical for performance and trust of agent, especially in the situation that user is dealing with many choices and wants to make a decision. The overall of proposed concept is shown in Figure 5.1.

***About of the term "Agent" and "Service" used in the thesis**

The term "agent" is to emphasis the whole system is running in certain type of agent framework, such as FIPA proposed. All agents communicate via mutual understandable protocol. We test User Agent scenario in which it will register itself into agent controller, and then talk to another registered agent who can provide certain request->response functionality. In order to simply the whole system implementation, we use "agent" as "services", it could be any programming thread running in the OS. In our case, they are running on the web server.

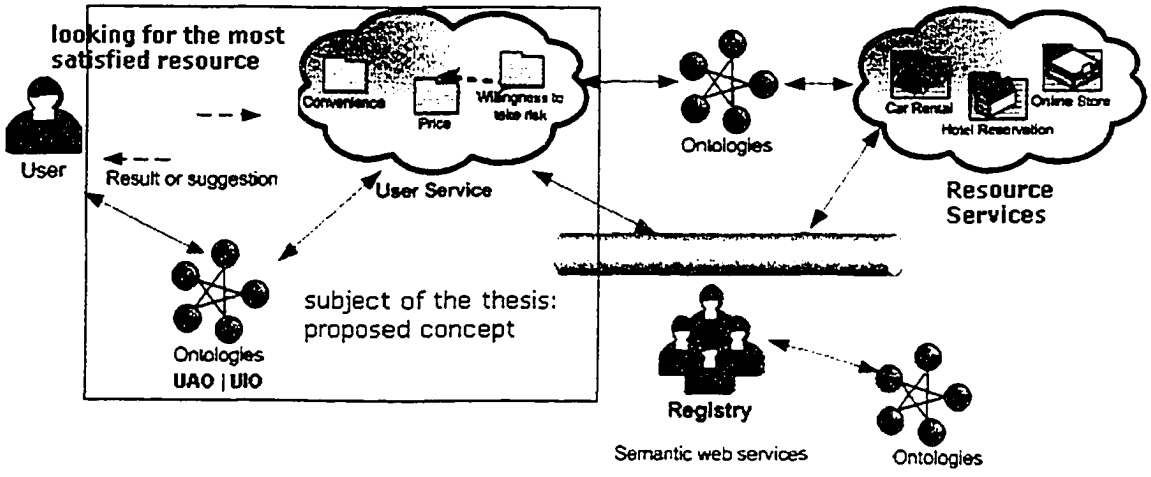

Figure 5.1: Overall of the proposed concept

In order to accomplish presented concept, a fuzzy logic has been used to implement an agent for semantic web service. The agent uses a simple fuzzy reasoner to perform the intelligent service in the Semantic Web environment. Its intelligence means the capability of finding the desired result, which can match against the user's preferences. The architecture of the agent is generic enough, so any kind of searching, depending on the domain knowledge and user's personal requirement, can be performed.

Section 5.2 Architecture

5.2.1 General Overview

A special set of tasks assigned to the agent and semantic web environment have had an influence on the architecture of the agent. The concept includes three modules: User Interface (UI), Resources Service and Ontology. System components are illustrated in Figure 5.2.

User Interface:
User Interface provides a dialog that users can input there searching criteria to reflect his or her preference. The interface can be either traditional user graphic application or webpage.

Resources Service

Resource service performs as the parser of resource ontology instances although it can also do some registry location. For example, hotel is one kind of resource, it physically exists in certain area in a city and they have different characters. Resource service is capable of communicating with hotel registrar who serves as the repository of resources. Resource service should also talk to the User service to pass a given resource information into user agent for further process.

Ontology

The resource information must include all that user acceptance ontologies. For example, in order to matching against user's AcceptanceInPrice, resource ontology should at least have price include in the instance. There are two approaches to solve the absence problem. One way to do this is providing an exception report and then skips the reasoning process; the other way that frequently used is to assign zero or a very low value as an unavailable indicator. For example, if hotel ontology instance does not provide information that whether it has car service or not, user services assume that such service is not available for this hotel.

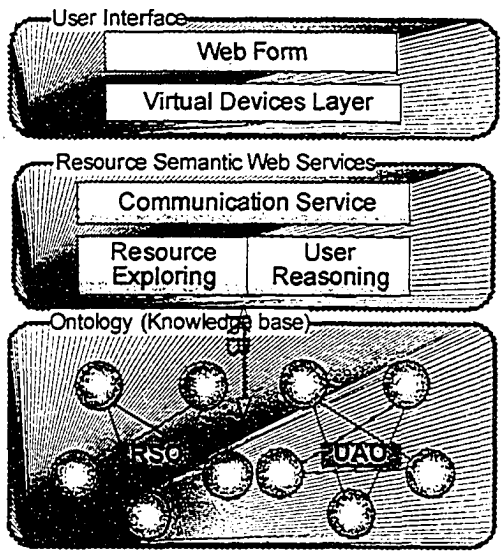

Figure 5.1: Three modules of the concept

5.2.2 Infrastructure Layers

According to above described pattern, we built the three-tier fuzzy resources searching system. Figure 5.3 shows the 3-tier structure: User Interface layer, Services layer and Storage layer.

1. User Interface

   User Interface layer is the Graphic User Interface to the end user, user can make the request and get the response via this interface.

2. Services

   Services handles the business logic process, there are Communication Unit, Knowledgebase Parser and Reasoner in this layer. Their functions are described in the following section.

3. Storage

   Storage layer includes both database (DB) and ontologies (ON). Database is mainly used for caching ontologies includes both resource schema and instances.

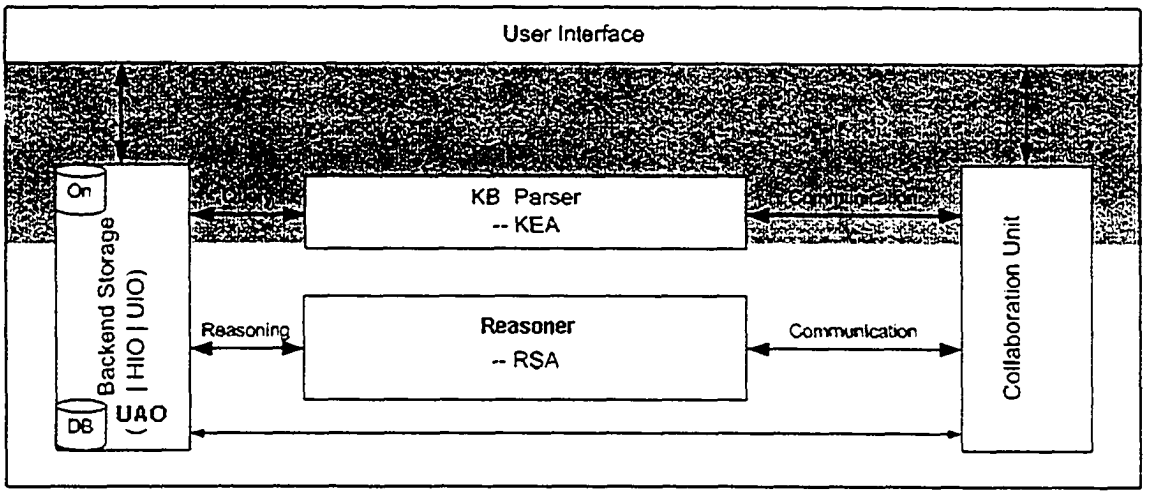

Figure 5.3: Three-tier architecture view

5.2.3 Detail Implementation of the Services

As explained above, there are four components; they are called CPC, KPC, RSC and RIC. RIC is the boundary between the system and resource registration service on the Internet.

1. Collaborative Presenting Component (CPC): Transfer users' request to KPC, send reasoning result back to users, CPC is a presenting interface application, CPC is for communication purpose
2. Knowledge Parsing Component (KPC): To parse the DAML+OIL document and passing the result (facts and rules) to Reasoning Service Component, KPC can query/parse *.daml files
3. Reasoning Service Component (RSC): Reasoner is the core part of the system which applying fuzzy rules inside. RSC does inference based on fuzzy logic and ontology facts to reflect user's preference and personality, RSC was implemented using FuzzyJ toolkit.
4. Resource Information Component (RIC): Collect resource information and exchange information with other agent system. All information retrieved will be stored in KB in the format that RA can understand, based on the Resource Information Ontology

5.2.3.1 CPC: Communication Presentation Component

The structure of CPC is shown in Figure 5. 4. User has to register to the system. The User Registrar saves user's information in User Database Unit (UDU). After registration, user can submit a request regarding the desired resource. CPC forwards the request information to KPC and RSC for further parse and inference; CPC returns the inference result to the user.

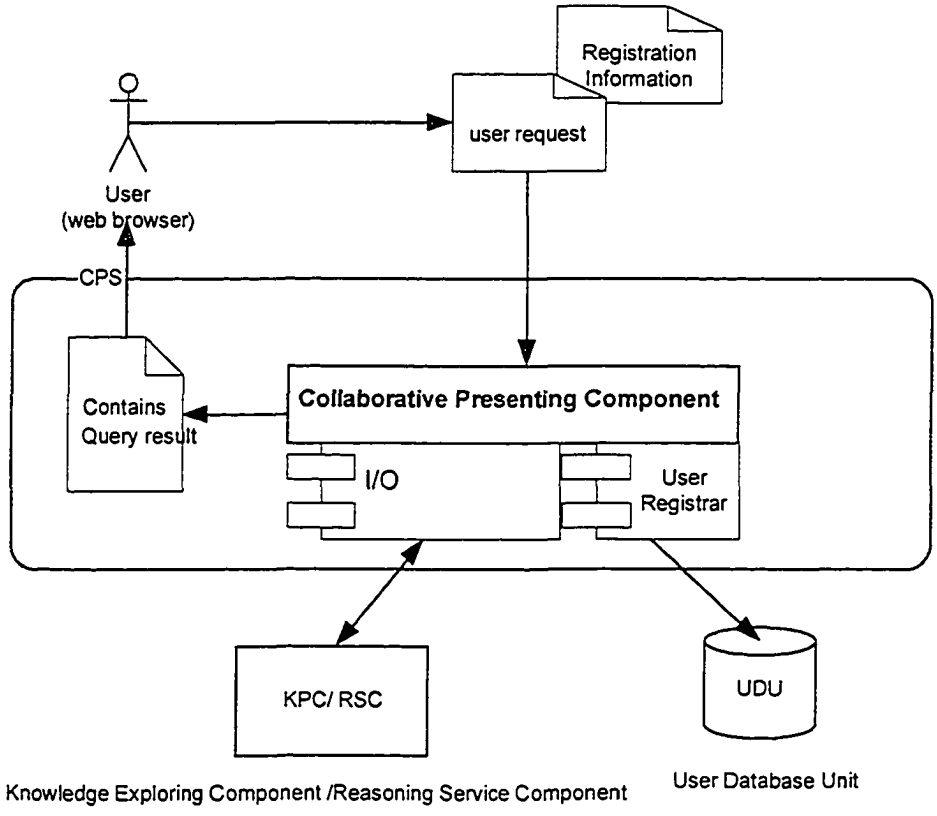

Figure 5.2: Structure of CPC

5.2.3.2 KPC: Knowledge Parsing Component

The structure of KPC is shown in Figure 5.5. KPC has two sub-components, Caching Services and DMAL+OIL Parser. Caching Service is used for quick search. If one user has submitted query before, KPC can simply return the result (Cached Consulting Result, called CCR) through the Caching Service. If it is a new request, KPC DMAL+OIL Parser will explore the ontologies (UIO and RIO) and pass them to RSC.

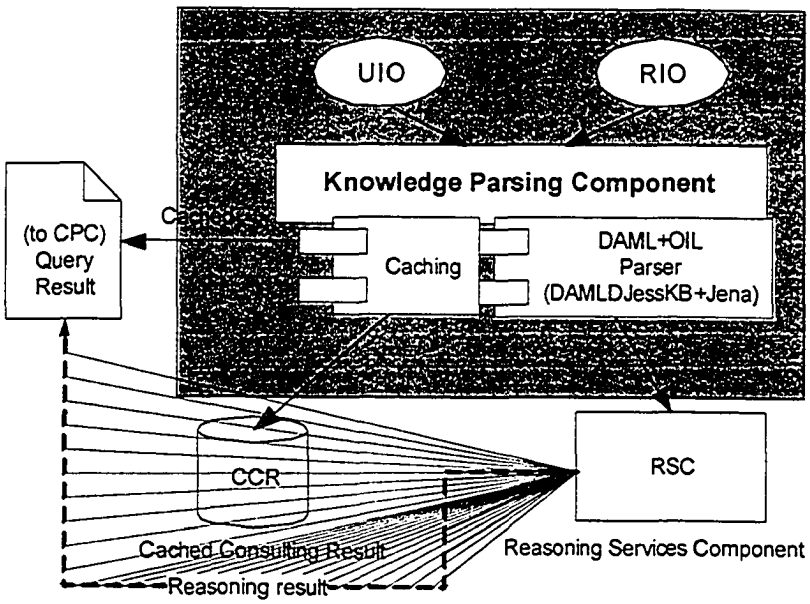

Figure 5.3: Structure of KPC

5.2.3.3 RSC: Resource Service Component

The structure of RSC is shown in Figure 5.6. This is the core part the whole system. RSC has a fuzzy inference engine which is using FuzzyJ toolkits. Based-on user acceptance ontology, the inference engine can produce the fuzzy curves. RSC will output an overall rate value against the user's query and send it back to the user through CPC.

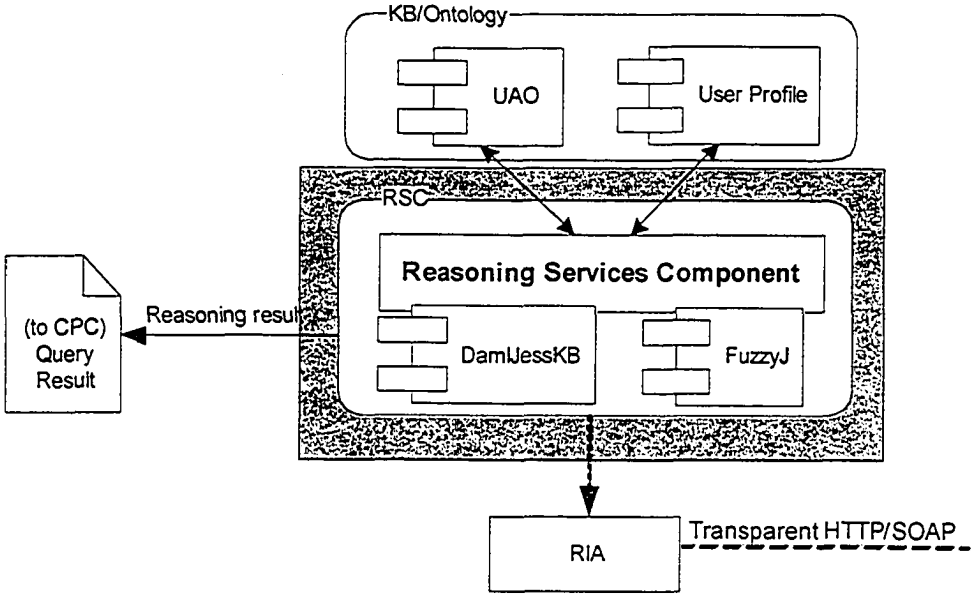

Figure 5.6: Structure of RSC

5.2.3.4 RIC: Resource Information Component

The structure of RIC is shown in Figure 5.7. RIC mainly collects resource information and exchanges the information with other agent systems. Resource information is saved in the format of Resource Information Ontology (RIO's definition in Chapter 3-1) and relational database table. RIC communicate with RSC using either SOAP or HTTP protocol.

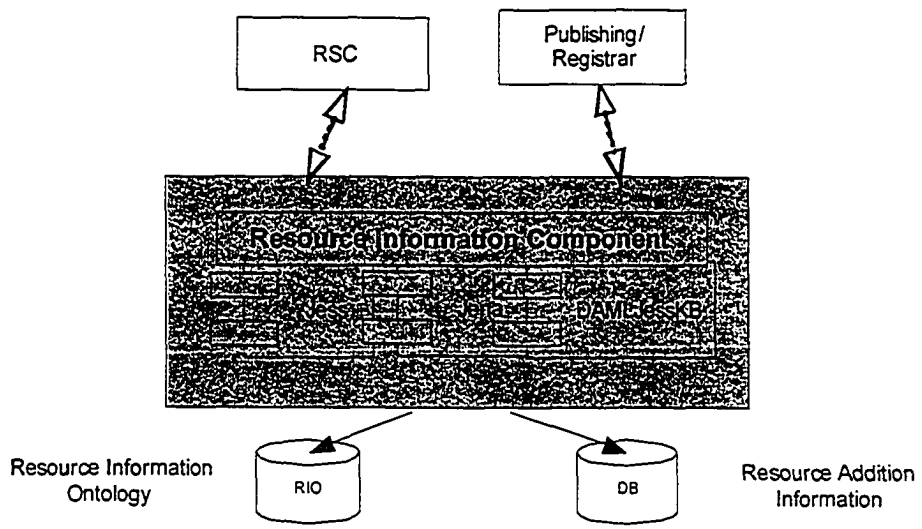

Figure 5.4: Structure of RIC

Section 5.3 Ontology with Fuzziness

On the surface, it seems that ontology and fuzziness are contrary to each other. Ontology is applied to express explicit and precise relations among entities, while fuzziness is applied to express imprecise, vague information. However, a combination of the two means the expression of things such as belief, preference, and whichever fuzziness is used. Ontologies are built in schema level and instances are created base on their models. The process of creating ontologies is definition of classes, properties, data type and the relationships. Ontology is capable of modeling fuzzy set through representing the fuzzy term and membership functions with rules, so in theory it is possible to apply such fuzzy ontologies into the Semantic Web [28].

Linguistic factors

LingFactA and LingFactB are the linguistic variables that will contribute to the conclusion. LingFactA is used for "small-medium" and LingFactB is used for "medium-large". LingFact can be presented with a fuzzy membership function which has values from [0, 1] as a fuzzyset. Figure 5.8 indicate the fuzzy MIN method for antecedent LingFactA and LingFactB. A fuzzy set is fulfilled in a certain membership function. Certain knowledge is needed to build such membership function curve.

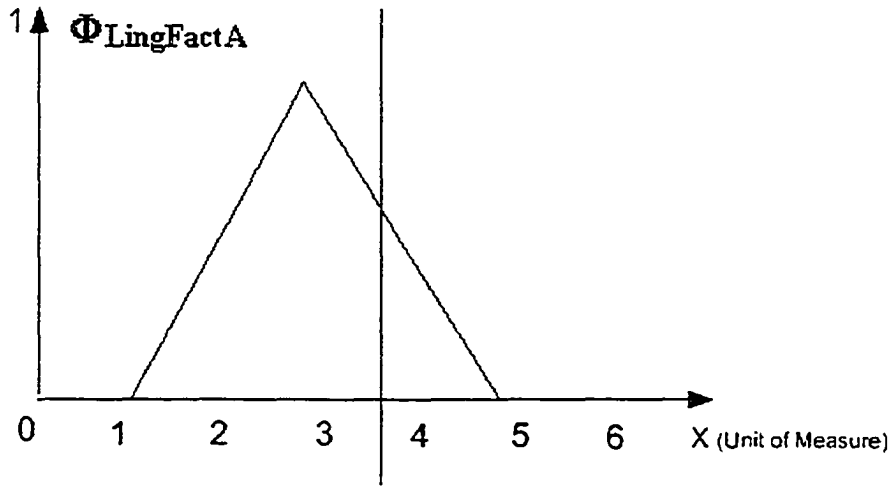

Figure 5.8: LingFactA Small-medium Membership Functions

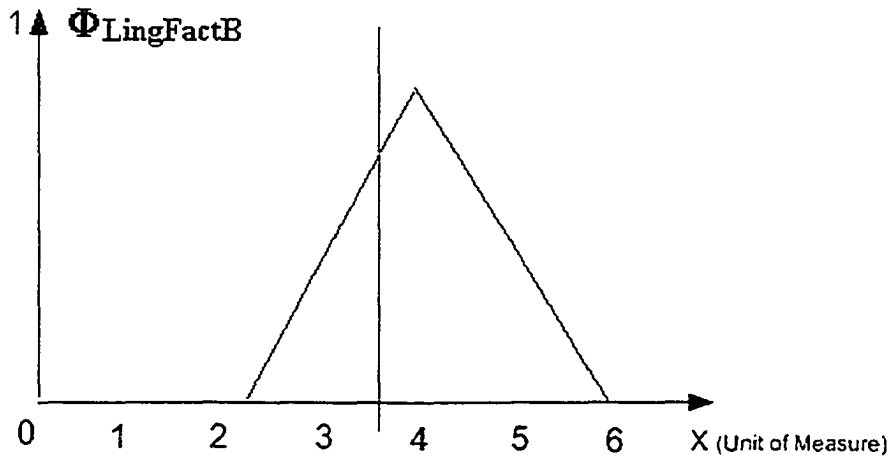

Figure 5.9: LingFactB Medium-large Membership Functions

The table below illustrates four individual Inputs from membership function A and B, assume that input A and input B have the same value (they are not necessarily the same).

Membership	Input 1 (4.2)	Input 2 (2.2)	Input 3 (3.4)	Input 4 (4.6)
$\phi_{LingFactA}$	0.3	0.7	0.7	0.2
$\phi_{LingFactB}$	0.9	0.1	0.9	0.5

A simple rule: **if LingFactA is "small-medium" and LingFactB is "medium-large" then conclusion/output is large**, using MAX yielding rule, we have $\phi_{Conl} = \max(\phi_{output1}, \phi_{output2,...outputn})$. In this case, there is only one rule, so the sum contribution is equal to the $\phi_{output1}$.

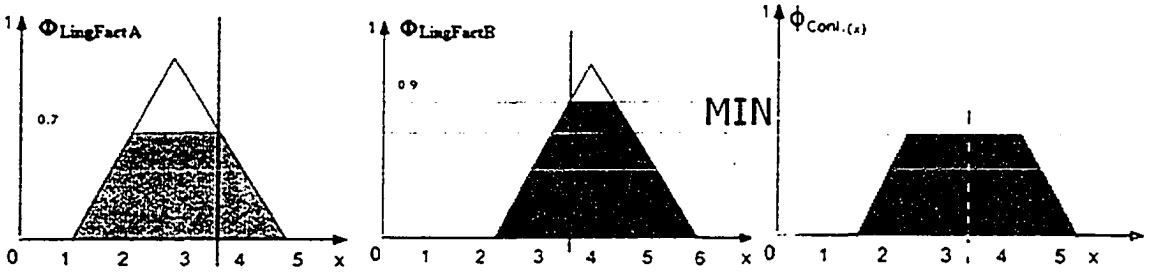

Figure 5.10: Fuzzy MIN Inference Method

Through MIN fuzzification, each Input was contributed to the output between [0,1]

Membership	Input 1	Input 2	Input 3	Input 4
ϕ_{Conl}	0.3	0.1	0.7	0.2

Once the output contribution has been produced, COG (Centre of Gravity) method is applied in order to get the crisp value from the output MF. This procedure is called "defuzzification" (See Figure 4.3 page42 for COG detail explanation)

With above fuzzification, we can draw a conclusion that Input 3 is the most satisfied candidate. We also can tell that Input 1 has pretty high score comparing with Input 2 and Input 4. So Input 1 can be seen as the alternative.

The process of using fuzzy ontology can be summarized as the following steps:

Step 1: define the linguistic term in certain domain

Determine the factors in order to draw a conclusion

Step 2: determine the corresponding rule

Domain knowledge is applied in this process; rules are adjustable and can be created on the fly although in most case, they are fixed.

Step 3: create the membership functions for each linguistic term

Domain knowledge is applied in this process; the membership function can be dynamically generated according to individual input

Step 4: represent the membership curve with ontology

This includes Schema and Instance ontology; schema is ontology that represents knowledge in a certain domain. Instance is the concrete ontology value in xml format. In summery, there are two steps to build such ontology, first build the ontology itself using Protégé 2000, then create DMAL instance, after that, we build graphic model to reflect the membership functions (PI, S, Z curves). During this process, corresponding fuzzy term are defined and linguistic variables are created. Finally we generate matrix of rules and feed those rules into RSC (Reasoning Service Component). With ontology fuzzification, we can use FuzzyJess or fuzzyJ tool kits to model the LHS and RHS, so that backward chain FuzzyRete algorithm can be used into semantic reasoning system. DAMLJessKB's default class is Rete, however we use FuzzyRete in FuzzyJess and FuzzyJ. FuzzyRete can be complied by running the following command. Figure 5.11 shows the command.

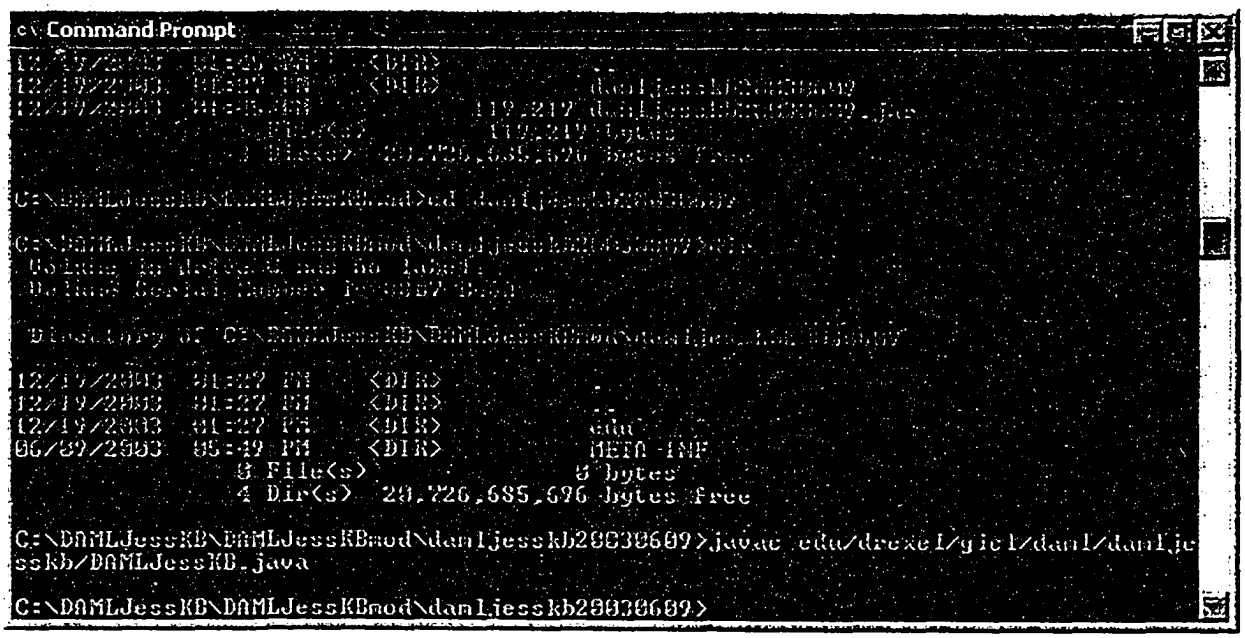

Figure 5.11: Using FuzzyRete in FuzzyJ

Step 5: integrate into web services

A web service is created to interpret the ontology as reasonor. The reasoner also deals with defuzzification process. Defuzzification is used to transform the fuzzy result sets to a crisp value. The most used defuzzification approaches are the Centre Of Gravity(COG) and Mean Of Maxima (MOM). COG has the advantage of producing smoothly varying controller output, and MOM has the advantage of greater speed due to fewer floating point calculations. We use COG in the thesis [29].

In my thesis, approximate reasoning methodology is applied in the Semantic Web environment. A concept of fuzzy ontology is proposed to represent individual's acceptance of semantic web services.

FuzzyJ Curves

Here we explain a little bit about the type of fuzzy curves which represent different FuzzySet, in my thesis three types of curves are used: S-curve | Z-curve | Pi-curve. Three curves (Membership Functions, MF) are possible to use in FuzzyJ tool.

S-curve

SFuzzySets has the similar shape like letter S, it has 0 at the left edge and 1 at the right edge [30]. See diagram below (figure 5.12)

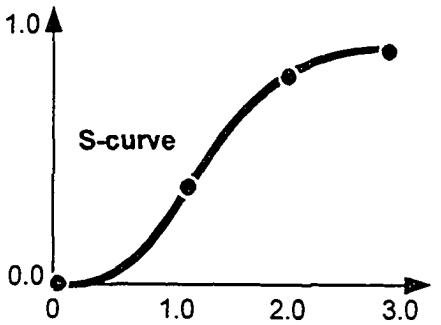

Figure 5.12: SFuzzySet

In order to create such curve, the X-value for the left and right edges of the curve must be provided.

Z-curve

ZFuzzySets has the similar shape like letter Z. it has 1 at the left edge and 0 at the right edge. See diagram below (figure 5.13)

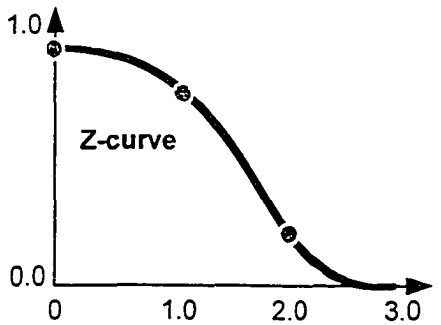

Figure 5.53: ZFuzzySets

In order to create such curve, the X-value for the left and right edges of the curve must be provided.

Pi-curve

PiFuzzySets has the similar shape like symbol π. it has 0 at the left and right edges of the curve and a 1 at the middle. See diagram below (figure 5.14)

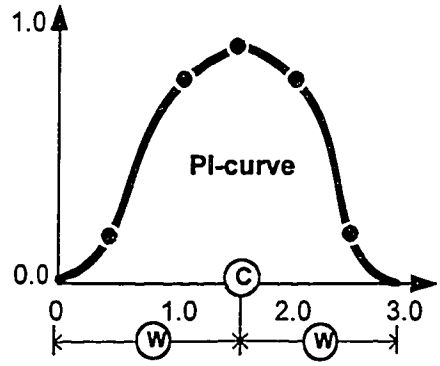

Figure 5.64: PiFuzzySets

In order to create such curve, the X-value for the center point (Ⓒ) and Deviation to the center (Ⓦ) of the curve must be provided.

The user service is capable of parsing fuzzy variable membership functions extracted from user acceptance ontology which is encoded in DAML format; there are many DAML reasoner around. We use FuzzyJ toolkit because it is powerful and easy to use.

Chapter 6
Hotel Reservation System

An example of proposed concept of semantic web service agent is illustrated using an online intelligent hotel reservation system. Registered users access the system via the web browser. They send requests to the service providers via Communication Presenting Component (CPC). Those requests are forwarded to Reasoning Services Component (RSC). Based on users' preference (which also called User Acceptance) represented by User Acceptance Ontology, RSC verifies hotel information services instance (represented by Hotel Information Ontology, collected by Hotel Information Agent) and parsed by Knowledge Exploration Component (KPC) against user's personalization to give corresponding response to the user. The system utilizes semantic web services. We assume several hotel services providers are available in the context and they publish or advertise their services according to the defined ontology though UDDI. The goal of this work is to build a light-weighed fuzzy expert system for hotel reservation service on the website.

Section 6.1 Ontologies

There are three types of ontologies involved:

1. User Acceptance Ontology (UAO): representing user's preferences related to acceptance of services. The reasoner will be built to parse and infer the UAO fuzzyset membership functions.
2. Hotel Information Ontology (HIO): representing general information about hotel. In contrast to UAO,
3. User Information Ontology (UIO): representing general information about user. UIO is another contributor to create UAO fuzzyset membership curves, it gives the concrete value of the preference of specified user, such as PriceAcceptance dollar amount, LocationAcceptance distance to the business center, availability of facilities and so on.

6.1.1 User Acceptance Ontology (UAO)

Regarding acceptance of services, a dedicated ontology which would "mimic" user's criteria regarding the like-minded service is proposed. User preferences of service acceptance (shown in Figure 6.1) are modeled by defining the following factors: Price Acceptance, Location Acceptance, Facility Acceptance. These factors altogether contribute to the final acceptance.

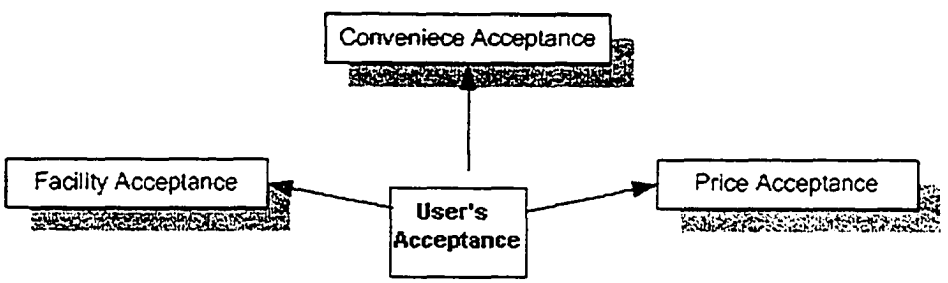

Figure 6.1: User Acceptance Ontology

We have a few assumptions on UAO:

Assumption 1: We define the User Acceptance Ontology (UAO) in the following factors to reflect user's "preferences"

a) Price Preference
b) Location Preference

c) Facility Preference

Assumption 2: We give the same weight (based on user's need) to each item

Assumption 3: User will fulfills the required information via the website

Let's take a look at the real scenario of a hotel reservation system. In the case of finding suitable hotel, we are presenting the degree of acceptance on an available room according to customer's specified preference. Human acceptance is more like belief than a crisp statement. For example, in the case of user's acceptance on the location of a hotel, is 1.5KM from hotel to business center close or far? The answer depends on individual user. If user likes walking along the street, he thinks that it is convenient concerning the location; others may think it is too far because they don't have time to do it. Even for the same person, what is close or how close it is quite vague. 1.5KM is close, what about 1.6 KM? In order to model this belief with more reflection of its nature, a fuzzy approach is used in this scenario. User acceptance ontology is used to express membership functions and relations. Figure 6.2 shows an example of the membership functions for three levels of acceptance: *HighAcceptance*, *MediumAcceptance* and *LowAcceptance*. Here HighAcceptance, MediumAcceptance and LowAcceptance are three linguistic predicates. Each of them has a membership function to describe the satisfied degree. $\phi_{HighAcceptance}$, $\phi_{MediumAcceptance}$, $\phi_{lowAcceptance}$.

The DAML-code represents User's *MediumAcceptance* on the hotel location.

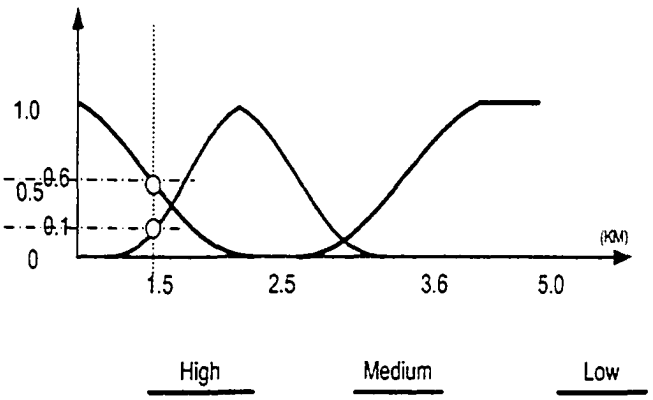

Figure 6.2: Location Acceptance Membership functions

```
<UserAcceptance:MediumLocationAcceptance
rdf:ID="myMediumLocationAcceptance">
  < UserAcceptance:hasMediumAt>
    <XMLSchema:double>
      <rdf:value>2</rdf:value>
    </XMLSchema:double>
  </ UserAcceptance:hasMediumAt>
  < UserAcceptance:hasDeviation>
    <XMLSchema:double>
      <rdf:value>1</rdf:value>
    </XMLSchema:double>
  </ UserAcceptance:hasDeviation>
</ UserAcceptance:MediumLocationAcceptance>
```

The following diagram (figure 6.3) displays the similar Facility Acceptance Membership functions.

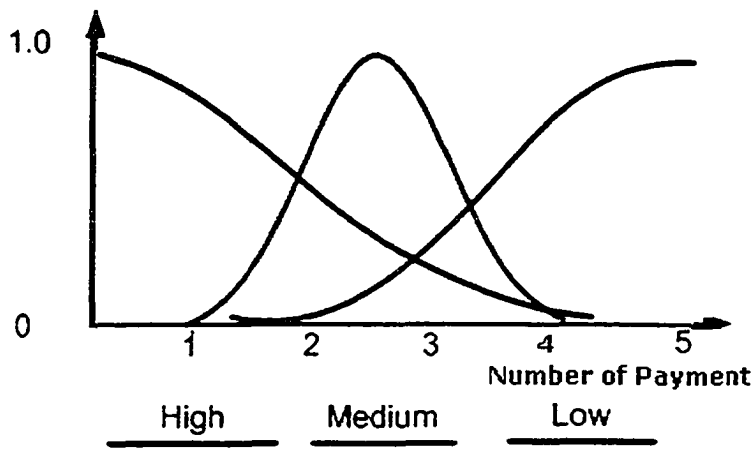

Figure 6.3: Facility Acceptance Membership functions

The following description shows the example of fuzzy rule for user location acceptance:

location-low-convenience-low →*if the location is low then the convenience acceptance is low*

location-med-convenience-med→if the location is medium then the convenience acceptance is medium

location-high-convenience-high →*if the location is high then the convenience acceptance is high*

Those linguistic term can be fuzzified into jess knowledge base by this command

*(assert (convenienceAcceptance (new FuzzyValue ?*locationAccept* "low")))*

There is preference affecting the priority of acceptance. Each factor was assigned with a weight (level of importance), such as [LocationAcceptance], the value of location acceptance contributes to the major decision of acceptance. The other preferred factors are in low priority, the following rules:

If LocationAcceptance is high and FacilityAcceptance is High Then Acceptance is High

If LocationAcceptance is Medium and FacilityAcceptance is Low Then Acceptance is Medium

If LocationAcceptance is low and FacilityAcceptance is High Then Acceptance is Low

We can tell from rules that "LocationAcceptance" is the primary factor. My thesis is targeting at showing the concept of semantic fuzziness application. We are not touching the real expert knowledge in any domain. All knowledge rules can be built by domain knowledge engineers so that we are not involving too much on defining the specific rule. In concern of this, we assume the fuzzy rule is simple and presumably correct.

Now let's have a look at the detail of UAO.

UAO represents user's preferences related to acceptance of factors. It directly related to the reasoner built to interpret those fuzzyset membership functions. AcceptanceInPrice, AcceptanceInLocation and AcceptanceInFacilityConvenience will be illustrated in explaining the fuzzyset curves (Z, S, Pi fuzzysets). [31]

1. AcceptanceInPrice

AcceptanceInPrice is determined by the price of hotel expected by the user (price unit is in Canadian dollar).$ 0.00-$120.00

HighAcceptanceInPrice: left diagram was expressed in RDF, right diagram was Z-curve, see figure 6.4

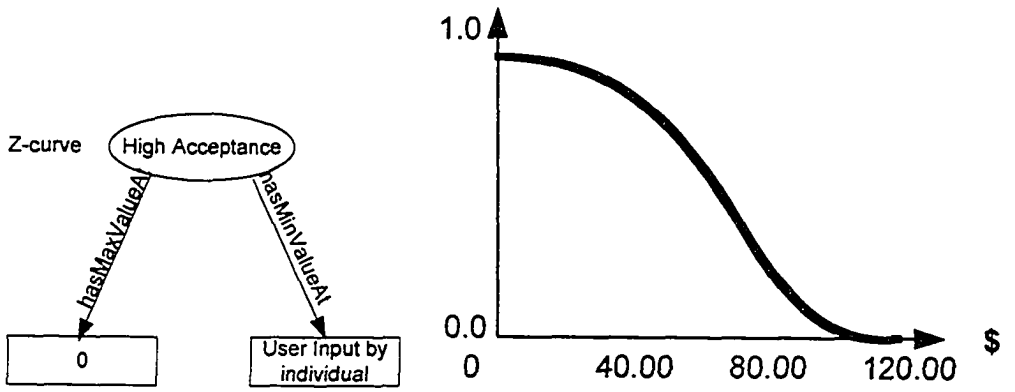

Figure 6.4: HighAcceptanceInPrice [32]

LowAcceptanceInPrice: left diagram is expressed in RDF, right diagram is S-curve, see figure 6.5.

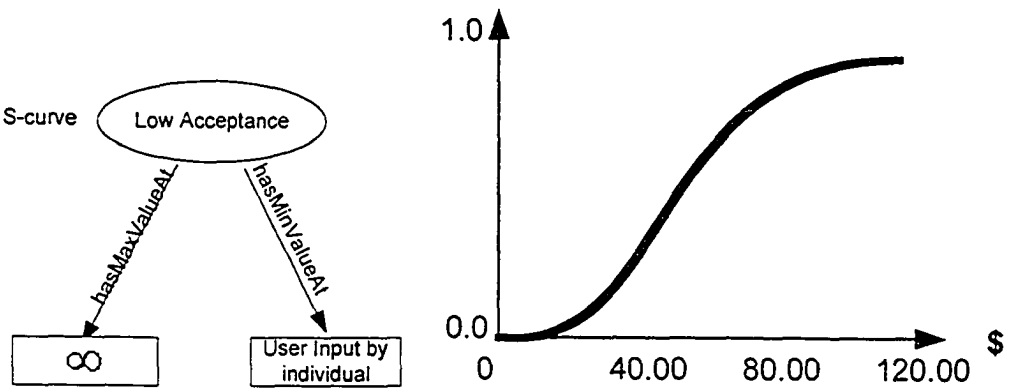

Figure 6.5: LowAcceptanceInPrice [33]

MediumAcceptanceInPrice: left diagram is expressed in RDF, right diagram is Pi-curve, see figure 6.6.

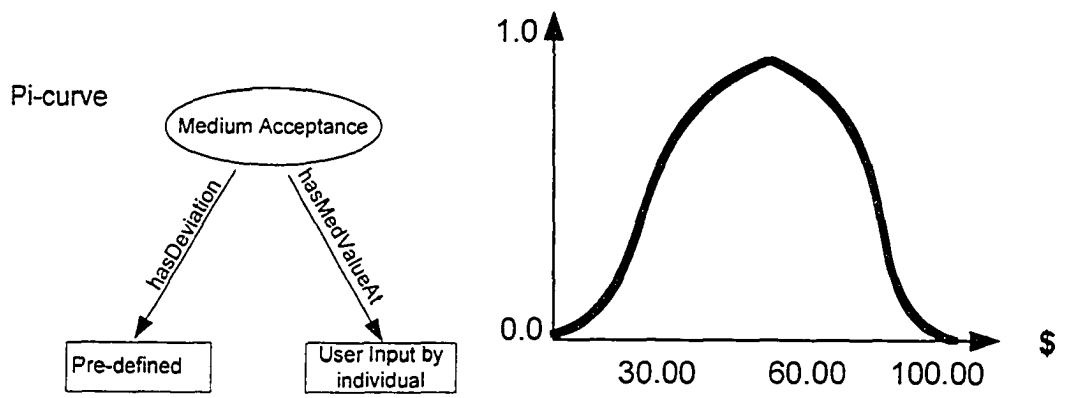

Figure 6.6: MediumAcceptanceInPrice [34]

2. AcceptanceInLocation

AcceptanceInLocation is determined by the distance from hotel to the business center expected by the user (distance unit is in kilometer). 0.0-3.0

HighAcceptanceInLocation: left diagram was expressed in RDF, right diagram was Z-curve, see figure 6.7

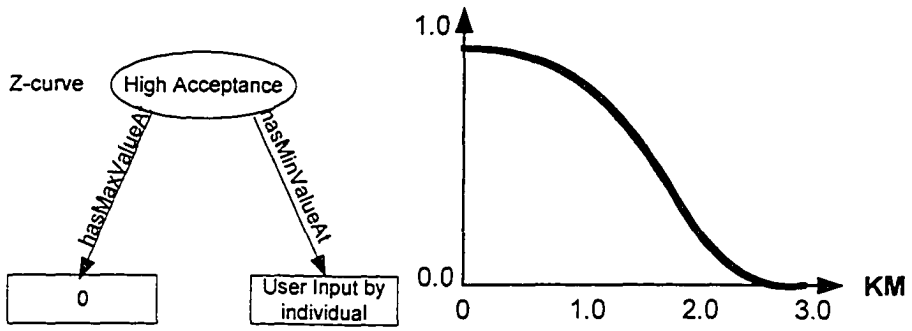

Figure 6.7: HighAcceptanceInLocation

LowAcceptanceInLocation: left diagram is expressed in RDF, right diagram is S-curve, see figure 6.8.

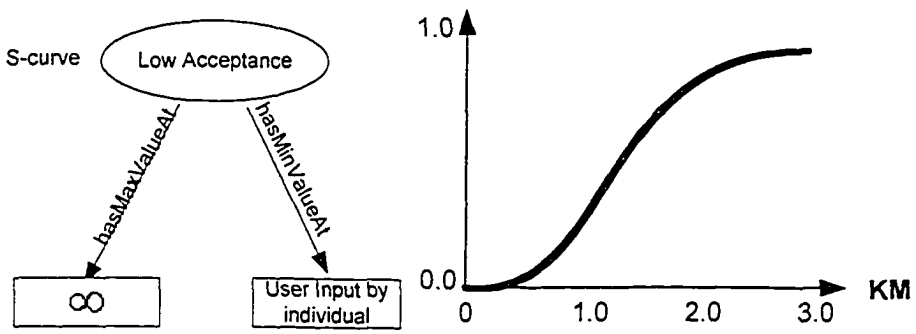

Figure 6.8: LowAcceptanceInLocation

MediumAcceptanceInLocation: left diagram is expressed in RDF, right diagram is Pi-curve, see figure 6.9.

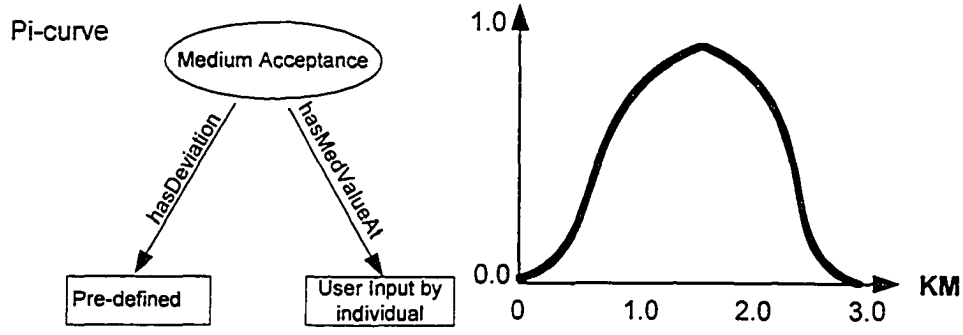

Figure 6.9: MediumAcceptanceInLocation

3. AcceptanceInFacilityConvenience

AcceptanceInFacilityConvenience is determined by the number of available facilities from the hotel. (e.g hotel has maximum six type of facilities). 1-6

Must Have: weight is 1

Nice to have: weight is 0.5

Don't care: weight is 0

No.	Facility Name	Value
1	Bus/Van Parking	Must Have/ Nice to have /Don't care
2	Auto Parking	Must Have/ Nice to have /Don't care
3	Car Rental	Must Have/ Nice to have /Don't care
4	Local Area Transportation	Must Have/ Nice to have /Don't care
5	Safety Deposit Box	Must Have/ Nice to have /Don't care
6	Laundry/Valet	Must Have/ Nice to have /Don't care

LowAcceptanceInFacilityConvenience: left diagram is expressed in RDF, right diagram is S-curve, see figure 6.10.

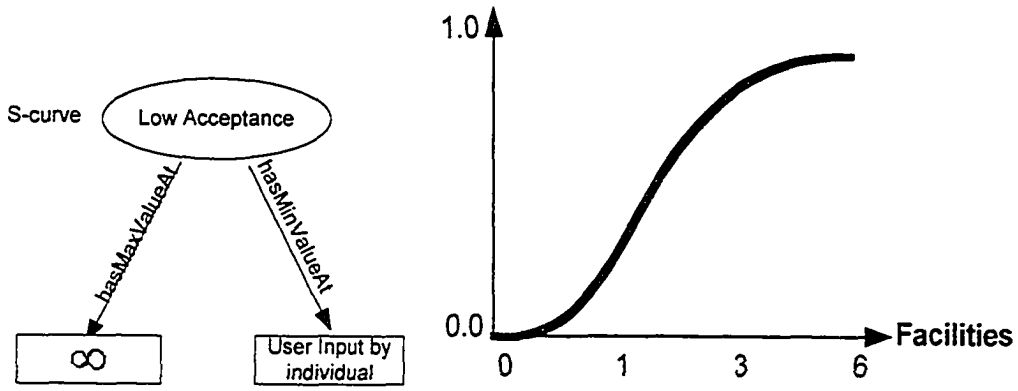

Figure 6.10: LowAcceptanceInFacilityConvenience

HighAcceptanceInFacilityConvenience: left diagram is expressed in RDF, right diagram is Z-curve, see figure 6.11.

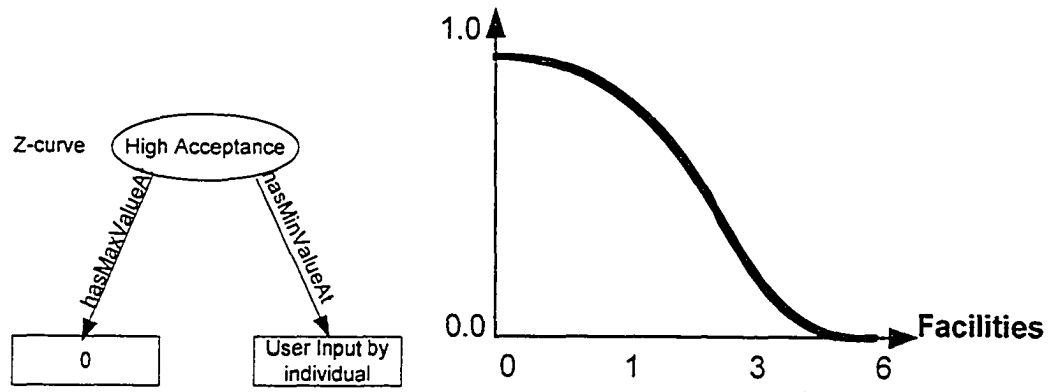

Figure 6.11: HighAcceptanceInFacilityConvenience

MediumAcceptanceInFacilityConvenience: left diagram is expressed in RDF, right diagram is Pi-curve, see figure 6.12.

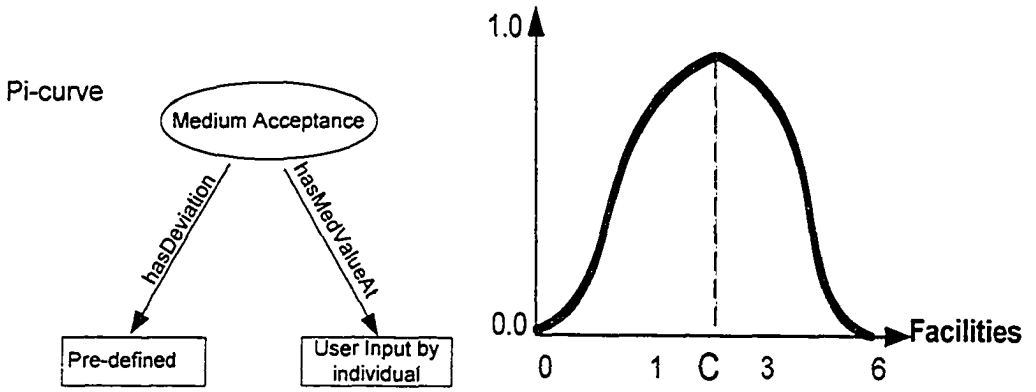

Figure 6.12: MediumAcceptanceInFacilityConvenience

The rest of factors which are contributing to UAO are very similar, they all have Z-fuzzyset, S-fuzzyset and Pi-Fuzzyset individually.

Sample of UAO:

```
<rdf:RDF
  xmlns:XMLSchema ="http://www.w3.org/2000/10/XMLSchema#"
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:daml_oil ="http://www.daml.org/2001/03/daml+oil#"
  xmlns:HumanBehaviourO ="http://HumanBehaviourOntology#"
  xmlns:rdfs ="http://www.w3.org/2000/01/rdf-schema#"
  xmlns ="http://HumanBehaviourOntology#"
  >
<daml_oil:Class rdf:ID="HighFacilityConvenienceAcceptance">
  <rdfs:subClassOf rdf:resource="#FacilityConvenienceAcceptance"/>
</daml_oil:Class>
 <daml_oil:DatatypeProperty rdf:ID="hasMediumAt">
  <daml_oil:domain rdf:resource="#HighPriceAcceptance"/>
  <daml_oil:domain rdf:resource="#MediumFacilityAcceptance"/>
  <daml_oil:domain rdf:resource="#MediumLocationAcceptance"/>
  <daml_oil:domain
rdf:resource="#MediumFacilityConvenienceAcceptance"/>
  <daml_oil:domain rdf:resource="#MediumPriceAcceptance"/>
  <daml_oil:domain rdf:resource="#MediumRisk"/>
  <daml_oil:domain rdf:resource="#MediumServiceAcceptance"/>
  <daml_oil:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#double"/>
</daml_oil:DatatypeProperty>
 <daml_oil:Class rdf:ID="LowLocationAcceptance">
  <rdfs:subClassOf rdf:resource="#LocationAcceptance"/>
</daml_oil:Class>
<daml_oil:Class rdf:ID="MediumLocationAcceptance">
  <rdfs:subClassOf rdf:resource="#LocationAcceptance"/>
</daml_oil:Class>
 <daml_oil:Class rdf:ID="HighLocationAcceptance">
```

```
  <rdfs:subClassOf rdf:resource="#LocationAcceptance"/>
</daml_oil:Class>
```

6.1.2 Hotel Information Ontology (HIO)

Contact Information

1. Name: ABC Grant Hotel
2. Address:
3. Tel:
4. Fax:
5. Email:

Pricing: $90 USD /per night

Location Convenience:

No.	Payment method Name	Value (KM)
1	To Business Center	2.5
2	To Conference Center	1.0
3	To Shopping Center	3.0
4	To Public Transportation	0.5
5	To the Airport	30.0

Facility Convenience:

No.	Service Name	Value
1	Airport Pickup/Dropoff	YES
2	Bus/Van Parking	YES
3	Auto Parking	NO
4	Car Rental	YES
5	Foreign Currency Exchange	YES
6	Local Area Transportation	NO
7	Safety Deposit Box	NO
8	Laundry/Valet	YES

9	Printer Use/Rental	YES
10	Fax: Receive/send Fax	YES
11	Photocopying Service	YES
12	Cable TV	YES
13	High Speed Internet	NO
14	Refrigerator	NO
15	Air Conditioning	YES
16	Iron & Ironing Board	
17	Swimming Pool	NO
18	Dryer	YES

Ontologies are built using Protégé 2000. Each ontology includes schema (class) and instance (object). Another tool used to generate such ontology is OilEd, which has the FaCT reasoner built-in.

Sample of HIO

```
<rdf:RDF
  xmlns:XMLSchema ="http://www.w3.org/2000/10/XMLSchema#"
  xmlns:HIO_daml ="file://C/Ontology/HIO/HIO.daml#"
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:daml_oil ="http://www.daml.org/2001/03/daml+oil#"
  xmlns:rdfs ="http://www.w3.org/2000/01/rdf-schema#"
  xmlns ="file://C/Ontology/HIO/HIO.daml#"
  >
<daml_oil:ObjectProperty rdf:ID="roomPrice">
  <daml_oil:domain rdf:resource="#Room"/>
  <daml_oil:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#double"/>
</daml_oil:ObjectProperty>
 <daml_oil:ObjectProperty rdf:ID="distanceToDest">
  <daml_oil:domain rdf:resource="#Hotel_Convenience"/>
  <daml_oil:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#double"/>
</daml_oil:ObjectProperty>
</rdf:RDF>
```

6.1.3 User Information Ontology (UIO)

UIO includes User basic contact information and user preference information

User basic contact information

1. First Name: John
2. Last Name: Smith

3. Email Address:
4. Phone Number:

User preference information

This information is to reflect user's preference defined in UAO ontology. As I mentioned in Section 1.1, UIO is a contributor to create UAO fuzzyset membership curves due to its information on preference of specified user. Such information includes PriceAcceptance, LocationAcceptance and FacilityAcceptance.

Sample of UIO

```
<rdf:RDF
  xmlns:XMLSchema ="http://www.w3.org/2000/10/XMLSchema#"
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:daml_oil ="http://www.daml.org/2001/03/daml+oil#"
  xmlns:UIO_daml ="file://C/Ontology/UIO/UIO.daml#"
  xmlns:rdfs ="http://www.w3.org/2000/01/rdf-schema#"
  xmlns ="file://C/Ontology/UIO/UIO.daml#">
<daml_oil:Class rdf:ID="AirConditioningPreference">
  <rdfs:subClassOf
rdf:resource="file://C/Ontology/UIO/UIO.daml#RoomFacilityPreference"/>
<daml_oil:ObjectProperty rdf:ID="emailAddr">
  <daml_oil:domain rdf:resource="#Person"/>
  <daml_oil:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</daml_oil:ObjectProperty>
  <rdfs:subClassOf
rdf:resource="file://C/Ontology/UIO/UIO.daml#HotelFacilityPreference"</
daml_oil:Class>
<daml_oil:Class rdf:ID="PricePreference">
</daml_oil:Class>
<daml_oil:Class rdf:ID="CarRentalPreference">
  <rdfs:subClassOf
rdf:resource="file://C/Ontology/UIO/UIO.daml#HotelServicePreference"/<U
IO_daml:AlarmCallPreference rdf:ID="UIOnew_00087">
</UIO_daml:AlarmCallPreference>
<daml_oil:Class rdf:ID="RefrigeratorPreference">
  <rdfs:subClassOf
rdf:resource="file://C/Ontology/UIO/UIO.daml#RoomFacilityPreference"/</
daml_oil:Class>
</rdf:RDF>
```

Section 6.2 System Implementation

Semantic Web Services is a set of ontologies that describe the properties and capabilities of various Web Services for automaton in service discovery, composition, invocation and

monitoring. In my thesis, a fuzzy resoner is built upon Web Services using DAML+Oil ontology instances as the knowledge base.

6.2.1 Implementation of Architecture

Based-on the architecture explained in Chapter 5, we simply instantiate the resource as hotel, there are naming changed for mapping the hotel reservation application, accordingly resource service/agent is name hotel information service/agent. The architecture of the User Service is represented in Figure 8. It consists of a set of components which ensure intelligent and automatic execution of user requests. The main elements are:

- **Communication service**: transform and routing information among user request and response, talk to hotel service and user service.
- **Ontology Parser/reasoner**: read ontology instances and transform to triples, use HP Jena as the parser, mostly the reasoner is dealing with User Acceptance Ontology
- **Knowledge Base** includes ontologies instances (Users, Hotel)

Figure 6.13 shows the overall services layer structure. Its components interact with UI, semantic web service registry, storage and other agents. The components of the services have the Knowledgebase Parser and Reasoning Service. Knowledgebase Parser is built with HP's Jena and the resoner is implemented with Jess+FuzzyJ. Jess is created and supported by Sandia Labs. It is an efficient Java rule engine for the rule-based system. As an extension, FuzzyJ introduces fuzzy concept to Jess. FuzzyJ has been developed by National Research Council of Canada's Institute for Information Technology.

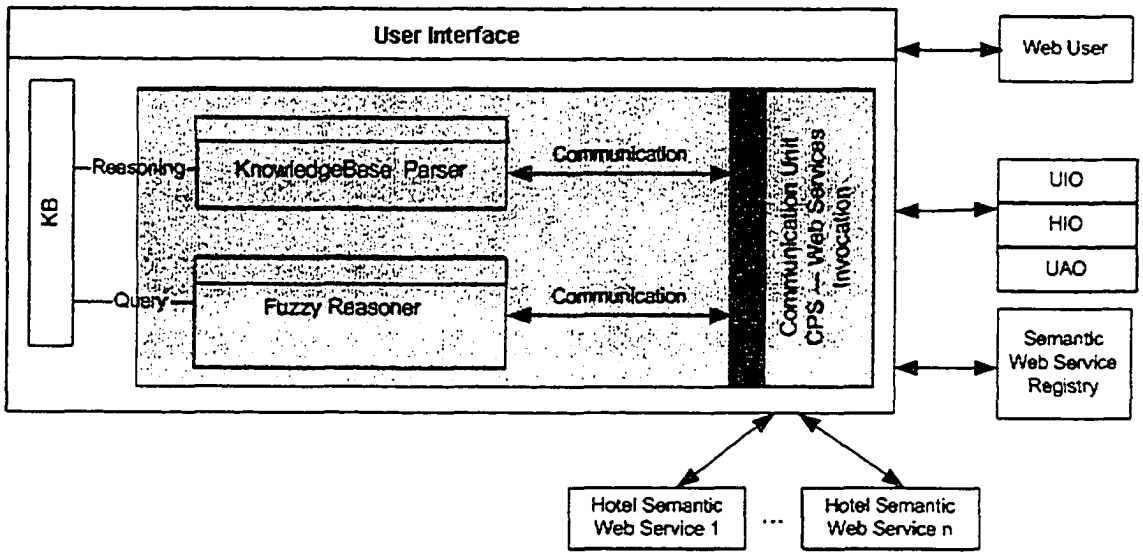

Figure 6.2: Layer implementation

Figure 6.14 shows the data flow of the system. The input is user's query and, the output of the Reasoning Service represents the rates of hotel instances. (Please reference Section 6.2 for detail execution model)

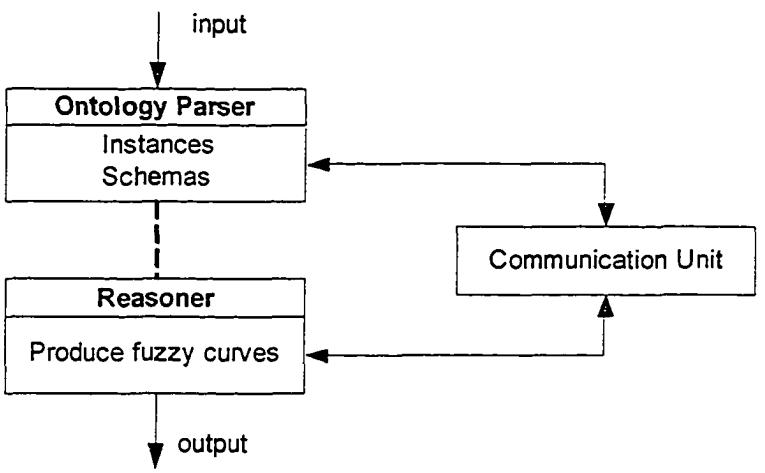

Figure 6.3: Overall Services Layer – data flow

There are two main services in this architecture: User Service and Resource Service (see figure 6.15). The process of locating the best-satisfied result is the preference-resource matching. User service accepts request from a user, get his or her personal information and requirement before passing them to resource services in the form of ontology

instance. Personal information includes user's basic contact such as name, email and so on; the user's requirements are what to inquire according to the preference, for example, in the case of looking for hotel, user may concern price, location, facility, service as acceptance factors. Each preference can be represented by a few languished variables. For example, user price acceptance can be literally classified into three fuzzy variables: HighAcceptanceInPrice | MediumAcceptanceInPrice | LowAcceptanceInPrice. Each fuzzy variable can be fulfilled with a membership function.

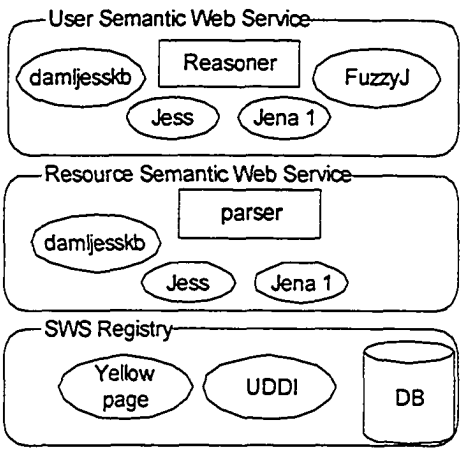

Figure 6.15: User Service and Resource Service layers

6.2.2 Dynamic Behavior of the System

The execution model of the service architecture with the proposal approach is shown in Figure 6.16. As you can see from the picture, it shows the 5-step execution of the system: Registration→Request→Communication and Parsing→Reasoner→Response. The user has to register his or her information in the system. The user submits a request containing his or her preferred hotel options. The parser parses the request and forward the relevant ontology information to the reasoner, and finally the reasoner presents the desired response for the user.

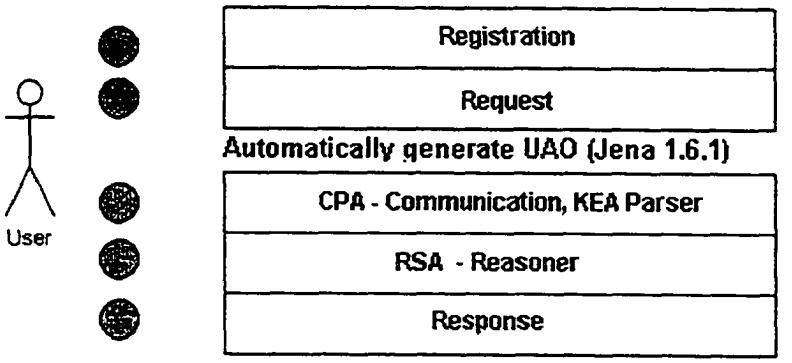

Figure 6.16: Model execution flow

Agents communicate with each other on behalf of their owners. So they execute the specified task interactively. All services are registered in the registry. In logic, there are User Service Agent and Hotel Service Agent (since the agent platform is not addressed, User Service Agent was implemented as a User Service; and Hotel Service Agent is called Hotel Service) in the system. Inside Hotel Service, there is a DAML-S processor to explore hotel web services and to discover hotel information that was represented using HIO. Inside the User Service, a fuzzy DAML-S reasoner is built, which is capable of inferring user's preference. The reasoner verifies HIO against UAO and returns the result or suggestion to the user.

Figure 6.17 illustrates how ontologies are used in the system. Here is the description of the whole process. There are three types of ontology, UIO, HIO, UAO. It starts from submitting a request by the front user. The request is made up by a string (string contains link information, such as user IP address etc.) and UIO. Based on these parameters, the hotel service agent locates HIOs (again, we assume that there are many HIO resources existing in the registration), then hotel services agent will refine the result by verifying against UAO to ensure that the result is specific suitable for the user. It ends up with the final HIO and a string (string contains link information, such as Hotel IP address etc.) to the front user.

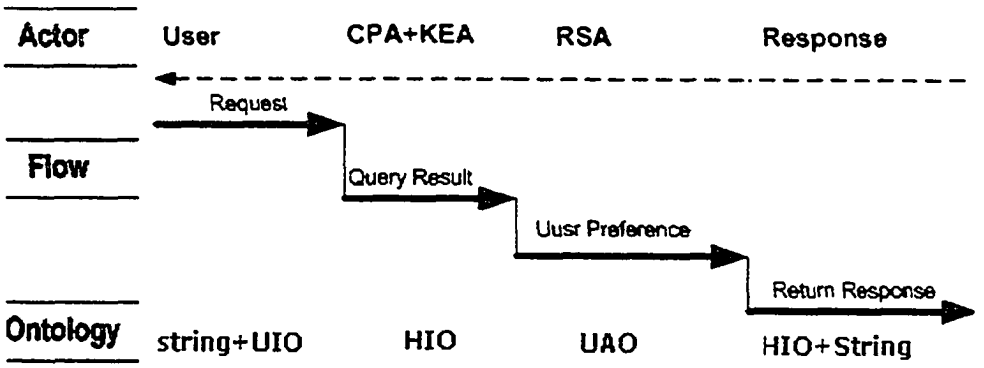

Figure 6.17: ontology usage

The processor and reasoner also can be incorporated as higher level of composite DAML-S process. , the assumption is made that there are pre-defined hotel web services and Hotel Service knows where to locate such services. A sample implementation framework will be given in the next section.

CPS gets user request and ask KPC to inquiry *HIO instances* according to registered hotels resources, the response (those *HIO instances*) from CPS+KPC is sent to RSC to inference based on UAO.

6.2.3 Implementation of the layers

6.2.3.1 User Interface implementation

Figure 6.18 shows the user interface. User can access a website from the Internet, filling his or her preference such as Price, Location, or Facilities, with contact information such as user name, room type, travel type and so on. Since we use FuzzyJ to build the reasoner, Min/Max/A/B values are needed to create the curves.

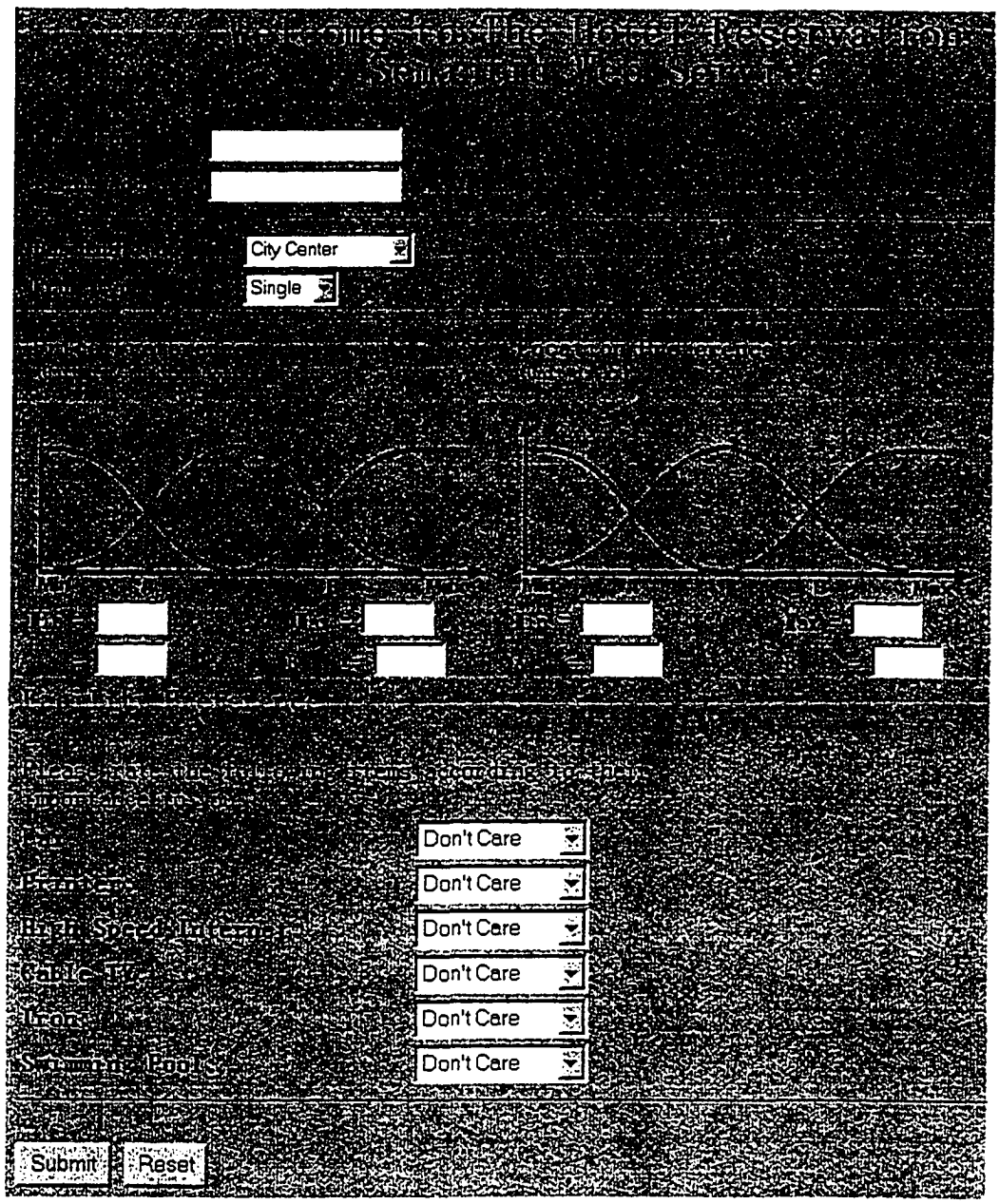

Figure 6.4: User interface

6.2.3.2 Service layers implementation

We assume hotel service advertisements available on service provider; it means the registration service can be hosted and discovered through UDDI. At matter of fact, more implantation of the system will include a service for Hotel Service Providers. Hotel Service Providers (HSP) is mostly dealing with hotel resource registration. Web service description (WSDL) and invoking (method SOAP envelope or HTML to the Web Services provider as a request, send a SOAP message to HSP, then wait for response.

- Describe the services – description/profile
- Locate the services – discovery
- Consume the services – use it
- Evaluate the services – quality

Implement the reasoner: Part A - fuzzyrulesEngine

FuzzyReasoning: the following codes show an example of LocationAcceptance inference rules (in CLIPS syntax).

```
HIO interactive with HSP - Dynamically create corresponding HIO
instances (return XML then convert it to DAML using Jena ARP as the
parser)
Ontology
;---------------------------------------------------------
;-----------HOTEL SERVICE APROXIMATE REASONING-----------
;---------------------------------------------------------
;--Import FuzzyJ package--
(import nrc.fuzzy.*)
(load-package nrc.fuzzy.jess.FuzzyFunctions)

;---------------------------------------------------------
;-------Hashtable to store results --------------------
;---------------------------------------------------------

(defglobal ?*finalResult* = (new java.util.Hashtable))
(defglobal ?*hotelInformation* = (new java.util.Hashtable))
  ;---------------------------------------------------------
  ;------------User's Acceptance Fuzzy Variable-------------
  ;---------------------------------------------------------
(defglobal ?*userAccept* = (new FuzzyVariable "userAccept" 0.0 10.0
""))

;------------------- Building Fuzzy Curves for User Acceptance
(defrule init_userAccept
=>
      (?*userAccept* addTerm "low" (new ZFuzzySet 3.0 5.0))
      (?*userAccept* addTerm "medium" (new PIFuzzySet 5.0 2.0))
      (?*userAccept* addTerm "high" (new SFuzzySet 5.0 8.0))
)
  ;---------------------------------------------------------
  ;-------------- Location Acceptance ----------------------
  ;---------------------------------------------------------
;--Global variable used for LocationAcceptance
(defglobal ?*locationAccept* = (new FuzzyVariable "location" 0.0 20.0
"km"))
      ;-----------------------------------------------------------------
      ;------------Building curve for HighLocationAcceptance-----------
--
      ;-----------------------------------------------------------------
 (defrule Build_HighLocationAcceptance_Curve
```

```
      "create Fuzzy Curve for High Location Acceptance based on user
Input(i.e. UIO)"
      (PropertyValue http://www.w3.org/1999/02/22-rdf-syntax-ns#type
?highLocAccept
            file://C/Ontology/UAO/UAO.daml#HighLocationAcceptance)
      ;--get Max Value (the first value when the curve reaches 1)
      (PropertyValue file://C/Ontology/UAO/UAO.daml#hasMaxValueAt
            ?highLocAccept
            ?max)
      (PropertyValue http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            ?max
            ?maxValue)
      ;--get Min Value (the first value when the curve reaches 0)
      (PropertyValue file://C/Ontology/UAO/UAO.daml#hasMinValueAt
            ?highLocAccept
            ?min)
      (PropertyValue http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            ?min
            ?minValue)
=>

      (?*locationAccept* addTerm "high" (new nrc.fuzzy.ZFuzzySet
?maxValue ?minValue))
      (printout t "High Location Acceptance MaxValueAt " ?maxValue
crlf)
      (printout t "High Location Acceptance MinValueAt " ?minValue
crlf)
      (printout t (call (new nrc.fuzzy.FuzzyValue ?*locationAccept*
"high") plotFuzzyValue "*") crlf)

)
```

Implement the reasoner: Part B – Implement the ontology instances generator

DAMLGenerator: the following java codes take the input from the user and convert the information into ontology instances.

```
package hotelreservation;
import junit.framework.*;
public class TestDAMLGenerator extends TestCase {
  private DAMLGenerator dAMLGenerator = null;
  private User userInfor;
  public TestDAMLGenerator(String arg0) {
    super(arg0);
  }
  protected void setUp() throws Exception {
    super.setUp();
    /**@todo verify the constructors*/
    dAMLGenerator = new DAMLGenerator();
    userInfor = new User("UID001");
    userInfor.SetUserName("John Smith");
    userInfor.SetUserEmail("jsmith@fuzzylogic.ece.ualberta.ca");
    userInfor.SetUserPriceA(50);
    userInfor.SetUserPriceB(150);
    userInfor.SetUserPriceMax(300);
    userInfor.SetUserPriceMin(0);
```

```
  }
  protected void tearDown() throws Exception {
    dAMLGenerator = null;
    super.tearDown();
  }
  public void testGenerateUIOinst() {
    User userInforNull = null;
    String expectedReturnNull = null;
    String actualReturnNull =
dAMLGenerator.generateUIOinst(userInforNull);
    assertEquals("return value", expectedReturnNull, actualReturnNull);
    String expectedReturn = "file://C/Ontology/UIO-inst/UID001.daml";
    String actualReturn = dAMLGenerator.generateUIOinst(userInfor);
    assertNotNull(actualReturn);
    assertEquals(expectedReturn,actualReturn);
    String expected_UAO_URL = "file://C/Ontology/UAO-inst/UID001.daml";
    String actual_UAO_URL =
dAMLGenerator.generateUAOinst(actualReturn);
    assertNotNull(actual_UAO_URL);
    assertEquals(expected_UAO_URL,actual_UAO_URL);
  }
}
```

Implement the reasoner: Part C – Service engine wrapper (Control unit)
ControlUnit: the following java codes wrap the hotel reservation services.

```
package hotelreservation;
import junit.framework.*;
public class TestControlUnit extends TestCase {
  private ControlUnit controlUnit = null;
  public TestControlUnit(String arg0) {
    super(arg0);
  }
  protected void setUp() throws Exception {
    super.setUp();
    /**@todo verify the constructors*/
    controlUnit = new ControlUnit();
  }
  protected void tearDown() throws Exception {
    controlUnit = null;
    super.tearDown();
  }
  public void testStartService() {
    User userInfor = null;
    controlUnit.startService(userInfor);
    /**@todo fill in the test code*/
  }
}
```

Testing result is shown in Figure 6.9. As you can see, "ABC Grant Hotel" has the highest rate to match user's preference query.

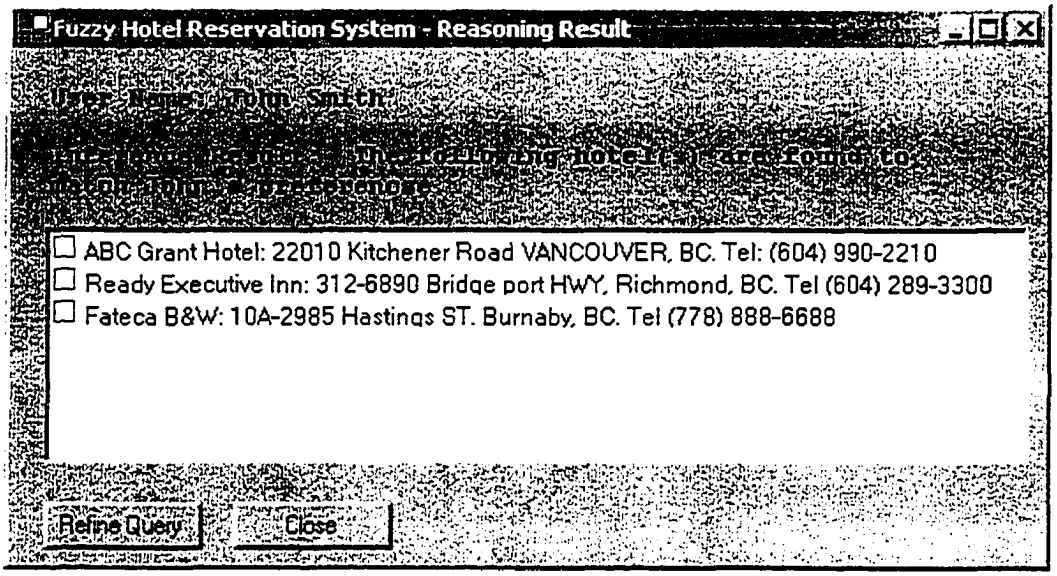

Figure 6.19: Testing Results

6.2.3.3 Storage layers

The main purpose of using database is to create a caching media for fast retrieving, to improve the performance of system. For example, the system will recognize an existing user's preference thus can directly parse hotel's resource without re-creating user's acceptance fuzzyset curves. Storage is also dealing with the hotel registration information as a repository.

Chapter 7
Overview of Enterprises Applications

This chapter focuses on overview of enterprise applications, such as eCommerce, CRM, ERP, PM, Comprehensive operation management.

Section 7.1 eCommerce

eCommerce (electrical commerce) is kind of commerce conducted in cyberspace. It is the execution of real-time business processes with the assistance of Internet technologies. eCommerce is one type of instances of eCommerce. Literally any system with multiple-entity transaction involved can be eCommerce, either the participants are inter-company (B2B) or customer-company (B2C) or internal company (Department to department, D2D). BPEL (Business Process Execution Language) is widely used in business workflow description, Java developers need to publish synchronous and asynchronous Web services and compose them into reliable and transactional business flows. Web service orchestration standards (SOAP Conversation, BPEL4WS and WS-Transaction) are emerging and need to be packaged into a reliable and easy-to-manage software solution. So we've gathered a wealth of information to get you up-to-speed quickly.

BPEL for Web services is an XML-based language designed to enable task-sharing for a distributed computing or grid computing environment - even across multiple organizations - using a combination of Web services. Written by developers from BEA

Systems, IBM, and Microsoft, BPEL combines and replaces IBM's WebServices Flow Language (WSFL) and Microsoft's XLANG specification. (BPEL is also sometimes identified as BPELWS or BPEL4WS.)

Using BPEL, a programmer formally describes a business process that will take place across the Web in such a way that any cooperating entity can perform one or more steps in the process the same way. In a supply chain process, for example, a BPEL program might describe a business protocol that formalizes what pieces of information a product order consists of, and what exceptions may have to be handled. The BPEL program would not, however, specify how a given Web service should process a given order internally. The use case of semantic web in BPEL is modeling the process. Basically we can encode above flow using daml+oil, after the ontology was created, it can be used by a built-in reasoner.

Section 7.2 CRM

CRM is an integrated approach to identifying, acquiring, and retaining customers. By enabling organizations to manage and coordinate customer interactions across multiple channels, departments, lines of business, and geographies, CRM helps organizations maximize the value of every customer interaction and drive superior corporate performance.

Today's organizations must manage customer interactions across multiple communications channels including the Web, call centers, field sales, and dealers or partner networks. Many organizations also have multiple lines of business with many overlapping customers. The challenge is to make it easy for customers to do business with the organization any way they wanted any time, through any channel, in any language or currency, and to make customers feel that they are dealing with a single, unified organization that recognizes them at every touch point. Figure 7.1 shows a typical structure of CRM system.

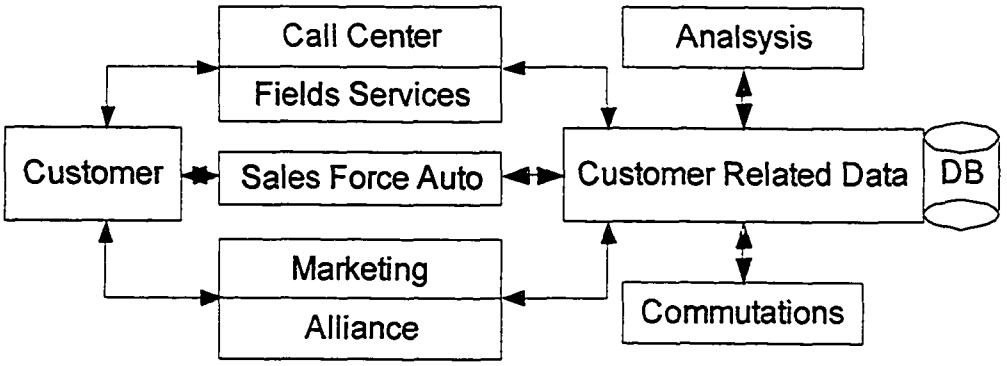

Figure 7.1: Typical structure of CRM system

Section 7.3 ERP

Development and advancement in computer technology have enabled MRPII\ERP to evolve so that it affects more of the organizations and runs more efficiently. ERP/MRP II has moved from running on large, expensive to maintain mainframe computers to faster, cheaper minicomputers. Wide and local area networks have allowed efficient data flow between computers throughout the company, and among customers, suppliers and vendors. Powerful personnel computers on user desktops have enabled client server technology and allowed end users to interact with ERP/MRP II systems through graphical user interfaces. Improved data base technology and software advances, such as object oriented programming, have given MIS staff and end users alike the ability to manipulate data easily. Therefore, organizations need to decide whether to first replace existing business software with a new ERP/MRP II system, or to first integrate new decision support software with the existing business system [35].

ERP automates the tasks necessary to perform a business process—such as order fulfillment, which involves taking an order from a customer, shipping it and billing for it. With ERP, when a customer service representative takes an order, he or she has all the necessary information—the customer's credit rating and order history, the company's inventory levels and the shipping dock's trucking schedule. Everyone else in the company can view the same information and has access to the single database that holds the order. When one department finishes with the order, it is automatically routed via the ERP system to the next department. To find out where the order is at any point, one need only

log in to the system. With luck, the order process moves like a bolt of lightning through the organization [36]. Figure 7.2 shows a typical structure of ERP system.

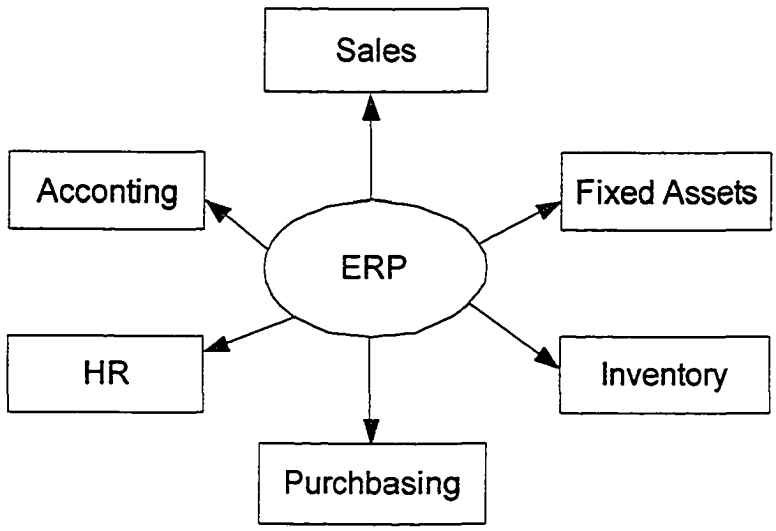

Figure 7.2: Typical structure of ERP system

In the case of Microsoft Greatplains, there are a few integrated modules in the system that composes the whole accounting package all together. The following diagram (figure 7.3) shows the relationship of those modules with the General Ledge as the central posting.

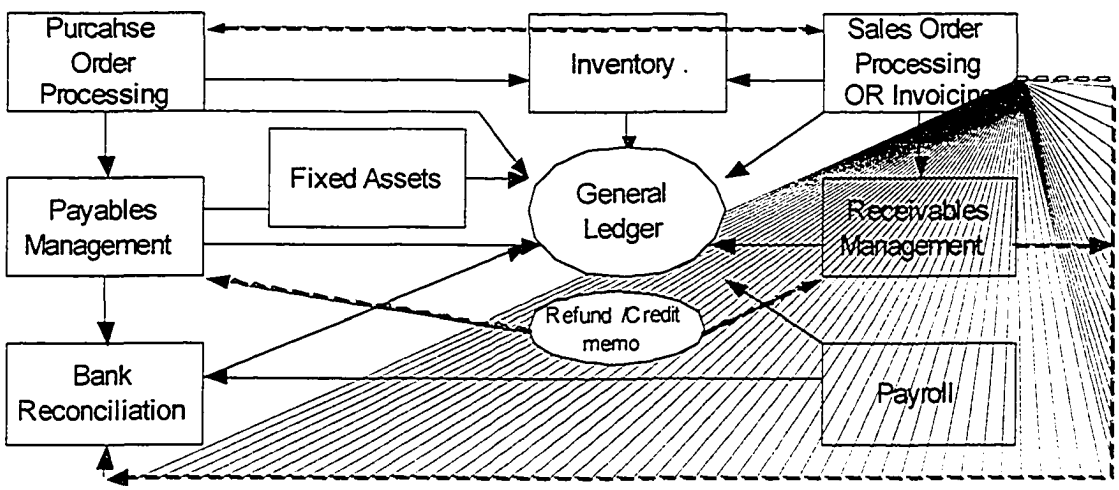

Figure 7.3: Posting to General Ledge

Section 7.4 Project Management

Project management is the combination of knowledge, skills, tools and techniques with broad range of activities to facilitate a particular project. Project management can be described in terms of their five processing components, which include the process of initiating, planning, executing, controlling and closing, see figure 7.4.

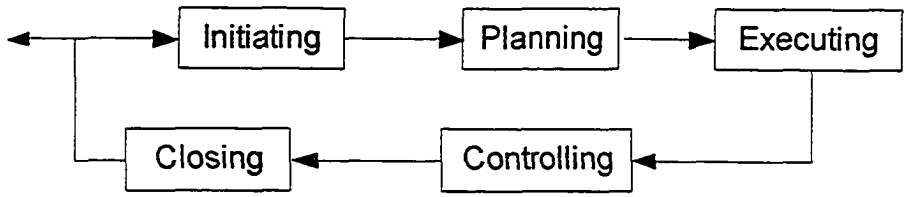

Figure 7.4: Process of managing a project

Project basically has nine important dimensions or measures. Integration, Scope, Cost, Time, Quality, Human Resources, Risk, Communications and Procurement, as shown in figure 7.5.

- Project integration management: legacy, integrating with other systems
- Project scope management: phases, working load
- Project time management: scheduling, milestones
- Project cost management: budgets and actual
- Project quality management: satisfactions and verifications
- Project human resource management: skills, task assignment to staff
- Project communications management: team, efficient communicating
- Project risk management: contract, risk of committing the project
- Project procurement management: purchasing, devices

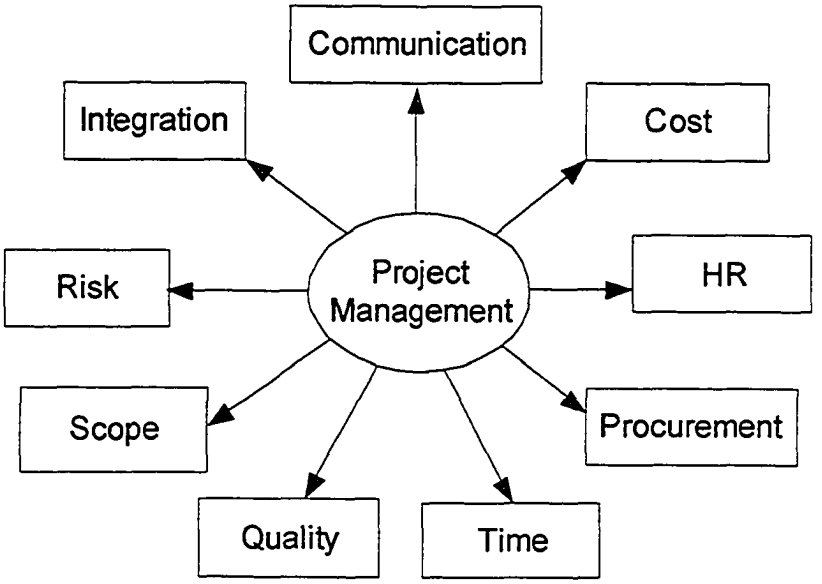

Figure 7.5: Project management nine dimensions

Section 7.5 Comprehensive Operation Management

Comprehensive operation management (COM) is proposed for managing company internal resources and workflow. It aims at maximizing the performance and reducing the cost related to projects. It is based-on project management, but with enhancement with introducing other related modules, such as the following:

- Workforce management: performance, review
- Knowledge management: experience, knowledge
- Contact management: document
- Workflow management: task list
- Billing management: invoicing

Collaborative Adapter of Project Engine (CAPE) is the project from Adaptrust Solutions Inc. The core part of CAPE is the project management (see figure 7.6). With the principle of managing project factors from different angle as explained in section 7.4, management process includes:

- Tracking project status (creation, modification, activation, access)
- Project Team Management

- Project Skill Management
- Project Paycode Management
- Project Contract Management
- Project Scope Management (Project→Sub-project→Task)
- Project Schedule Management
- Project progress/milestone
- Multiple user escalation: assigned user and supervisor
- Project Document Management
- Project report and analysis

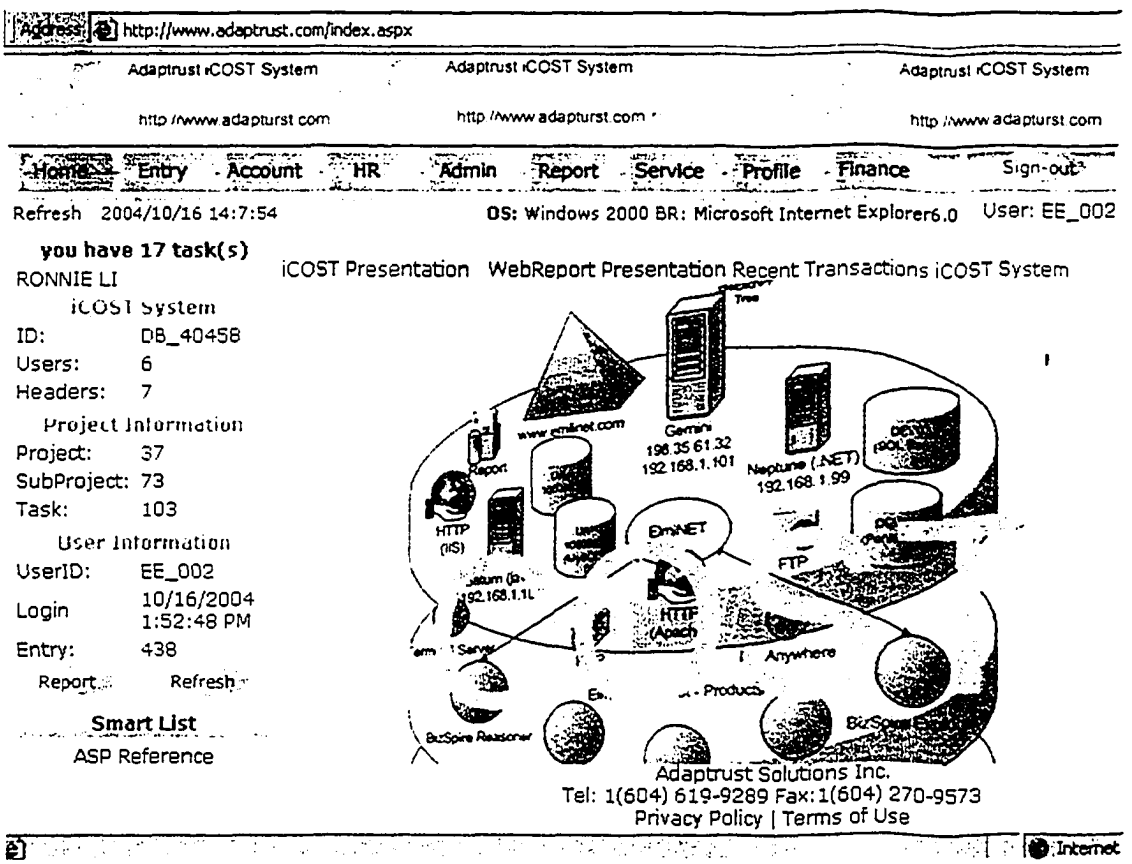

Figure 7.6: Adaptrust CAPE system

Chapter 8
Modeling the Enterprise Applications

Section 8.1 Apply the Semantic Web Technology

Current enterprise software is facing the challenge of integrating multiple systems across different platforms. Many computer information systems exist in a company such as portal, campaign management, ERP and CRM, legacy and other type of COTS. There are two fundamental catalogues of those systems: supply chain and demand chain. How to appropriately integrate and thus make them function efficiently is becoming core concern of operation.

No doubt that adoption of various applications and repositories will continue to develop. This thesis presents the vision on how to integrate those applications using innovated model-driven approach, called Collaborative Ontology Enterprise Planning (COEP). The target of this solution is to achieve seamless integration enterprise front-end and back-end systems over the web.

The Semantic Web is a mesh of information linked up in such a way as to be easily processable by machines, on a global scale. A knowledge representation is most fundamentally a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting. Ontology provides shared and common

expression in a domain and will be a key component of the semantic web because of its ability to represent knowledge.

Web services are self-contained, self-describing web application that can be published, located, and invoked across the Web. Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service to perform the integration between them.

COEP applies the concept of Semantic Web inside the company in the form of Semantic Intranet; it represents enterprises components as interrelated knowledge entities. Modeling those components is the process of representing schema in both function and storage dimensions. The ability of communicating among different applications via knowledge representation consequently makes the whole enterprise infrastructure intelligent and machine-understandable. Using Semantic Web technology, we deploy a Semantic Intranet with built-in integration web services as the prototype of implementation.

The core part of COEP is to build the Semantic Web Services. This process includes modeling the functions and repository by creating ontology for existing applications (in DAML+OIL format). The modeling gives us hierarchical components representation of those applications to ensure the communication with each other.

Section 8.2 Overall design

8.2.1 Ontology based infrastructure

Ontology is the generically for knowledge representation in a certain domain. There exist multiple entities and interactions among those components. Enterprise Application (EA) belongs to software system domain, EA instances include and ERP, CRM, SCM, KM etc. Existing and emerging industry standards in EA orchestration direct those applications to certain formalization. They are following the same terms so that user knows what the software does and how to use it. For example, Sale Order (or Order) means written

confirmation to be delivered to the customer, so Order Number was issued to track the sale with corresponding process in ERP system. Not only for those basic terms are expressed using ontology, also business process and flow can be modeled in ontology, called Process Ontology. The process reflect series reaction of certain event, in the case of Sale Order of CompanyA to CompanyB, after it has been entered and posted into General Ledger, Accounting Receivable will increase and Inventory will decrease, then it notifies the HR payroll for commission. At the same time, the corresponding Purchase Order was processed in CompanyB site, as PO process ontology shown in Figure 8.1.

TERM	ALSO CALLED	Explanation
ERP		Enterprise Resource Planning.
Product	Component, Part.	Can be a Component from a Vendor, as well as a manufactured item of Equipment.
Suppliers	Vendors	Organizations that supply Components or Items that are purchased. A Supplier can be a Vendor or a Manufacturer.
Sale Order	Order	Official file to indicate of sales of products or services to customer

Figure 8.1: PO process ontology

ERP components can be shown as Figure 8.2, it includes Inventory, Supplier, PM (Project Management), HR (Human Resources), AR (Accounting Receivable), SOP (Sales Order Processing), Customer, AP (Accounting Payable), GL (General Ledge), Bill Of Material, POP (Purchase Order Processing).

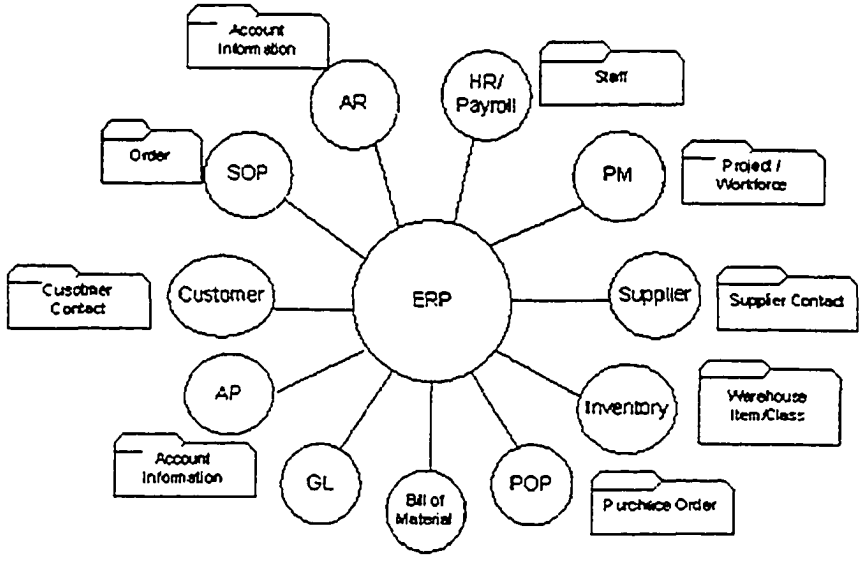

Figure 8.2: An ERP components

Figure 8.3 shows the scenario in which two companies communicate through the Internet. Sale Order ontology of Company A has been sent to Company B.

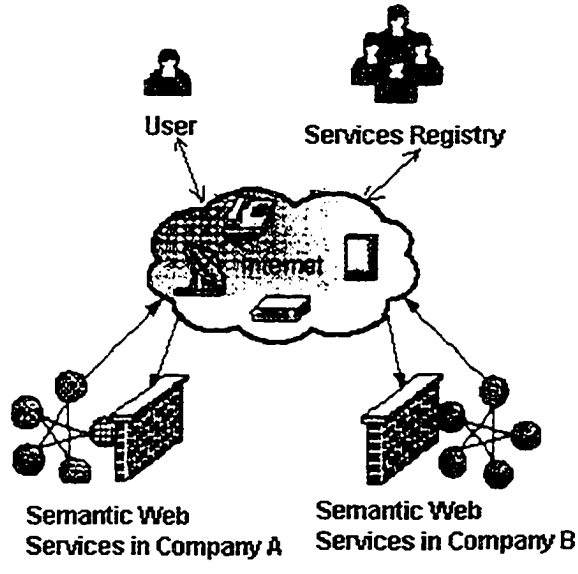

Figure 8.3: Inter-company communication

In summary, Figure 8-4 shows the ontology of enterprise application. It means that Enterprise Application has many Services, these services are used by different group of users (customer, partner or software agent). The enterprise application is guided by standards in order to reflect the management. Physically the enterprise application is implemented via software.

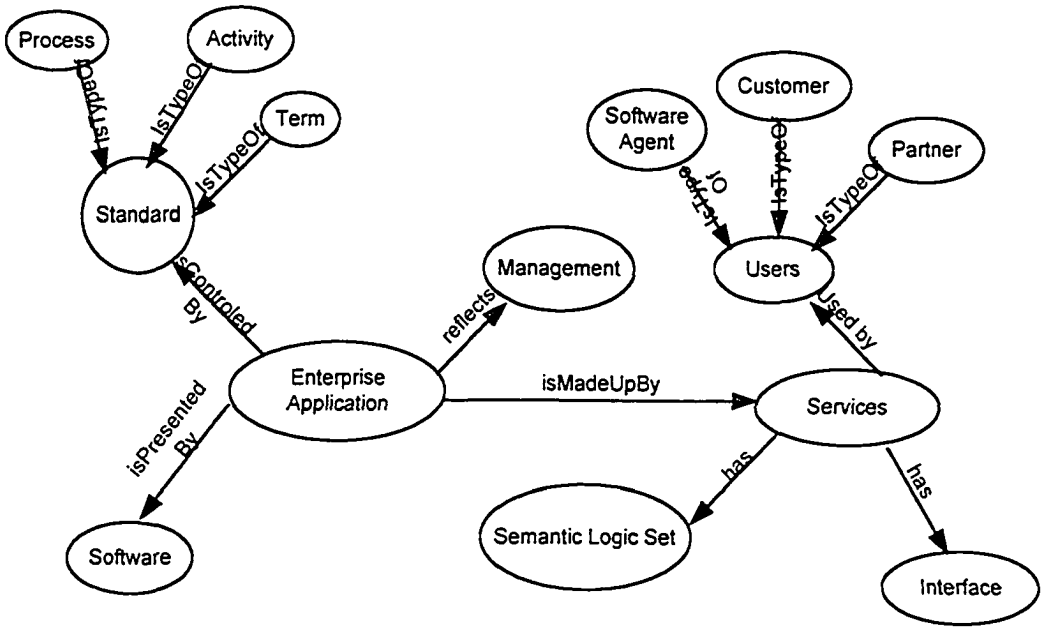

Figure 8.4: A simple ontology of enterprise application

8.2.2 An ERP COEP Model Requirement

Any trades between two companies can be manually done by their accounting clerks through making individual journal entries and exchanging the documents via email, fax, or mails. However, in order to have software agents automatically perform these tasks, machine itself has to understand the concept of the whole process. There are fundamental four requirements to achieve this (see Figure 8.5): machine has to understand:

- Terms in the ERP domain
- Data element
- Business process, activity
- Relationship of inter-data and inter-process

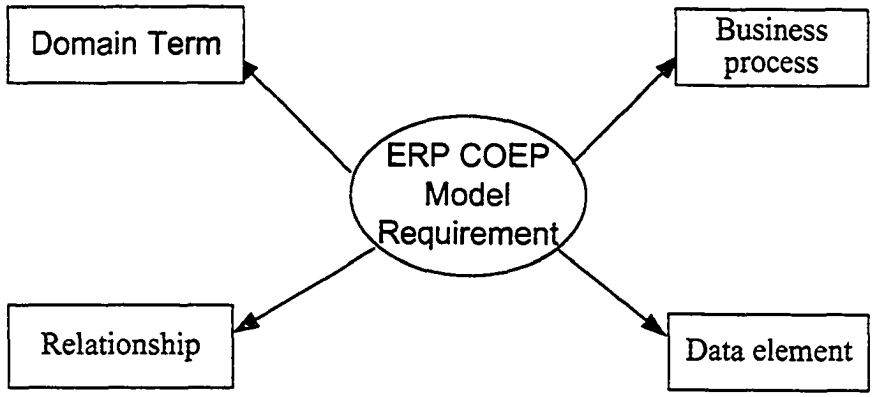

Figure 8.5: An ERP COEP Model Requirement

Section 8.3 COEP ERP model

8.3.1 ERP

We propose COEP to addressing this automatic interaction to the prospective use in B2B, Enterprise Application Integration (EAI), Application Reporting, Data Analysis, KDD, searching. Enterprise Resource Planning (ERP) is known as classic enterprise application. It is efficient for managing the business transactions, easy to operate and reliable to coordinate the Accounting/HR and eminent to generate comprehensive reports. ERP Figure 4-3 shows the components of an ERP system.

8.3.2 COEP data ontology

Like schema of database in ERP system, COEP specifies the "concept "of data element, i.e. data ontology. PO ontology is presented in a DAML file format. Given an instance of a PO, it contains the full semantic description of a transaction. Software agent will know what's the PO all about, such as purchased items, dollar amount, unit of measure, transaction date and so on. This format of expression is also capable of performing OLAP, in the multiple-dimension view. See Figure 8.6.

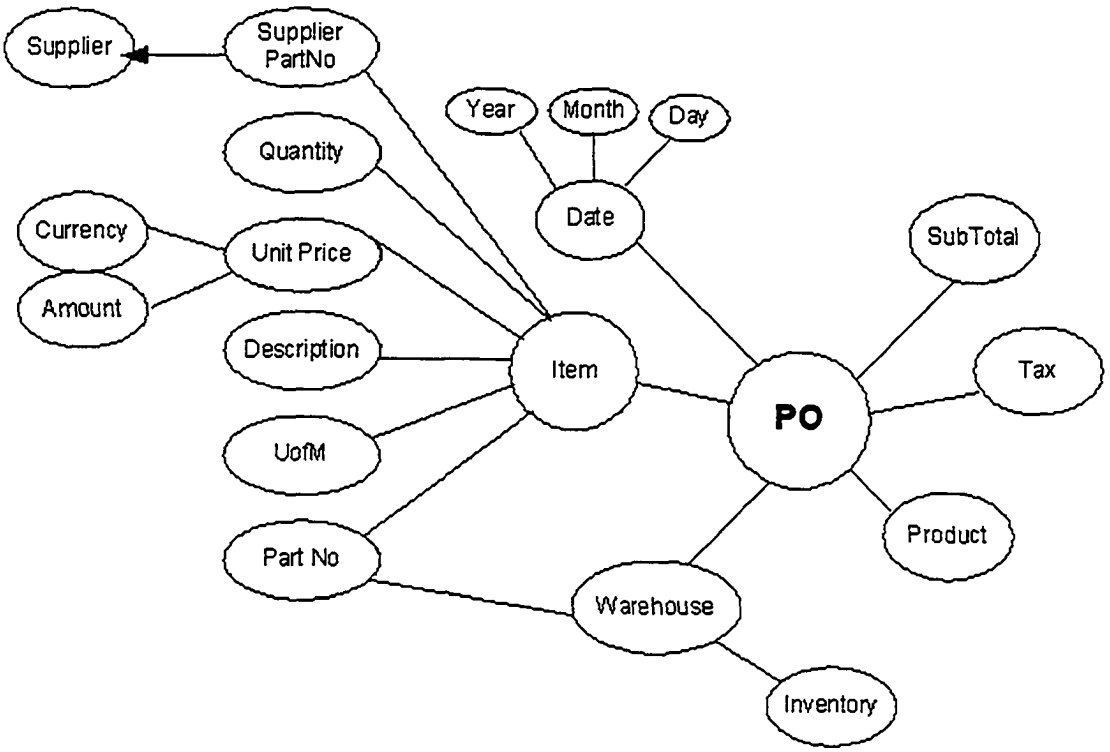

Figure 8.6: A simple PO ontology

Section 8.4 Application of using COEP

COEP has the vision of representing enterprise application in two expect: data and business process. It models in the way of semantics so that they are explicitly expressed to reflect the activities and data flow. Software application includes data and functions which combine and interact together in certain business logic. Any software application is design to perform certain task or process. Most enterprise applications have user interface to grant certain control on it, and return response to the user. They are working as different type of internal or external services. Either business process or data can be seen as knowledge. So the concept of using Semantic Web technology into enterprise application is to provide a way of understand the meaning of those services.

Using Semantic Web Services technology, COEP can help to model the following applications

- E-Business
- Reporting

- Data Analysis
- Integration
- Searching

In the next section, I will explain COEP's applications in detail.

Section 8.5 E-Business

E-business is an open business. Success in the 21st century won't come from smoke and mirrors. Business is simply the voluntary trading of products and services (with money as a mechanism) for the mutual benefit of all. E-business lets companies and customers communicate on the same level. It's a revolution. The overall feature of E-Business is openness and integration [37]. It is established on state-of-the-art technology and also affecting the way that computer are used. We can see that email, B2B, B2C, collaborative services etc are showing that information flow is more reliable and intelligent in 21st century e-business. Any program that provides certain functions can be deployed as service, either internal or external. COEP is capable of modeling applications as service providers. With process ontology support and reasoning mechanism, COEP programs are able to "talk" to each other, expose the interface to the public in the semantic way.

Section 8.6 Reporting

COEP builds application reports based on Extensible Business Reporting Language (XBRL). XBRL formerly code-named XFRML, is an open specification which uses XML-based data tags to describe financial statements for both public and private companies [38]. XBRL benefits all members of the financial information supply chain. On top of the XBRL syntax, another layer is introduced in COEP architecture to model the report. The report itself is meaningful as basic service in the semantic web environment, such as indicating what does the report mean, in which area it applies, who are readers, what does it related and so on. As RDF using XML syntax, through global schema and ontology, COEP makes those XBRL reports understood by software agent in specific domain.

```
e.g
<group>
```

```
<group type="FixedAssets.Machines">
<item period="2004">$123,500</item>
<item period="2003">$100,500</item>
</group>
```

For machine understanding, make the business process meaningful.

Section 8.7 Online Analytical Processing (OLAP)

As one of the most active applications in Business Intelligence, On-line analytical processing is the foundation for enterprise and business applications, including sales and marketing analysis, budgeting, statutory consolidation, profitability analysis, predicated planning, performance measurement and data warehouse reporting [39]. Through OLAP, data set can be analyzed or viewed in multiple dimensions to help decision-making. In COEP, data element is defined as the individual knowledge entity. Data set together form the knowledge pool, therefore, data can be shared and utilized by human and machine. Data was analyzed by the same concept frame no matter distinguished wording. Expression of "Jan 01, 2004 – Mar 31, 2004" is recognized as the same as "first quarter of the current year" in pivot table. The semantic web OLAP service tells user this is the time-dimension. In ERP/CRM system, OLAP is extremely useful in sales, purchase history or Customer visit analysis such as click-stream.

Section 8.8 Integration

Enterprise Application Integration (EAI) has its complexity by nature. EAI can be described as "The process of integrating multiple applications that were independently developed, may use incompatible technology, and remain independently managed." From this point of view, EAI is not a technology per se, but a collection of tools, techniques, and technology, which enable applications to effectively inter-operate with each other [40]. Use of ontology throughout the enterprise applications makes EAI more efficient. COEP has an Inference Engine to explore process rules according to application business logic. For example, a closed product sale to a customer is originated from the front-end transaction, generating a bill or invoice and creating an account receivable that has to be taken into the ERP or financial package being used by the company. We have both front-end and backend transaction ontology as the instances of Knowledge to ensure the two

systems can speak to each other at the transaction level based on business rule, consequently achieve the interoperability of distributed applications.

Section 8.9 Searching

Searching the COEP Web Services is not simply performing key word matching; instead, it is so-called "Intelligent Searching". It means that it is capable of understanding the searches in the semantic level. For example, by typing "Child-care payment software in Vancouver", information about "FAN (First Aboriginal Nation) Payment and Inter-agency Billing System" will be returned, or alternatively, command the user agent to find such service. COEP provide the inner ability of performing the semantic searches with data ontology support.

Section 8.10 COEP Architecture

8.10.1 COEP layers

COEP uses ontology as the knowledge representation method to build Semantic Web Services for enterprise applications. Its architecture includes data storage, transportation, semantic layer, transformation and presentation. Figure 8.7 shows COEP's components.

Components	Description
Presentation	Interface, XML visualization
Transformation	Ontology mapping
Semantic layer	Model Business Logic
Transportation	HTTP, SOAP
Data Storage	XML, database

Figure 8.7: COEP Components

8.10.2 Application/Service Ontology

Application/Service ontology is made up of process ontology and data storage ontology (see Figure 8.8). The principle of COEP is to make enterprise software self-descriptive, self-execution. COEP model applies to all kinds of MIS products, such as ERP, CRM, SCM, and KM etc.

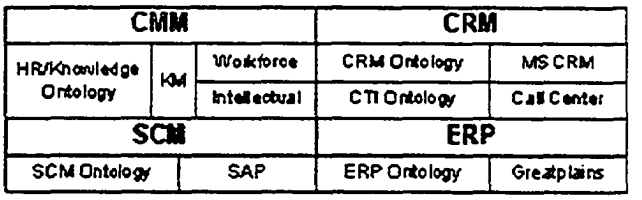

Figure 8.8: COEP application instances

Section 8.11 COEP Components

8.11.1 Data Storage

Data could be any kind of backend repository such as relational database, XML, files. These data can be also encoded in DAML or OWL instances for inquiry.

For example, customer table contains the schema of customer profile with rows of dataset. Being identified by unique customerID as the primary key, customer table is related to sales order processing table (SOP) by customerID as one of the foreign keys in SOP, see Figure 8.9. COEP takes advantage of ontology philosophy in the sense of modeling customer schema with constrains, cardinality and inter-relationship. Based on RDF concept, the modeling describes any instance of customers as a single resource (subject) that contains corresponding property (predicate) and value (objective), as shown in the following DAML encoding.

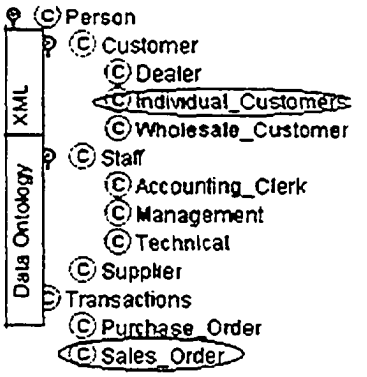

Figure 8.9: A portion of COEP SOP ontology

As introduced in Chapter two, a customer can be represented by a customer ontology. See the DAML+OIL code below

```
<daml_oil:Class rdf:ID="Person">
</daml_oil:Class>
<daml_oil:ObjectProperty rdf:ID="hasName">
  <daml_oil:domain rdf:resource="#Person"/>
  <daml_oil:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</daml_oil:ObjectProperty>
<daml_oil:Class rdf:ID="Customer">
  <rdfs:subClassOf rdf:resource="#Person"/>
</daml_oil:Class>
<daml_oil:ObjectProperty rdf:ID="placedBy">
  <daml_oil:range rdf:resource="#Customer"/>
  <daml_oil:domain rdf:resource="#Sales_Order"/>
</daml_oil:ObjectProperty>
```

8.11.2 Transportation

This layer addresses HTTP, SOAP, TCP/TP etc. Transportation ensures the information routing in Internet cloud.

8.11.3 Semantic layer

Semantic Layer is using DAML-S concept to model the business process of an application, see Figure 8.10.

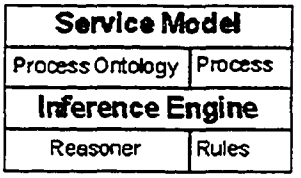

Figure 8.10: Semantic Layer of an application

8.11.3.1 ServiceProfile

An ERP system can be seen as Service, the functional components inside ERP infrastructure are also individual services, as explained before. In any case, a transaction involved in service has three parties: provider, requester, and infrastructure components. For example, the profile of a Purchase Order Processing (POP) Service includes such information as what POP provides, what kind of function it services, what feature and comment of the POP and so on.

8.11.3.2 ServiceProcess

Process is the core part of a service. In Semantic Web Services, process includes ProcessModel and ProcessControlModel. ProcessModel uses AI planning, process language and workflow management concept. Process ontology is capable of modeling

event, parameters, action, state of an application or a functional unit of the application. The processModel describes a service of the following four basic information:

- Input: as input parameters
- Output: as output parameters
- Precondition: state of process before execution
- Effect: effect of the service

Process could be divided as one of the following categories:

- Atomic Process, the realization unit
- Simple Process, has multiple atomic processes
- Composite Process, can be broken into several simple processes

Apart from the basic description of Input/output/precondition/effect to a service, sequence ontology is necessary for multiple processes to specify the interaction procedures such as SPLIT, JOI, ITERATE, LOOP CONTROL, TIMING.

In the case of Purchase Order Processing, a POP service can be viewed as the combination of:

- Issue of purchase order: simple process of purchase order entry
- Items receiving: has sub-process of standard cost management
- Posting to GL: related to general ledger account
- Invoices processing: has sub-process of actual cost, landed cost management

It affects other process such as Accounting Payable, Inventory control, Reporting and so on. Those processes follow certain defined sequence, for example, after purchase order was posted, analysis of PO is from WORK to HISTORY. Variable report happened after both receiving and invoice received. They all can be defined using DAML-S. Through DAML-S, an application is encoded as services entity in semantic level.

8.11.3.3 ServiceGrounding

ServiceGrounding is turning the DAML-S into WSDL to expose the services to public user. It deals with the realization of a service.

8.11.3.4 Inference Engine

Reasoner is made up of rules and facts for logic control and ontology parsing. As there is mechanism to specify the flow of service components by process ontology, inference engine is built to coordinate the control of inter-service on higher level. For example, given an XBRL report ontology, meaning of the report can be parsed and interpreted into data analysis process. COEP uses DAMLJessKB+Jess for matchmaking.

8.11.3.5 Transformation

Ontology mapping, term mapping is directly related to integration. For example, two similar terms with different name could be interpreted using "sameClassAs" with or without constrains. For example, vendor is the same with supplier except vendor does not do manufacturing; the process of check-link is the same as "data integrity verification" process. This layer must specify which part of service description needs to be mapped and how they are mapped.

8.11.3.6 Presentation

The interface is to present the human or machine readable input, output, monitoring of the web service. For instance, In COEP reporting is handled as one of the main components such as POP, SOP. COEP uses XBRL report format, such as for Financial Statement. Tool for data analysis and semantic searching can be deployed through the interface.

Chapter 9 Conclusions

Semantic web technology is a very powerful concept that provides means to represent web contents using ontologies. Both human beings and software agents can explore the Semantic web content. However, usage of XML-based specification languages makes the information stored on the web readable for the software agents. This introduces possibilities of building more intelligent agents capable of providing more sophisticated services to the users.

In order to show that agents can act in a way similar to users, the fuzzy approach has been applied to build a reasoner in the Semantic Web application in order to deal with ambiguous query scenarios. The concept of fuzziness has been introduced and applied into the Semantic Web Services. An architectural model of the semantic reasoner with fuzziness has been illustrated using a simple semantic web application (Hotel Reservation Searching Service). According to the test cases, user can obtain and service responses with a close match of his or her preference.

Ontology is capable of expressing things explicitly. It can express fuzzy membership functions. Fuzzy ontologies can be parsed by using FuzzyJ toolkit.

As the perfect combination of the Semantic Web and Web Services technology, COEP represents the type of software with ontology-based knowledge management methodology. COEP uses the Semantic Web Services as its principle architecture and then encodes the application through DAML-S at semantic level. COEP's concept can be

seamlessly integrated with front-end and back-end enterprise systems; its final goal is to achieve the innovate manufactory of industry software.

Using the Semantic Web technology, an intelligent fuzzy engine can be built in the World Wide Web community. Intelligent components will play an important role in building web-enabled services across the Internet. Semantic Web empowers the interaction of supported knowledge-base applications. By defining shared namespace properly, those reasoners can be deployed with more complex functions to achieve the interoperability of enterprise applications.

The enhancement of the proposed system includes further implementation of a group of full web-based services which are able to automatically search and discover hotel information from the web. In order to augment the performance, a database (for example BerkeleyDB) can be used as embedded database to cache frequency queries. System logs, such as user activity, process monitor and certain statistics are also kept. One of the services can be implemented as the subscription through which user will have the option of subscribing hotel information on the fly.

Chapter 10

Reference

[1] Michael Day, Metadata in a nutshell Information Europe 6 (2), Summer 2001, page. 11.

[2] Rachel Heery & Pete Johnston, with Dave Beckett & Damian Steer (ILRT, University of Bristol) - October 2002 Proceedings of the International Conference on Dublin Core and Metadata for e-Communities, 2002. Florence: Firenze University Press, 2002, pp. 125-132.

[3] Murtha Baca, Introduction to Metadata: Pathways to Digital Information (Los Angeles: J. Paul Getty Trust, 1998

[4] Gruber, T.E. A translation approach to portable ontology specifications. Knowledge Acquisition, 5(2), 199—220, 1993

[5] Scott Farrar, William D. Lewis, and D. Terence Langendoen. An Ontology for Linguistic Annotation 1-2, Fourteenth Innovative Applications of AI Conference, 2002 Edmonton, Alberta, Canada 11-19

[6] Steffen Staab, Michael Erdmann, Alexander Maedche, Stefan Decker. An Extensible Approach for Modeling Ontologies in RDF(S) ECDL Workshop on the Semantic Web 2000, 234 - 253

[7] N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, & M. A. Musen. Creating Semantic Web Contents with Protege-2000. *IEEE Intelligent Systems* 16(2):60-71, 2001.

[8] N. F. Noy, R. W. Fergerson, & M. A. Musen. The knowledge model of Protege-2000: Combining interoperability and flexibility. 2th International Conference on

Knowledge Engineering and Knowledge Management (EKAW'2000), Juan-les-Pins, France, . 2000.

[9] Sean Bechhofer, Carole Goble, Ian Horrocks. *DAML+OIL is not enough.* SWWS-1, Semantic Web working symposium, Stanford (CA), July 29th-August 1st, 2001.

[10] Sean Bechhofer, Ian Horrocks, Carole Goble, Robert Stevens. OilEd: a Reasonable Ontology Editor for the Semantic Web. Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, September 19-21, Vienna. Springer-Verlag LNAI Vol. 2174, pp 396--408. 2001.

[11] Seth Russell Knowledge Representation 3/24/99

[12] Norman Walsh A Technical Introduction to XML from XML.Com, October 03, 1998

[13] Janus Boye XML - What's in it for us from irt.org articles, XML, 28th March 1998

[14] The International SGML/XML Users' Group, weblink: http://www.isgmlug.org/graphics.html 2002

[15] Tim Berners-Lee, Dan Brickley, Dan Connolly, Mike Dean, Stefan Decker, Pat Hayes, Jeff Heflin, Jim Hendler, Ora Lassila, Deb McGuinness, Lynn Andrea Stein. Reference description of the DAML+OIL (March 2001) ontology markup language, URL: http://www.w3.org/2001/09/06-ecdl/slide17-0.html

[16] Semantic Web News and Events Archive. W3C World Wide Web Consortium, URL: http://www.w3.org/2001/sw/news#x20031215b

[17] Deborah L. McGuinness, Frank van Harmelen. OWL Web Ontology Language Overview, W3C World Wide Web Consortium, URL: http://www.w3.org/TR/2003/PR-owl-features-20031215

[18] Semantic Web. W3C World Wide Web Consortium, URL: http://www.w3.org/2001/sw/

[19] Eric Miller. Digital Libraries and the Semantic Web, W3C World Wide Web Consortium, URL: http://www.w3.org/2001/09/06-ecdl/slide17-0.html

[20] Dan Wu, Bijan Parsia, Evren Sirin, James Hendler, and Dana Nau. Automating DAML-S web services composition using SHOP2. In *Proceedings of 2nd*

International Semantic Web Conference (ISWC2003), Sanibel Island, Florida, October 2003. #212

[21] David Martin DARPA Agent Markup Language (DAML) Program official website services section. weblink http://www.daml.org/services/daml-s/2001/05/daml-s.html 2001-06-25

[22] John L. Hawkins What's E-Business ADVISOR ZONE URL: Doc # 05270 January 1999

[23] DARPA Agent Markup Language (DAML) Program official website services section: introduction

[24] http://www.xbrl.org/ official website

[25] Barbara Hammer Softcomputing Vorlesung im SS'03 15th July 2003

[26] Lalit Mohan Pant, Ashwagosha Ganju, CURRENT SCIENCE, VOL. 87, NO. 1, 10 JULY 2004

[27] Boumedine M. and Ramirez-Serrano A., "Fuzzy Knowledge-Based Controller Design for Autonomous Robot Navigation", Journal of Expert Systems with Applications, Vol. 14, No. 1/2, pp. 179-186, January/Spring 1998.

[28] Michael A. Goodrich A Fuzzy Logic Tutorial, July 17, 2001

[29] Orchard, R.A. "FuzzyCLIPS Version 6.04 User's Guide," NRC/ERB-1054. June 1996. NRC 40228.

[30] R. A. Orchard, Section Fuzzyset User's Guide of NRC FuzzyJ Toolkit for the Java(tm) Platform Version 1.2 Integrated Reasoning June 2001

[31] R. A. Orchard, Section Z-Fuzzyset User's Guide of NRC FuzzyJ Toolkit for the Java(tm) Platform Version 1.2 Integrated Reasoning June 2001

[32] R. A. Orchard, Section S-Fuzzyset User's Guide of NRC FuzzyJ Toolkit for the Java(tm) Platform Version 1.2 Integrated Reasoning June 2001

[33] R. A. Orchard, Section Pi-Fuzzyset User's Guide of NRC FuzzyJ Toolkit for the Java(tm) Platform Version 1.2 Integrated Reasoning June 2001

[34] http://www.agentcities.org/News/index_full.jsp agentcities website , 2003

[35] Charles J. Murgiano ERP/MRP II – Replace or Enhance , CPIM WATERLOO MANUFACTURING SOFTWARE

[36] Christopher Koch What Is ERP, article #060100 from www.cio.com website 2004

[37] Michael Kuhbock Thoughts from the EAI Consortium:*An Introduction to the True Definition of Enterprise Application Integration (EAI)* January 15, 2004 Column published in DMReview.com

[38] Gruber, T.E. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2), 199—220, 1993

[39] Scott Farrar, William D. Lewis, and D. Terence Langendoen. An Ontology for Linguistic Annotation 1-2, Fourteenth Innovative Applications of AI Conference, July 28 - August 1, 2002, Edmonton, Alberta, Canada 11-19

[40] Steffen Staab, Michael Erdmann, Alexander Maedche, Stefan Decker. An Extensible Approach for Modeling Ontologies in RDF(S) ECDL Workshop on the Semantic Web 2000 234-253

Appendix

DAML+OIL ontology Specification

The following lists the revised specification of DAML+OIL (http://www.daml.org/2001/03/daml+oil). For detail information, visit DAML official website: http://www.daml.org/

```
<!-- $Revision: 1.7 $ of $Date: 2001/06/06 01:38:21 $. -->

<rdf:RDF
 xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
 xmlns     ="http://www.daml.org/2001/03/daml+oil#"
>

<rdf:Description rdf:about="">
 <versionInfo>$Id: daml+oil.daml,v 1.7 2001/06/06 01:38:21 mdean Exp $</versionInfo>
 <imports rdf:resource="http://www.w3.org/2000/01/rdf-schema"/>
</rdf:Description>

<!-- (meta) classes of "object" and datatype classes  -->

<rdfs:Class rdf:ID="Class">
 <rdfs:label>Class</rdfs:label>
 <rdfs:comment>
  The class of all "object" classes
 </rdfs:comment>
 <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Datatype">
 <rdfs:label>Datatype</rdfs:label>
 <rdfs:comment>
  The class of all datatype classes
 </rdfs:comment>
 <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdfs:Class>

<!-- Pre-defined top/bottom thing/nothing most/least-general (object) classes. -->

<Class rdf:ID="Thing">
 <rdfs:label>Thing</rdfs:label>
 <rdfs:comment>
  The most general (object) class in DAML.
  This is equal to the union of any class and its complement.
 </rdfs:comment>
```

```
  <unionOf rdf:parseType="daml:collection">
   <rdfs:Class rdf:about="#Nothing"/>
   <rdfs:Class>
    <complementOf rdf:resource="#Nothing"/>
   </rdfs:Class>
  </unionOf>
</Class>

<Class rdf:ID="Nothing">
 <rdfs:label>Nothing</rdfs:label>
 <rdfs:comment>the class with no things in it.</rdfs:comment>
 <complementOf rdf:resource="#Thing"/>
</Class>

<!-- Terms for building classes from other classes. -->

<Property rdf:ID="equivalentTo"> <!-- equals? equiv? renames? -->
 <rdfs:label>equivalentTo</rdfs:label>
 <comment>
  for equivalentTo(X, Y), read X is an equivalent term to Y.
 </comment>
</Property>

<Property rdf:ID="sameClassAs">
 <rdfs:label>sameClassAs</rdfs:label>
 <comment>
  for sameClassAs(X, Y), read X is an equivalent class to Y.
  cf OIL Equivalent
 </comment>
 <rdfs:subPropertyOf rdf:resource="#equivalentTo"/>
 <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#subClassOf"/>
 <rdfs:domain rdf:resource="#Class"/>
 <rdfs:range rdf:resource="#Class"/>
</Property>

<Property rdf:ID="samePropertyAs">
 <rdfs:label>samePropertyAs</rdfs:label>
 <rdfs:comment>
  for samePropertyAs(P, R), read P is an equivalent property to R.
 </rdfs:comment>
 <rdfs:subPropertyOf rdf:resource="#equivalentTo"/>
 <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#subPropertyOf"/>
</Property>

<Property rdf:ID="sameIndividualAs">
 <rdfs:label>sameIndividualAs</rdfs:label>
 <rdfs:comment>
  for sameIndividualAs(a, b), read a is the same individual as b.
 </rdfs:comment>
 <rdfs:subPropertyOf rdf:resource="#equivalentTo"/>
 <rdfs:domain rdf:resource="#Thing"/>
 <rdfs:range rdf:resource="#Thing"/>
</Property>
```

```
<rdf:Property rdf:ID="disjointWith">
 <rdfs:label>disjointWith</rdfs:label>
 <rdfs:comment>
  for disjointWith(X, Y) read: X and Y have no members in common.
  cf OIL Disjoint
 </rdfs:comment>
 <rdfs:domain rdf:resource="#Class"/>
 <rdfs:range rdf:resource="#Class"/>
</rdf:Property>

<Property rdf:ID="differentIndividualFrom">
 <rdfs:label>differentIndividualFrom</rdfs:label>
 <rdfs:comment>
  for differentIndividualFrom(a, b), read a is not the same individual as b.
 </rdfs:comment>
 <rdfs:domain rdf:resource="#Thing"/>
 <rdfs:range rdf:resource="#Thing"/>
</Property>


<!-- NOTE: the Disjoint class has been deleted: use disjointWith -->
<!-- or disjointUnionOf instead. -->

<rdf:Property rdf:ID="unionOf">
 <rdfs:label>unionOf</rdfs:label>
 <rdfs:comment>
  for unionOf(X, Y) read: X is the union of the classes in the list Y;
  i.e. if something is in any of the classes in Y, it's in X, and vice versa.
  cf OIL OR
 </rdfs:comment>
 <rdfs:domain rdf:resource="#Class"/>
 <rdfs:range rdf:resource="#List"/>
</rdf:Property>

<rdf:Property rdf:ID="disjointUnionOf">
 <rdfs:label>disjointUnionOf</rdfs:label>
 <rdfs:comment>
  for disjointUnionOf(X, Y) read: X is the disjoint union of the classes in
  the list Y: (a) for any c1 and c2 in Y, disjointWith(c1, c2),
  and (b) unionOf(X, Y). i.e. if something is in any of the classes in Y, it's
  in X, and vice versa.
  cf OIL disjoint-covered
 </rdfs:comment>
 <rdfs:domain rdf:resource="#Class"/>
 <rdfs:range rdf:resource="#List"/>
</rdf:Property>

<rdf:Property rdf:ID="intersectionOf">
 <rdfs:label>intersectionOf</rdfs:label>
 <rdfs:comment>
  for intersectionOf(X, Y) read: X is the intersection of the classes in the list Y;
  i.e. if something is in all the classes in Y, then it's in X, and vice versa.
```

```
    cf OIL AND
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#List"/>
</rdf:Property>

<rdf:Property rdf:ID="complementOf">
  <rdfs:label>complementOf</rdfs:label>
  <rdfs:comment>
    for complementOf(X, Y) read: X is the complement of Y; if something is in Y,
    then it's not in X, and vice versa.
    cf OIL NOT
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#Class"/>
</rdf:Property>

<!-- Term for building classes by enumerating their elements -->

<rdf:Property rdf:ID="oneOf">
  <rdfs:label>oneOf</rdfs:label>
  <rdfs:comment>
    for oneOf(C, L) read everything in C is one of the
    things in L;
    This lets us define classes by enumerating the members.
    cf OIL OneOf
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#List"/>
</rdf:Property>

<!-- Terms for building classes by restricting their properties. -->

<rdfs:Class rdf:ID="Restriction">
  <rdfs:label>Restriction</rdfs:label>
  <rdfs:comment>
    something is in the class R if it satisfies the attached restrictions,
    and vice versa.
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Class"/>
</rdfs:Class>

<rdf:Property rdf:ID="onProperty">
  <rdfs:label>onProperty</rdfs:label>
  <rdfs:comment>
    for onProperty(R, P), read:
    R is a restricted with respect to property P.
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Restriction"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</rdf:Property>

<rdf:Property rdf:ID="toClass">
```

```
 <rdfs:label>toClass</rdfs:label>
 <rdfs:comment>
  for onProperty(R, P) and toClass(R, X), read:
  i is in class R if and only if for all j, P(i, j) implies type(j, X).
  cf OIL ValueType
 </rdfs:comment>
 <rdfs:domain rdf:resource="#Restriction"/>
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Property>

<rdf:Property rdf:ID="hasValue">
 <rdfs:label>hasValue</rdfs:label>
 <rdfs:comment>
  for onProperty(R, P) and hasValue(R, V), read:
  i is in class R if and only if P(i, V).
  cf OIL HasFiller
 </rdfs:comment>
 <rdfs:domain rdf:resource="#Restriction"/>
</rdf:Property>

<rdf:Property rdf:ID="hasClass">
 <rdfs:label>hasClass</rdfs:label>
 <rdfs:comment>
  for onProperty(R, P) and hasClass(R, X), read:
  i is in class R if and only if for some j, P(i, j) and type(j, X).
  cf OIL HasValue
 </rdfs:comment>
 <rdfs:domain rdf:resource="#Restriction"/>
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Property>

<!-- Note that cardinality restrictions on transitive properties, or    -->
<!-- properties with transitive sub-properties, compromise decidability. -->

<rdf:Property rdf:ID="minCardinality">
 <rdfs:label>minCardinality</rdfs:label>
 <rdfs:comment>
  for onProperty(R, P) and minCardinality(R, n), read:
  i is in class R if and only if there are at least n distinct j with P(i, j).
  cf OIL MinCardinality
 </rdfs:comment>
 <rdfs:domain rdf:resource="#Restriction"/>
 <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"/>
</rdf:Property>

<rdf:Property rdf:ID="maxCardinality">
 <rdfs:label>maxCardinality</rdfs:label>
 <rdfs:comment>
  for onProperty(R, P) and maxCardinality(R, n), read:
  i is in class R if and only if there are at most n distinct j with P(i, j).
  cf OIL MaxCardinality
 </rdfs:comment>
 <rdfs:domain rdf:resource="#Restriction"/>
```

```
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"/>
</rdf:Property>

<rdf:Property rdf:ID="cardinality">
  <rdfs:label>cardinality</rdfs:label>
  <rdfs:comment>
    for onProperty(R, P) and cardinality(R, n), read:
    i is in class R if and only if there are exactly n distinct j with P(i, j).
    cf OIL Cardinality
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Restriction"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"/>
</rdf:Property>

<rdf:Property rdf:ID="hasClassQ">
  <rdfs:label>hasClassQ</rdfs:label>
  <rdfs:comment>
    property for specifying class restriction with cardinalityQ constraints
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Restriction"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Property>

<rdf:Property rdf:ID="minCardinalityQ">
  <rdfs:label>minCardinality</rdfs:label>
  <rdfs:comment>
    for onProperty(R, P), minCardinalityQ(R, n) and hasClassQ(R, X), read:
    i is in class R if and only if there are at least n distinct j with P(i, j)
    and type(j, X).
    cf OIL MinCardinality
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Restriction"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"/>
</rdf:Property>

<rdf:Property rdf:ID="maxCardinalityQ">
  <rdfs:label>maxCardinality</rdfs:label>
  <rdfs:comment>
    for onProperty(R, P), maxCardinalityQ(R, n) and hasClassQ(R, X), read:
    i is in class R if and only if there are at most n distinct j with P(i, j)
    and type(j, X).
    cf OIL MaxCardinality
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Restriction"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"/>
</rdf:Property>

<rdf:Property rdf:ID="cardinalityQ">
  <rdfs:label>cardinality</rdfs:label>
  <rdfs:comment>
    for onProperty(R, P), cardinalityQ(R, n) and hasClassQ(R, X), read:
    i is in class R if and only if there are exactly n distinct j with P(i, j)
    and type(j, X).
```

```
    cf OIL Cardinality
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Restriction"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"/>
</rdf:Property>

<!-- Classes and Properties for different kinds of Property -->

<rdfs:Class rdf:ID="ObjectProperty">
  <rdfs:label>ObjectProperty</rdfs:label>
  <rdfs:comment>
    if P is an ObjectProperty, and P(x, y), then y is an object.
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</rdfs:Class>

<rdfs:Class rdf:ID="DatatypeProperty">
  <rdfs:label>DatatypeProperty</rdfs:label>
  <rdfs:comment>
    if P is a DatatypeProperty, and P(x, y), then y is a data value.
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</rdfs:Class>

<rdf:Property rdf:ID="inverseOf">
  <rdfs:label>inverseOf</rdfs:label>
  <rdfs:comment>
    for inverseOf(R, S) read: R is the inverse of S; i.e.
    if R(x, y) then S(y, x) and vice versa.
    cf OIL inverseRelationOf
  </rdfs:comment>
  <rdfs:domain rdf:resource="#ObjectProperty"/>
  <rdfs:range rdf:resource="#ObjectProperty"/>
</rdf:Property>

<rdfs:Class rdf:ID="TransitiveProperty">
  <rdfs:label>TransitiveProperty</rdfs:label>
  <rdfs:comment>
    if P is a TransitiveProperty, then if P(x, y) and P(y, z) then P(x, z).
    cf OIL TransitiveProperty.
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#ObjectProperty"/>
</rdfs:Class>

<rdfs:Class rdf:ID="UniqueProperty">
  <rdfs:label>UniqueProperty</rdfs:label>
  <rdfs:comment>
    compare with maxCardinality=1; e.g. integer successor:
    if P is a UniqueProperty, then if P(x, y) and P(x, z) then y=z.
    cf OIL FunctionalProperty.
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</rdfs:Class>
```

```
<rdfs:Class rdf:ID="UnambiguousProperty">
 <rdfs:label>UnambiguousProperty</rdfs:label>
 <rdfs:comment>
  if P is an UnambiguousProperty, then if P(x, y) and P(z, y) then x=z.
  aka injective. e.g. if firstBorne(m, Susan)
  and firstBorne(n, Susan) then m and n are the same.
 </rdfs:comment>
 <rdfs:subClassOf rdf:resource="#ObjectProperty"/>
</rdfs:Class>

<!-- List terminology. -->

<rdfs:Class rdf:ID="List">
 <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq"/>
</rdfs:Class>

<List rdf:ID="nil">
 <rdfs:comment>
  the empty list; this used to be called Empty.
 </rdfs:comment>
</List>

<rdf:Property rdf:ID="first">
 <rdfs:domain rdf:resource="#List"/>
</rdf:Property>

<rdf:Property rdf:ID="rest">
 <rdfs:domain rdf:resource="#List"/>
 <rdfs:range rdf:resource="#List"/>
</rdf:Property>

<rdf:Property rdf:ID="item">
 <rdfs:comment>
  for item(L, I) read: I is an item in L; either first(L, I)
  or item(R, I) where rest(L, R).
 </rdfs:comment>
 <rdfs:domain rdf:resource="#List"/>
</rdf:Property>

<!-- A class for ontologies themselves... -->

<rdfs:Class rdf:ID="Ontology">
 <rdfs:label>Ontology</rdfs:label>
 <rdfs:comment>
  An Ontology is a document that describes
  a vocabulary of terms for communication between
  (human and) automated agents.
 </rdfs:comment>
</rdfs:Class>

<rdf:Property rdf:ID="versionInfo">
 <rdfs:label>versionInfo</rdfs:label>
```

```
 <rdfs:comment>
  generally, a string giving information about this
  version; e.g. RCS/CVS keywords
 </rdfs:comment>
</rdf:Property>

<!-- Importing, i.e. assertion by reference -->

<rdf:Property rdf:ID="imports">
 <rdfs:label>imports</rdfs:label>
 <rdfs:comment>
  for imports(X, Y) read: X imports Y;
  i.e. X asserts the* contents of Y by reference;
  i.e. if imports(X, Y) and you believe X and Y says something,
  then you should believe it.
  Note: "the contents" is, in the general case,
  an il-formed definite description. Different
  interactions with a resource may expose contents
  that vary with time, data format, preferred language,
  requestor credentials, etc. So for "the contents",
  read "any contents".
 </rdfs:comment>
</rdf:Property>

<!-- Importing terms from RDF/RDFS -->

<!-- first, assert the contents of the RDF schema by reference -->
<Ontology rdf:about="">
 <imports rdf:resource="http://www.w3.org/2000/01/rdf-schema"/>
</Ontology>

<rdf:Property rdf:ID="subPropertyOf">
 <samePropertyAs  rdf:resource="http://www.w3.org/2000/01/rdf-schema#subPropertyOf"/>
</rdf:Property>

<rdfs:Class rdf:ID="Literal">
 <sameClassAs rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Property">
 <sameClassAs rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</rdfs:Class>

<rdf:Property rdf:ID="type">
 <samePropertyAs rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#type"/>
</rdf:Property>

<rdf:Property rdf:ID="value">
 <samePropertyAs rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#value"/>
</rdf:Property>

<rdf:Property rdf:ID="subClassOf">
 <samePropertyAs rdf:resource="http://www.w3.org/2000/01/rdf-schema#subClassOf"/>
```

```
</rdf:Property>

<rdf:Property rdf:ID="domain">
 <samePropertyAs rdf:resource="http://www.w3.org/2000/01/rdf-schema#domain"/>
</rdf:Property>

<rdf:Property rdf:ID="range">
 <samePropertyAs rdf:resource="http://www.w3.org/2000/01/rdf-schema#range"/>
</rdf:Property>

<rdf:Property rdf:ID="label">
 <samePropertyAs rdf:resource="http://www.w3.org/2000/01/rdf-schema#label"/>
</rdf:Property>

<rdf:Property rdf:ID="comment">
 <samePropertyAs rdf:resource="http://www.w3.org/2000/01/rdf-schema#comment"/>
</rdf:Property>

<rdf:Property rdf:ID="seeAlso">
 <samePropertyAs rdf:resource="http://www.w3.org/2000/01/rdf-schema#seeAlso"/>
</rdf:Property>

<rdf:Property rdf:ID="isDefinedBy">
 <samePropertyAs rdf:resource="http://www.w3.org/2000/01/rdf-schema#isDefinedBy"/>
 <rdfs:subPropertyOf rdf:resource="#seeAlso"/>
</rdf:Property>

</rdf:RDF>
```