APPROVAL SHEET

Title of Thesis: Agent-Based Services for the Semantic Web

Name of Candidate: Youyong Zou

Doctor of Philosophy, 2004

Thesis and Abstract Approved: _____

(Timothy W. Finin)

(Professor)

(Computer Science and Electrical Engineering)

Date Approved: _____

Name: Youyong Zou

Address: 4858 W.Braddock Rd #201
Alexandria, VA, 22311

Degree and Date to be conferred: Ph.D., 2004

Date of Birth: July 28, 1972

Place of Birth: JiangXi, P.R.China

Collegiate Institution Attended:

University of Maryland, Baltimore County, Maryland 7/1999—5/2004

Ph.D. Degree in Computer Science.

Nankai University, Tianjin, P.R.China 09/94—07/97

Master Degree in Computer Science.

Nankai University, Tianjin, P.R.China 09/90—07/94

Bachelor Degree in Computer Science.

Major: Computer Science

ABSTRACT

Title of Thesis: Agent-Based Services for the Semantic Web

Youyong Zou, Doctor of Philosophy, 2004

Thesis Directed by: Professor Tim W. Finin

Department of Computer Science and Electrical Engineering

The semantic web suggests having data and information on the web defined and linked in a way that it is both human readable and machine understandable. Machine understandability means that data has been explicitly prepared for programs and softrware agents to reason about and reuse. However, most current software agents have not been designed to run in a web environment: the software agents and web components are working in different models. For example, most software agent frameworks and tools do not support reasoning over content expressed in a mixture of several ontologies. Web components, on the other hand, lack notions of autonomy and a rich message-oriented communication. This thesis suggests adding to a semantic web site a dedicated service agent, capable of understanding the ontologies used by the site and answering queries about the content found on the site. Semantic web tasks are accomplished by the cooperation of personal agents, service agents and semantic web servers. The semantic web language OWL is used as the agent content language for message passing and interaction. An OWL inference engine, F-OWL, has been developed to manage logical sentences explicitly stated in the web documents as well as those that can be inferred. Within this thesis, TAGA system, a Trading Agent System running in the open

Agentcities platform, is used to support the hypothesis and demonstrate how the software agent and semantic web paradigms can be integrated.

We see two main contributions in our work. First, this thesis presents an enhanced semantic web vision involving software agents, semantic web and web services. Both human and software agents can directly access the semantic-marked web pages through web interface. An agent communicates with other agents by exchanging messages encoded in a semantically rich agent communication language. Both web services and agent services are described in a semantically rich language to improve the interoperability. The personal agents and service agents work together to understand the semantic web content, automate web services and better serve humans. Second, the TAGA system attests this semantic web vision and provides a flexible environment for simulating agent-based trading in dynamic markets.

AGENT-BASED SERVICES FOR

THE SEMANTIC WEB

by
Youyong Zou

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2004

Preface

Although this thesis is written in the form of a monograph, some of the results presented have already been published in various articles and reports, which are listed below:

1. **Youyong Zou**, Tim Finin and Harry Chen: "F-OWL: an Inference Engine for the Semantic Web", FAABS III, Greenbelt Maryland, April 2004.
2. **Youyong Zou**, Tim Finin, Li Ding, Harry Chen and Rong Pan: "Using the Semantic Web Technology in Multi-Agent Systems: A Case Study in TAGA Trading Agent Environment", ICEC 2003, Pittsburgh PA, Oct 2003.
3. **Youyong Zou**, Tim Finin, Li Ding, Harry Chen and Rong Pan: "TAGA: a Travel Market Framework in Agentcities", Proceeding of IJCAI-2003, Page 1645-1646, AAAI Intelligent Systems Demonstrations, Acapulco MX, August 2003.
4. **Youyong Zou**, Tim Finin, Li Ding, Harry Chen, and Rong Pan: "TAGA: Trading Agent Competition in Agentcities", IJCAI-2003 Workshop on Trading Agent Design and Analysis, Acapulco MX, August 2003.
5. Li Ding, Tim Finin, Yongmei Shi, **Youyong Zou**, Zhongli Ding and Rong Pan: "Strategies and Heuristics used by the UMBCTAC Agent in the Third Trading Agent Competition", IJCAI-2003 Workshop on Trading Agent Design and Analysis, Acapulco MX, August 2003.
6. Jonathan Dale, Luigi Ceccaroni, **Youyong Zou** and Avigail Agam:" Implementing Agent-based Web Services", AAMAS 2003, Workshop on Challenges in Open Agent Environments, Melbourne Australia, July 2003.

7. Yun Peng, **Youyong Zou**, Xiaocheng Luan, Nenad Ivezic, Michael Gruninger and Albert Jones: "Semantic Resolution for E-Commerce", Proceedings of AAMAS 2002, Page 1037 - 1038 , Bologna Italy, July 2002.
8. Scott Cost, Tim Finin, Anupam Joshi, Yun Peng, Charles Nicholas, Ian Soboroff, Harry Chen, Lalana Kagal, Filip Perich, **Youyong Zou** and Sovrin Tolia: "ITTALKS: a Case Study on the Semantic Web and DAML+OIL", IEEE Intelligent Systems, Volume 17-1 Page 40-47, January/February 2002.
9. **Youyong Zou**, Tim Finin, Yun Peng, Anupam Joshi and Scott Cost: "Agent Communication in DAML World", First GSFC/JPL Workshop on Radical Agent Concepts (WRAC), NASA Goddard Space Flight Center MD, January 2002.
10. Yun Peng, Nenad Ivezic, **Youyong Zou** and Xiaocheng Luan: "Semantic Resolution for E-Commerce", First GSFC/JPL Workshop on Radical Agent Concepts (WRAC), NASA Goddard Space Flight Center MD, January 2002.
11. Lalana Kagal, Filip Perich, Harry Chen, Sovrin Tolia, **Youyong Zou**, Tim Finin, Anupam Joshi, Yun Peng, R. Scott Cost and Charles Nicholas: "Agents Making Sense of the Semantic Web", First GSFC/JPL Workshop on Radical Agent Concepts (WRAC), NASA Goddard Space Flight Center MD, January 2002.
12. Filip Perich, Lalana Kagal, Harry Chen, Sovrin Tolia, **Youyong Zou**, Tim Finin, Anupam Joshi, Yun Peng, R. Scott Cost and Charles Nicholas: " ITTALKS: an Application of Agents on the Semantic Web", First GSFC/JPL Workshop on Radical Agent Concepts (WRAC), NASA Goddard Space Flight Center MD, January 2002.

13. Filip Perich, R. Scott Cost, Tim Finin, Anupam Joshi, Yun Peng, Charles Nicholas, Harry Chen, Lalana Kagal, **Youyong Zou** and Sovrin Tolia: "ITTALKS: an Application of Agents on the Semantic Web", Workshop on Engineering Societies in the Agents World, Prague, July 2001.
14. R. Scott Cost, Tim Finin, Anupam Joshi, Yun Peng, Filip Perich, Charles Nicholas, Harry Chen, Lalana Kagal, **Youyong Zou** and Sovrin Tolia: "ITTALKS: a Case Study in How the Semantic Web Helps", Proceedings of the First International the semantic web Workshop - Infrastructure and Applications for the semantic web, Page 477-494, Stanford CA, 2001.
15. Jonathan Dale, Luigi Ceccaroni, **Youyong Zou** and Avigail Agam: "Fujitsu Evening Organizer Prototype", Technical Report FLA-NARTM02-13, Fujitsu Laboratories of America, 2002.
16. Jonathan Dale, Luigi Ceccaroni, **Youyong Zou** and Avigail Agam: "Implementing Agent-based Web Services", Technical Report FLA-NARTM02-10, Fujitsu Laboratories of America, 2002.

Acknowledgements

First and foremost, I would like to express my sincere thanks to my advisor, Dr. Tim Finin. His support and guidance is crucial to this thesis, but not as much as his great patience and the will to drive research forward.

My research also greatly benefited from numerous discussions with Dr. Yun Peng, Dr. Anupan Joshi, Dr. Scott Cost, Dr. Jonathan Dale and Dr. Francis McCabe. They kindly shared their experience with me and helped me to focus on my scientific pursuit.

I would to thank the current and former members of the LAIT lab and ebiquity group, namely Ye Chen, Li Ding, Rong Pan, Zhongli Ding, Harry Chen, Lalana Kagal, Filip Perich, Sovrin Tolia and Xiaochen Luan. They made my time at UMBC scientifically and personally rewarding.

Mrs. Dorrie Brass reviewed the whole thesis several times and provided invaluable comments and suggestions.

Finally, I would dedicate this thesis to my wife Shanshan, my son Eric and my parents. The enduring support and loving care they gave me are everything I could possibly have asked for.

Table of Contents

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.1 Objective

From the creation of the web at CERN, Geneva, in 1990, its interaction model never changed. The human moved a mouse onto a highlighted web link and clicked on it, thus initializing an HTTP communication between the web browser and the web server; the web server supplied the requested web pages; and the browser displayed the readable web content on the human's screen. All the improvements to the original text-based web (like image, sound, CGI, mpeg or flash) kept this same interaction model.

However, in the past few years, we have seen increased automation of the web interoperation, primarily in the Business-to-Business (B2B) and E-Commerce areas. For example, an E-Commerce application is able to extract content from the HTML- or XML-encoded web pages. The final consumer of the extracted content is a computer program rather than a human being. Therefore, a web site not only provides the web pages for human viewing, but also provides services to the computer programs and software agents. This new interaction model helps to gather information from the unbounded Internet and enables automation of complex tasks on the web.

The realization of this model requires that the programs and software agents understand the content of the web pages. HTML and XML are not sufficiently expressive and lack semantic support. The semantic web suggests marking up the web pages with semantic tags. Rooted in the knowledge representation and ontology, the semantic web languages, including RDF/RDFS, DAML+OIL and W3C's OWL, are more expressive and support further reasoning.

From the definition of the semantic web [Berners-Lee 2001], we learn that the agents play a very important role on the semantic web. The web parts of the semantic web include a web client and a web server. The web server provides the web pages for both the humans and the agents. The web pages for humans are mainly text and image. The web pages for agents are encoded with a semantic rich web language (e.g., RDF or OWL) and made available in the form of the documents on the web. The content of these pages deals with the services: the profile information of the service provider, what services it provides and how to access the services.

However, today's software agents are not designed to work in the web environment. While the agents use the common knowledge base, the billons of web pages on the World-Wide-Web are unlikely to be saved in a single knowledge base. The multiple agents need to share ontologies with each other. The web sites owned by the different people and organizations are unlikely to agree on the same ontology. Thus without prior knowledge of the content or the ontologies used in the web pages, the agents may not know what the web page is talking about, or even worse, different agents may have variant understanding about the same web

page. The agents run on the same home platform or federated platforms, which are considered to be safe and trustworthy. The web is neither a safe place (hacker, DOS attack, etc.) nor trustworthy (fake identity, snoop, Non-paying-buyer, etc.), requiring the ability to decide who can be trusted and the ability to evaluate the credibility of the information sources.

We conclude with the following research questions: a) what the agent's role will be in the future service-based semantic web; b) how the semantic web technologies can be used by the agent system; c) how the agents work on the semantic web.

1.2 Approach

A semantic web site should provide both attractive web pages to the human and robust services to the agents. The agents are able to access the Internet directly, collecting information from the web and other agents and reasoning about the knowledge in the web pages. The web sites supply the web content and describe what they mean by the data. The agents consume the data and decide what to do with it.

To support these claims, we suggest adding to the semantic web site a dedicated service agent, which is capable of understanding ontology and answering queries. As part of the web site, the service agent knows more detailed and extended information about the services provided by the web site and has better understanding of the knowledge than any outside agents. The semantic web tasks are accomplished with the cooperation of the personal agents, the service agents and the semantic web server. We also suggest using the OWL language as

the agent language for message passing and interaction. Compared with XML, SL or RDF, OWL has the improved support for ontology sharing and ontology mapping which are essential for working on the semantic web.

Within this thesis, Travel Agent Game in Agentcities (TAGA) system is developed to support these claims and demonstrate how the software agents and the semantic web can fit together. The TAGA framework uses a distributed peer-to-peer approach based on the standard agent languages, protocols and infrastructure components (FIPA, Agentcities). It exploits the emerging standards for representing ontologies, knowledge and services (RDF, OWL, OWL-S) and web infrastructure (e.g., Sun's Java Web Start). To demonstrate the heterogeneous agent interoperability, several FIPA platform implementations are currently used within TAGA, including Jade [Bellifemine, 2001] and AAP. The travel agent implemented in Jade can be downloaded and executed to instantiate a new TA agent. The GUI interface allows the user to set operating parameters. By modifying or extending the Java code, the TA can support the complex trading and auction strategies. A set of web based monitoring services allows one to see the status of a game, examine the messages being sent or look up the reputation of agents.

Sets of OWL assertions are loaded into the F-OWL reasoning engine. These assertions include facts about the individuals that are members of classes, as well as various derived facts that are entailed (logically implied) by the semantics of OWL. Based on XSB and Flora-2, F-OWL uses frame system to manage knowledge and extract hidden knowledge via

resolution. F-OWL supports multiple rule languages and query languages, provides multiple user interfaces (GUI, command-line and Java API).

1.3 Contributions

We see two main contributions in our work. First, this thesis presents an enhanced semantic web vision involving software agents, semantic web and web services. Consumers of the semantic information are the software agents. The agents, running in an open environment, use the ontology-based semantic web to promote agent communication and interaction. Both human and software agents can directly access the semantically annotated web pages through web interface. An agent communicates with other agents by exchanging messages encoded in a semantically rich agent communication language. Both web services and agent services are described in a semantically rich language to improve their interoperability. The personal agents and service agents work together to understand the semantic web content, automate web services and better serve humans.

Second, the TAGA system implements our semantic web vision and provides a flexible environment for exploring semantic web technology and agent-based approaches. Our current framework allows users to create their own agents to represent a Travel Agent (TA) or Service Agent (SA) and to include these agents in a running game where they will compete with other user-defined and system-provided agents. We hope that this might be a useful teaching and learning tool; not only for multi-agent systems technology, but also for the semantic web languages RDF, OWL and their usage in agent-based systems. Also, we hope that TAGA will be seen as a flexible, interesting and rich environment for simulating

agent-based trading in dynamic markets. Agents can be instantiated to represent customers, aggregators, wholesalers or service providers, who can make decisions about price and purchase strategies based on complex strategies and market conditions. Moreover, simulations like TAGA encourage exploring aspects of e-commerce that go beyond auction theory. TA agents might compete on their ability to better understand the descriptions of services sought, services offered and the basic models of the preferences of their users in order to best satisfy the needs of their clients. These descriptions, of course, will be in a semantic web language like OWL.

1.4 Outline

The rest of the thesis is organized as follows. First of all, the background of the semantic web and agent technologies is introduced. Chapter 3 argues why semantic and agents can and should working together and presents a list of unsolved problems. After that, a detailed description of the TAGA system that has been designed and implemented is given. TAGA is a platform for the semantic web and the agents working together. Chapter 4 explains the architecture of TAGA, the function of different agents, the service descriptions and ontologies used. The research work includes adding a service agent to the semantic web, using OWL as the language for agent communication and designing the F-OWL inference engine. Chapter 5 finishes with a discussion of the work done and an outline of some potential future research.

CHAPTER 2 BACKGROUND

The semantic web is an emerging research area built on the foundations of diverse previous works. First, the semantic web, designed for the Web, is built on top of the existing Web standards. Second, the semantic web adds semantic into the Web. The semantic web languages like DAML and OWL are rooted in knowledge representation and description logic. Third, the semantic web is available in the form of services. Integrating with ongoing industry-driven web services activities is essential for its success. Finally, agents, the consumers of the semantic information, should play important roles on the semantic web. This chapter will discuss each of these areas, as well as the background of TAGA, a trading agent game running in the open Agentcities environment using the semantic web technology.

2.1 Web Standards

To understand the World Wide Web (WWW), we will first look at the Hyper Text Markup Language (HTML), the language used to describe the majority of the existing web pages. Then, we will discuss the eXtensible Markup Language (XML), which serves as the foundation of the semantic web.

2.1.1 Web and HTML

The World Wide Web (WWW), as its name implies, is a globally connected network of distributed documents (web pages). Tim Berners-Lee at CERN developed the first version of WWW program in 1990. He envisioned a system in which a document could be linked to other documents, enabling researchers to share their work across the Internet by simply following a link from one document to another. There are four mechanisms that make WWW work: Hypertext, URL, HTML and HTTP.

Hypertext, introduced by Ted Nelson in the late 1960s, provides a spontaneous method of accessing in-depth information. Rather than reading a document from beginning to end, the reader can jump to another document through the highlighted word that sparked the interest and received more information about that word. Hypertext consists of a hyperlink and a clickable highlight word, icon or graphics that draw people's attention. The hyperlink, usually a URL, points to the information located at any place on the web.

Uniform Resource Locator (URL) is the mechanism used for addressing objects on the Web. A typical URL looks like this:

http://www.yahoo.com/index.html

It is interpreted as: "go to host machine www.yahoo.com, retrieve a document with filename index.html using HTTP protocol". URL consists of a scheme followed by a colon and a scheme-specific part. The scheme specifies a protocol by which the object is accessed and determines the form of the scheme-specific part. The most commonly used schemes are HTTP and FTP. The scheme-specific part of HTTP consists of the domain name of the host machine and the directory of the file, followed by the filename. There

are two other W3C standards related to addressing. Uniform Resource Indicator (URI) is anything that indicates unequivocally a resource, while URL, a subset of URI, only indicates the access to a resource. Uniform Resource Name (URN) refers to a subset of URI that indicates names remaining unique even when the resource ceases to be available.

HyperText Markup Language (HTML) is the language that tells the browser how to display the document content. It was derived from Standard Generalized Markup Language (SGML), which was developed as a framework for defining documents and data. HTML is written as a plain text file. An HTML example is given in Figure 1:

```
<html>
        <head>
                <title> my web site </title>
                <meta content="text/html">
        </head>
        <body>
                <h3>This is my web site </h3>
I am a student at <a href="http://www.umbc.edu" > UMBC </a>
        </body>
</html>
```

FIGURE 1: HTML EXAMPLE

An HTML file consists of elements that are defined by tags. The most common tag in HTML is the anchor tag, indicated by *<A>*. With the *<A HREF="URL">* form, the anchor tag creates a hypertext link to another document by specifying the *URL*. This tag indicates that a web browser should retrieve the document represented by the *URL* if the link is activated. Most of HTML's other tags refer to organizing and presenting of the document. When copied and published to a web server, a HTML document becomes a web page.

HyperText Transport Protocol (HTTP) is a client/server protocol for sending HTML documents. Upon receiving a HTTP request from the web browser (client), the web host (server) locates the files using the provided URL and returns the requested HTML files. The web browser renders the HTML for presentation on the screen.

Despite its popularity, HTML is designed primarily for presentation to humans. It is difficult for the programs to extract content and perform automated processing on the documents. To solve this problem, W3C has developed the Extensible Markup Language (XML), an alternative encoding intended primarily for machine processing.

2.1.2 XML

XML has syntax similar to HTML. Like HTML, XML allows the angle-bracketed tags to be embedded in a text data stream, and uses these tags to provide additional information about the text data. However, there are significant differences between the two markup languages. The first difference is that XML tags relate to the meaning of the enclosed text, whereas HTML tags specify how to display the enclosed text. The second difference is that XML is extensible. XML supports defining new tags to describe the content in a particular type of document. HTML limits its use to only those tags that have been predefined in the W3C HTML specification.

From the XML example in Figure 2, we can see that XML consists of several components that describe the makeup of the different parts of a document. The following is a list of the major XML components:

```
<?xml version="1.0"?>
<PERSON>  <NAME>
        <FIRSTNAME>John</FIRSTNAME>
        <LASTNAME>Smith</LASTNAME>
</NAME>  </PERSON>
```

FIGURE 2: XML EXAMPLE

- Element tag: There are three kinds of element tags in XML: start tag, end tag, and empty-element tag. A start tag consists of a name and a set of optional attributes, surrounded by the angle brackets. Each attribute is a name/value pair, separated by an equal sign. An end tag consists of the name from a previous start tag, but being proceeded by a slash (“/”) and without any attribute. Every start tag must have exactly one matching end tag. An empty-element tag is indicated by a slash just before the closing bracket;
- Entity reference: Because of the rigid structure of XML, some pieces of information must be specially encoded. Entity reference is a way of referring to a piece of data by using a special name. For example, ampersand character “&” appears as “&”;
- Processing instrument: XML document includes special commands passing along to the program that processes or views the XML document. In the XML example above, *<?xml version=“1.0”?>* is a processing instrument;
- Document Type Declaration (DTD): DTD describes the tags and attributes used in the document, along with the relationships between them. An XML document is regarded as a valid document if it conforms to the rules defined in the associated DTD.

XHTML 1.0 is the W3C Recommendation for the next generation web language. It is a

reformulation of HTML 4.01 in XML, combining the strength of HTML with the power of XML.

As a representation language, XML provides essentially a mechanism to declare and use simple data structures and thus leaves much to be desired as a language of expressing complex knowledge. There is no special meaning associated with attributes or content elements. Attributes might be used to specify supplemental information or single-valued properties, while element content might be used to describe properties of an object or group-related items. The same term may be used with different meanings in the different contexts, and different terms may be used for the items that have the same meaning. Also, XML is intended for exchanging data between parties who have agreed on definitions beforehand. The programs might not understand a new XML vocabulary because of the lack of semantics, thus being prevented from achieving the reliably interoperation.

Recent enhancements to the basic XML, such as XML Scheme or XSL, address some of the shortcoming, but do not result in an adequate language for representing and reasoning about the knowledge. Approaches like ebXML and Meta-Data Coalition suggest developing an open XML-based infrastructure and enabling the interoperability of global use of electronic business information. However, those approaches only represent different terminologies used in different types of businesses. Lacking semantics is the essential problem for XML. The need for understandable knowledge leads us directly to the AI's ontology.

2.2 Ontology

In philosophy, ontology is a discipline that studies the nature of existence. Tom Gruber gives the definition of ontology in the context of AI [Gruber 1993]:

> *[An ontology is a specification of a conceptualization. An ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents.]*

This definition states that the ontology defines the terms used to describe and represent an area of knowledge. People and applications use the ontology to share the domain information. An ontology includes the computer-interpretable definitions of the basic concepts and their relationships. The knowledge may cover a single domain or span multiple domains.

This definition also suggests that the ontology is designed to enable knowledge sharing and reusing among multiple agents. In this context, the ontology is the specification making ontological commitments. An agent is considered committing to an ontology if its observable behaviors are consistent with the definitions in the ontology.

To be used within an application, an ontology must be delivered using some concrete representations. The field of Knowledge Representation (KR) has long been a focal point of research in the Artificial Intelligence community. Expressed in a logic-based language, the KR languages can make accurate and meaningful distinctions among the classes, properties, and relationships. There are various KR languages, with varying characteristics in terms of expressiveness, ease of use and computational complexity.

They fall into three categories: vocabularies defined with natural language, languages based on objects such as frames, and languages based on predicates expressed in logic such as Description Logic.

Vocabularies support the creation of purely handcrafted ontologies with the simple tree-like inheritance structures. For example, the Yahoo hierarchy has a hierarchical structure that is composed of the parent-child and is-a relationships. Although the vocabularies provide great flexibility, the lack of any structure in the representation can lead to difficulties with maintenance or preserving consistency. The single inheritance provided by the tree structure has proven limiting.

Frame-based systems are structured around the notion of frames representing the collections of instances. Each frame has an associated collection of slots or attributes that can be filled by values or other frames. In particular, frames allow the frame taxonomy in the slot. This hierarchy can then be used for inheritance of slots, allowing a sparse representation. As well as frames representing concepts, a frame-based representation may also contain instance frames, which represent particular instances. One of the most well-known frame systems is Ontolingua [Fikes 1997].

An alternative to frames is logic, notably Description Logic (DL). DL describes knowledge in terms of concepts and relationships that are used to automatically derive classification taxonomies. A major characteristic of DL is that the concepts are defined in terms of descriptions using roles and other concepts. DL supports a number of reasoning

services that allow the construction of classification hierarchies and the consistency checking of the descriptions. Two of the most well known DL systems are Classic and FaCT [Horrocks 1998].

The definition of a general exchange language for ontology is the subject of many research efforts in the ontology field. One of the most well known languages is Knowledge Interchange Format (KIF) [Genesereth, 1992], a language designed for knowledge interchanging among disparate computer systems. KIF has the following essential features:

- Declarative semantic: The expressions in KIF are understandable without being appealed to an interpreter for manipulation. In this way, KIF differs from other languages that are based on specific interpreters, such as Prolog.
- Logical comprehensive: KIF supports the expression of arbitrary logical sentences. In this way, KIF differs from the relational database languages (like SQL) and the logic programming languages (like Prolog).
- Knowledge about knowledge: KIF allows the user to make the knowledge representation decisions explicitly and permits the user to introduce new knowledge representation constructs without changing the language.

Ontology can enhance the function of the web in many ways. Ontology can be used in a simple fashion to improve the accuracy of web searching. As a result, the search program can look for only those web pages referring to a precise concept, rather than those using ambiguous keywords. Additionally, more advanced applications can use ontology to

relate the information on a web page to the associated knowledge structures and inference rules, helping applications to understand the domain and improve the flexibility.

2.3 Semantic Web

The semantic web defines and links the web data in both human readable and machine understandable form. “Human readable” refers to the traditional text/image web documents intended for machine display and human consumption. “Machine understandable” means that the data has been explicitly prepared for reasoning and reusing across various applications. The semantic web models the semantics of the web information and covers the aspects from knowledge representation, databases, information retrieval and digital libraries to multi-agent systems, natural language processing and machine learning.

The first part of this section reviews the Semantic Web Vision brought up by Tim Berners-Lee, inventor of the WWW. Then various previous and ongoing works on the semantic web language are discussed.

2.3.1 Semantic Web Vision

The semantic web is a vision (Figure 3) [Berners-Lee 2001]: making web information practically understandable by a computer program.

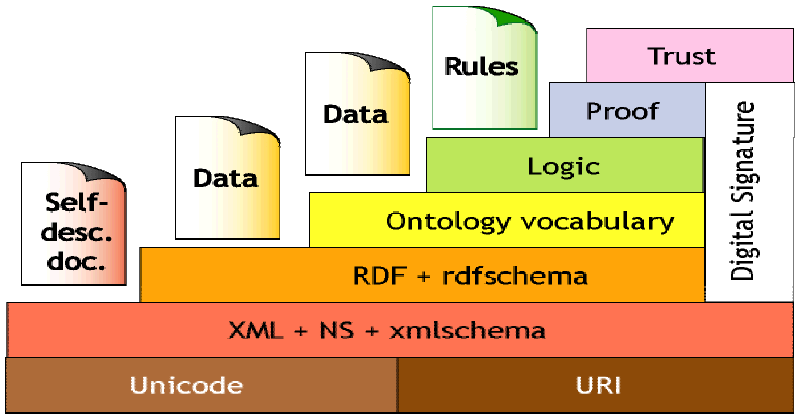

FIGURE 3: BERNERS-LEE'S SEMANTICWEB VISION

This vision is a layered architecture, starting with the foundation composed of URI and Unicode. On top of URI sits syntactic interoperability in the form of XML. As the foundation, XML provides a set of rules for creating vocabularies that can bring structure to both documents and data on the web. XML gives clear rules for syntax; XML Schemas then serve as a language for restricting the structure of XML documents and a method for composing XML vocabularies. On top of XML lies the data interoperability layer: RDF and RDF Schema (RDFS). RDF imposes semantic constraints on the meaning of these XML documents with a clear set of rules for providing simple descriptive information. RDFS then provides a way to combine those descriptions into a single vocabulary. On top of RDF is ontology, allowing the further description of objects and their interrelations and enabling the programs to reason over the description. Logic, Proof and Trust sit on top of ontology sequentially. The logic layer provides an interoperable language for describing the sets of deductions one can make from a collection of data. The proof language describes the steps taken to reach a conclusion from the facts, which support the proving of whether a statement is true. The proof can then be passed around and verified, providing shortcuts to new facts in the system without having each node conduct the

deductions itself. The digital signature layer runs right up the side of the stack, emphasizing its widespread utility. Based on mathematics and cryptography, a digital signature is used to unambiguously verify that one wrote a certain document and to allow content from a layer to be labeled with an assured provenance.

Once all these layers are in place, we will envision a system where we can place trust in the data we are seeing, the deductions we are making, and the claims we are receiving. The final goal is to make a user's life easier by aggregately creating new and trusted information over the web.

To make the semantic web vision a reality, supporting standards and policies are needed to enable machines to make more sense of the web and make the web more useful for the human. Since XML is emerging as the standard language for data interchange on the web, it is desirable that the future semantic web language also uses XML syntax. Many XML-based semantic web languages have been developed in the last few years: Simple HTML Ontology Extensions (SHOE), Resource Description Framework Schema (RDFS), Ontology Inference Layer (OIL), Darpa Agent Markup Language (DAML) and W3C Web Ontology Language (OWL).

2.3.2 SHOE

SHOE [Heflin 2000] [Luke 2000], an HTML-based knowledge representation language developed at University of Maryland, College Park, is the first language intended for incorporating semantics into the web.

As a superset of HTML, SHOE adds the tags to embed semantic data into web pages. These tags are divided into two categories. The first category is for constructing ontologies. SHOE ontologies are sets of rules that define what kinds of assertions SHOE documents can make and what these assertions mean. For example, a SHOE ontology might define that a SHOE document declares some data entity as a "dog" which is permitted to have a "name". The second category is for annotating web documents to conform to one or more ontologies, declare data entities, and make assertions about those entities under the rules prescribed by the ontologies.

The following figure is a SHOE example. It defines that "Chair" is both "Professor" and "AdministrativeStaff".

```
<shoe xmlns="http://www.cs.umd.edu/projects/plus/SHOE/"
version="1.0">
    <DEF-CATEGORY NAME="Chair" ISA="AdministrativeStaff">
    <DEF-CATEGORY NAME="Chair" ISA="Professor">
</shoe>
```

FIGURE 4: SHOE EXAMPLE

SHOE gives HTML authors a simple but powerful way to encode useful knowledge into the web documents and offers the software agents a much more sophisticated mechanism for knowledge discovery than the keyword-based search engine. SHOE can greatly expand the speed and usefulness of the software agents on the web by removing the single most significant barrier to their effectiveness: a need to comprehend text and graphical presentation as people do [Luke 2000].

2.3.3 RDF and RDF Schema

Resource Description Framework (RDF) provides the interoperability for applications to exchange machine-understandable information on the web. RDF includes three basic object types:

- Resource: A resource is anything addressable by a URI, for example, a web page or a specific part of a web page;
- Property: A property is a specific attribute or relationship being used to describe a resource;
- Statement: A statement specifies the value of a property for a particular resource.

The following figure is a simple RDF example:

```
<rdf:Description rdf:about="http://www.foo.com/website">
      < abc:webadmin rdf:resource="http://www.foo.com/Bob" />
</rdf:Description>
```

FIGURE 5: RDF EXAMPLE

The example states that Bob is the web administrator of www.foo.com's website. A RDF file can be parsed into a set of triples, which include three parts: subject, predicate and object. Figure 6 is the triple of the above RDF example:

Subject	http://www.foo.com/website
Predicate	http://www.foo.com/webadmin
Object	http://www.foo.com/Bob

FIGURE 6: RDF TRIPLE EXAMPLE

The modeling primitives provided by RDF are very basic. They provide only binary relationships between ground terms. RDF Schema (RDFS) is designed as a data-typing model for RDF and can be used to describe further modeling primitives such as defining subclass and type relationship, creating properties and classes, and specifying the range

and domain constraint. With these extensions, RDFS enables the definition of schema in an object-oriented manner.

The following example sets the restriction that the web administrator must be the employee of the company.

```
<rdfs:Class rdf:about="&foo;Employee"
        <rdfs:subClassOf rdf:resource="&foo;Person"/>
</rdfs:Class>
<rdfs:Class rdf:about="&foo;Person"/>
<rdfs:Class rdf:about="&foo;Website"/>
<rdf:Property rdf:about="&foo;WebAdmin"
        <rdfs:domain rdf:resource="&foo;Website"/>
        <rdfs:range rdf:resource="&foo;Employee "/>
</rdf:Property>
```

FIGURE 7: RDFS EXAMPLE

The following figure is the model of the RDFS example in Figure 7.

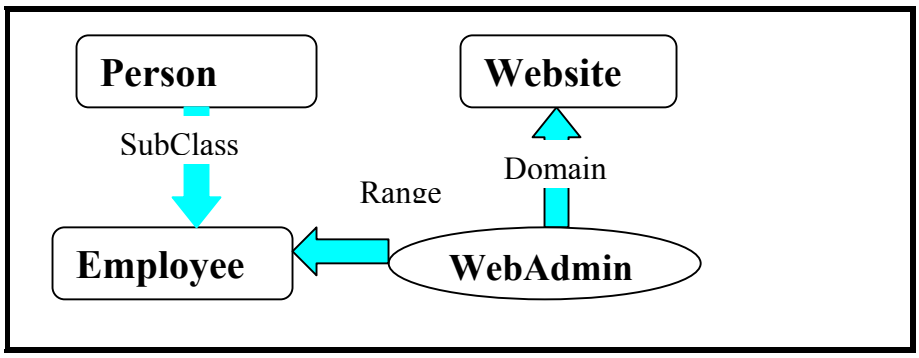

FIGURE 8: RDF MODEL

RDF and RDFS emphasize the facilities enabling automatically web resource processing and can be used as metadata in a variety of applications. For example, they can be used to improve the capabilities of web search engine; to describe the content and content relationships at a particular web site or digital library; to support the intelligent software agents in sharing and exchanging knowledge; to describe intellectual property rights of the web pages.

One problem with RDF and RDFS is their lack of a clear distinction between the object and meta-levels. In RDF and RDFS, properties are the central modeling primitive. This model is very hard for the knowledge modelers to understand and use. The second problem is that the semantics of their primitives are loosely defined. There is no inference model that precisely fits the semantics of the RDF modeling primitives. The third problem is that RDF and RDF Schema allow only simple semantics to be associated with identifiers. For example, they cannot specify that the "Person" and "Car" are disjoint. Other RDF-based languages, including OIL, DAML and OWL, try to solve these three problems by introducing description logic into the web.

2.3.4 OIL

The Ontology Inference Layer (OIL) [Fensel 2000], a project proposed by a group of European researchers with the support of the European Commission, suggests a web-based representation and inference layer for ontologies. Compatible with RDFS, OIL includes a precise semantic for describing term meanings and implied information.

OIL unifies three important aspects provided by three different research fields:

- Description Logic: OIL inherits the formal semantics and the efficient reasoning support from Description Logic. FaCT [Horrocks 1998] system provides an efficient reasoner for OIL language. Subsumption is decidable in OIL [Fensel 2000]. Other reasoning tasks, like instance classification, query subsumption and query answering over classes and instances can be reformulated in terms of subsumption checking;

- Frame-Based System: Based on the notion of concept, super-classes and attributes, OIL incorporates the essential modeling primitives of frame-based systems. In addition to the attributes of a class, the relationship in OIL can be defined as an independent entity having a certain domain and range;
- Web: The syntax of OIL is based on W3C's RDF and RDFS, which provide two important contributions: a standardized syntax for writing ontologies and a standard set of modeling primitives including subclass, domain and range relationships.

The following figure is an OIL example.

```
class-def Product
  slot-def Price
        domain Product
  class-def DellNotebook
        subclass-of Product
   slot-constraint Price
    has-value "$779"
```

FIGURE 9: OIL EXAMPLE

This example defines the property "Price" for class "Product". The instance "DellNotebook" has super class "Product". As a result, "DellNotebook" inherits the property "Price", which has value "$779".

2.3.5 DAML

The Darpa Agent Markup Language (DAML) project is the joint effort from US DAML group and Europe Semantic Web Technologies, supported by the United States Defense Advanced Research Projects Agency (DARPA). The aim of this project is to achieve semantic interoperability between web pages, databases and programs. The essential task is to develop DAML language, a universal semantic web markup language that supports

software agents and other applications. Based on RDF/RDFS and benefited from SHOE/OIL, the DAML language provides a set of tools for programmers to incorporate broad concepts into their web pages.

The first version of the DAML language, DAML-Ont, was published in October 2000. It defines the basic concepts such as class, subclass, property, and subproperty. The second version, DAML+OIL, was published in December 2000. It borrows concepts like inference and slot constraint from OIL. The latest version, DAML+OIL+DT, published in March 2001, adds the XML data type into the DAML language.

The following figure is a DAML class definition:

```
<DAML:Class rdf:ID="Adult">
  <DAML:intersectionOf rdf:parseType="DAML:collection">
    <DAML:Class rdf:about="#Person"/>
    <DAML:Restriction>
      <DAML:onProperty rdf:resource="#age"/>
      <DAML:hasClass rdf:resource="http://www.DAML.org/2001/03/DAML-ex-dt#over17"/>
    </DAML:Restriction>
  </DAML:intersectionOf>
</DAML:Class>
```

FIGURE 10: DAML EXAMPLE

This example defines the class “Adult” as a “Person” whose property “age” has value “over17”.

In addition to inheriting the modeling primitives from RDF and RDFS, DAML provides a method of stating relationships and restrictions, including inverses, unambiguous properties, unique properties, lists, restrictions, cardinalities, disjoint and data types.

DAML can dramatically improve traditional information retrieval because the semantics will improve the quality of retrieval results. It is expected to foster the use of the software agents and to aid both humans and programs in finding and using information.

2.3.6 OWL

Web Ontology Language (OWL) is part of the growing stack of W3C recommendations related to the semantic web. OWL is designed for the applications to process the content of information instead of just presenting information to humans. By providing additional vocabularies along with formal semantics, OWL facilitates greater machine interpretability of web content than that supported by XML, RDF or RDFS.

Evolved from the DAML language, OWL removes a set of ambiguous definitions and adjusts the confusing words used in the ontology. OWL provides three increasingly complex sub-languages:

- OWL Lite: It supports the users who primarily need a classification hierarchy and simple constraints. For example, while supporting cardinality constraints, OWL Lite permits only cardinality values of 0 or 1. It is simpler to provide the supporting tools for OWL Lite than for other two sub-languages;
- OWL DL: It supports the users who want the maximum expressiveness while retaining computational completeness and decidability. OWL DL includes all OWL language constructs, but can be used only under certain restrictions. For example, while a class may be a subclass of many classes, it cannot be an instance of another class;

- OWL Full: It supports the users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantee. For example, a class in OWL Full can be treated simultaneously as an individual in its own and as a collection of individuals.

The ontology developers adopting OWL need to choose the sub-language best fits their needs. The choice between OWL Lite and OWL DL depends on the extent to which users require the more expressive constructs provided by OWL DL and OWL Full. The choice between OWL DL and OWL Full depends on the extent to which users require the meta-modeling facilities of RDFS. When using OWL Full as compared to OWL DL, reasoning support is less predictable since complete OWL Full implementations do not currently exist [Peter 2003].

2.4 Web Services

Along with the growing need for interoperability among web-based applications, web services are expanding rapidly. Web services are modular, self-describing and self-contained applications accessible over the Internet. Web services enable the user to build dynamic context-driven web-based applications and combine those applications to perform complex operations.

2.4.1 Web Services

Web services, as the name suggests, are services offered via the web. W3C Web Services group gives the following definition of web services [WSA 2003]:

[A web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.]

This definition reveals that XML is the base technology of web services. Web services are composed of three parts: service description, service discovery and message encoding. The features of web services include [WSA 2003]:

- Interoperability: Web services extend beyond the programming languages and operate on the "system boundaries";
- Modularity: Web services use the component object model and can be reused or combined as new services;
- Versatility: Web services have been developed to be easily accessible by both humans and software agents;
- Ubiquity: Web services use and respect exist web infrastructure. As a result, web services are accessible from anywhere on the Internet.

A web service can be as simple as invoking a single web-accessible computer program that does not rely upon other web services and has no further interaction beyond a response. For example, a weather service that returns the temperature when given a zip code would be in this category. Alternately, a web service can be complex, being composed of multiple web services and often requiring more interactions between the service consumers and the service providers. For example, a travel organizing service uses the hotel and airline booking services and asks the customers to choose among multiple travel preferences and recommended travel packages.

Once deployed, the web services providers turn to server mode, waiting to be discovered and invoked by the service consumers. In a typical scenario, a service consumer sends a request message to a service provider at a given URL using the SOAP protocol over HTTP. Upon receiving the message, the service provider processes the request and returns a result message to the service consumer if needed.

The service consumers and service providers are typically businesses, making web services predominantly B2B transactions. An enterprise can be both the service provider and the service consumer. For example, a travel agent is in the consumer role when it checks the local weather using a weather service, and in the provider role when it supplies the customers with the travel organizing service.

2.4.2 Web Services Architecture

W3C's Web Service Architecture (WSA) working group suggested the layers in the following figure for web services architecture [WSA 2003].

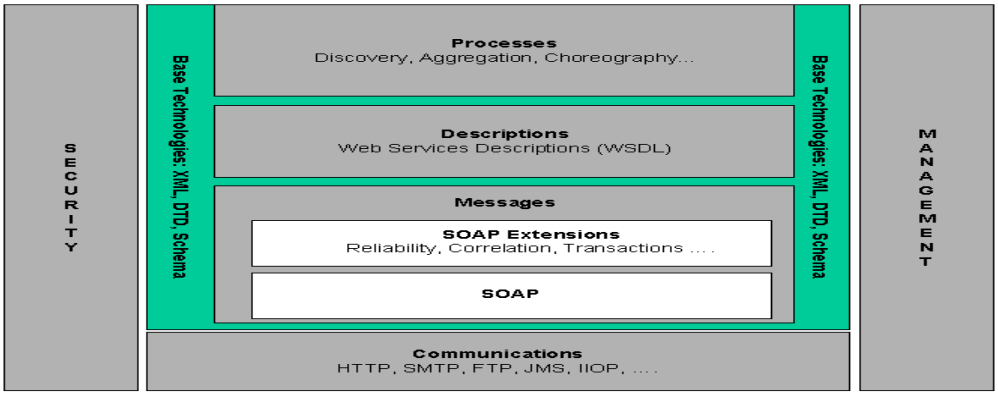

FIGURE 11: WEB SERVICES ARCHITECTURE

The “base technology” of WSA is XML. It is the essential technology for extensibility and platform-independence. While HTTP is the dominant web protocol for communication, WSA allows using a variety of other communications mechanisms like SMTP or IIOP. SOAP is an XML-based protocol for exchanging information in the decentralized and distributed environment. It defines a mechanism to pass commands and parameters between the clients and servers. Like web services as a whole, SOAP is independent of the platform, object model and programming language.

To obtain the interoperability across heterogonous systems, a metadata language is needed. Web Services Description Language (WSDL) is developed to allow the precise structure and data types of the messages being understood by the web services provider and consumer. WSDL defines XML grammar for describing contracts between a set of endpoints exchanging messages. These contracts provide documentation for distributed systems and enable automating communication in applications.

The following figure is a WSDL definition of weather service:

```
<definitions name ='weatherservice'
        xmlns='http://schemas.xmlsoap.org/wsdl/'>
        <service name='WeatherService' >  </service>
</definitions>
```

FIGURE 12: WSDL EXAMPLE

Sitting on top of WSDL, UDDI acts as the information database of web services. A UDDI registry stores the descriptions about a company and its services in a common XML format. Just as businesses list their services in the telephone directory, web services

providers publish their services in the UDDI. The data in the UDDI registries can be divided into three different section of telephone directories: a white-pages section providing service profile and contact information, a yellow-pages section categorizing the various services and a green-pages section providing additional technical information about the services.

There have been lots of efforts on realizing the web service architecture suggested in Figure 11 in the last few years. Two of the challenging tasks are to improve web services discovery and automate service execution. Many companies have announced platforms to support some level of web-service automation. Examples of such products include Hewlett-Packard's e-speak, Microsoft's .NET platform, Oracle's Dynamic Services Framework, IBM's Application Framework for E-Business and Sun's Open Network Environment (ONE).

The web services standards focus on presenting the services. However they are limited in expressing logical statements, for example, describing dependencies between the service elements. Web services promote interoperability by minimizing the requirements for shared understanding, but are still based on the assumption that the service provider and the service consumer use a common set of words. The “free” web style (anyone can publish any service in any manner) calls for the “smarter” web services – the semantic web services.

2.4.3 Semantic Web Services

The DAML Services group develops an OWL-based Web Service Ontology (OWL-S) [OWL-S 2004], as well as a set of tools to enable the automation of services on the semantic web. OWL-S aims at accomplishing the following tasks:

- Automatic service discovery: The ontology-enhanced search engine is used to locate a particular service that adheres to requested constraints;
- Automatic service invocation: OWL-S provides a set of declarative, computer interpretable APIs that are necessary for automated web service execution. A software agent is able to interpret the markup to understand what input is needed, what information will be returned, and how to execute the service automatically;
- Automatic service composition and interoperation: OWL-S provides declarative specifications of the prerequisites and consequences of individual service. A software agent can select and compose multiple web services to achieve a given task automatically;
- Automatic service execution monitoring: OWL-S aims at providing descriptors for the execution of services.

The OWL-S group publishes the OWL-S 1.0 in 2004. It includes a list of service ontologies designed to answer three questions [OWL-S 2004]:

- "What does the service do": By representing the capabilities of a service, the service profile helps the service-seeking agent determining whether the service meets its needs;
- "How does the service work": The process model describes the detailed input and output information needed when carrying out the service. The service-seeking agent

performs an in-depth analysis on the process model and decides whether the service meets its needs. The service model is also important in composing multiple services to accomplish a specific task. The composing action basically matches and connects the inputs and outputs from multiple services;

- "How to access the service": The service grounding specifies the details about how to access a service. Typically the grounding includes a communication protocol, message formats, and other service-specific details such as the socket port numbers used for contacting the service.

As WSDL emerges to be the de facto standard for service description, OWL-S is not intended to compete with WSDL. As a result, OWL-S working group has defined the OWL-S/WSDL grounding, which uses OWL classes as the abstract types of message parts declared in WSDL, and then relies on WSDL binding constructs to specify the formatting of the messages. Since WSDL is unable to express the semantics, OWL-S can co-exist and enhances WSDL, bringing intelligence into the web services.

2.5 Agent

The consumer of the semantic-encoded information provided by the semantic web must be intelligent enough to understand and utilize the knowledge. The software agents fit in this role perfectly.

2.5.1 Definition

Russell, S. and Norvig, R gives the following definition of Agent in [Russel 1995]:

[An agent is any entity that can be viewed as perceiving its environment through sensors and acting upon its environment through effectors.]

By this definition, the agent is essentially similar to the traditional AI-systems because they both interact with the environment and perform actions. The difference is that in the traditional AI-systems there is a human operator who observes the environment and describes it to the computer. An autonomous agent, on the other hand, observes the environment by itself and turns the observation into descriptions for further computations. Moreover, the agent interprets the computing results and performs the appropriate actions.

Usually, an agent has some or all of the following characteristics:

- Adaptive: It must cope appropriately and in a timely fashion with changes in the environment;
- Robust: The minor changes in the properties of the environment should not lead to total collapse of the agent's behavior;
- Tactical: It should be able to maintain multiple goals and, depending on the circumstances it finds itself in, change the particular goals it is actively pursuing;
- Versatile: it must be able to perform a large variety of tasks (in contrast to being single-purposive);
- Cooperative: it should be able to interact with other agents. The goal is achieved through cooperating or competing with other agents.

The software agents are divided into multiple categories based on how and to what degree they actually interact with the real world. In principle, there are two scenarios

possible for an agent: either the agent is alone in its environment, or there are other agents in the environment. Multi-agent systems are often assumed to be heterogeneous, i.e., the agents are of different kinds and are not specified by a single designer. An example of the multi-agent frameworks is the Trading Agent Competition [Wellman, 1999]. The trading agents cooperate in the Double auction in order to trade goods and achieve a common goal. They also compete against each other in the English auction in order to achieve the conflicting goals.

2.5.2 Agent Communication

It is often assumed that the agents are able to communicate with each other in the multi-agent scenarios. One approach for inter-agent communication, originating from the Darpa knowledge sharing effort, is Agent Communication Language (ACL) which uses the Knowledge Query and Manipulation Language (KQML) as a high-level language and protocol, and uses the Knowledge Interchange Format (KIF) to specify content.

KQML [Finin 1992] is a widely used language for exchanging information and knowledge. It supports a set of performatives, which define the permissible communicative operations that the agents may perform on each other such as: informing, asking questions and commanding actions. A KQML expression consists of three layers: (i) A content expression (ii) A message wrapper that specifies the performative (iii) A communication wrapper that specifies communication parameters such as sender and receiver. The content can be expressed in any representation language that follows some general syntactic constraints. KQML can be used for an application program to interact

with an intelligent system or for two or more intelligent systems to share knowledge in support of cooperative problem solving.

Based on KQML, the FIPA-ACL specifies a standard message language by setting out the encoding, semantics and pragmatics of the messages. The syntax of the FIPA-ACL is very close to KQML. However, it differs from KQML fundamentally, as there is a formal semantics in the FIPA-ACL eliminating any ambiguity and confusion from the usage of the language. The FIPA-ACL standard supports common forms of inter-agent conversations through the specification of interaction protocols, which are patterns of messages exchanged by two or more agents. Such protocols range from simple query-request protocols, to the well-known contract net negotiation protocol, English and Dutch auctions.

2.5.3 Agent Standard Organizations

Agent-based technologies can realize their full potential only when adequate standards supporting agent interoperability become available and widely adopted by the agent developers and agent systems. The Foundation for Intelligent Physical Agents (FIPA) is an international non-profit association of companies and organizations who share the effort to produce specifications of generic agent technologies allowing an easy interoperability between the agent systems. The FIPA specifications specify the normative rules that allow a society of agents to operate, inter-operate, and be managed. First of all they describe the reference model of an agent platform. Three key agents are identified: Agent Management System (AMS), Agent Communication Channel (ACC)

and Directory Facilitator (DF). AMS exerts supervisory control over access to and use of the platform; it is responsible for authentication of resident agents and control of registrations. ACC provides the path for basic contact between agents inside and outside the platform; it is the default communication method that offers a reliable, orderly and accurate message routing service. DF provides a yellow-pages-like service to the agent platform. The FIPA specifications also specify the Agent Communication Language (ACL). The agent communicates by formulating and sending individual ACL messages to each other.

Another standard organization is the Open Agent Architecture (OAA). It focuses on building distributed communities of agents, where agent is defined as any software process that meets the OAA society conventions. An agent satisfies this requirement by registering the services it provides in an acceptable form, by being able to speak the Inter-agent Communication Language (ICL), and by sharing functionality common to all OAA agents, such as the ability to install triggers and manage data in a certain way.

2.5.4 April and AAP

The April Agent Platform (AAP) is a FIPA 2000-compliant agent platform written in April language. The Agent PRocess Interaction Language (April) is developed by Francis G. McCabe and Keith L. Clark at Imperial College, London. It is a symbolic programming language for building distributed applications, in particular, multi-agent systems.

There are a number of novel features in the April language:

- Object-based concurrent language: Using active objects as processes to send messages and invoke the methods, the April language is a highly suitable base for extending to the multi-agent application platforms;
- Processes facility: April allows processes to communicate with each other and has powerful data structuring and expression;
- Type system: In addition to the traditional roles, the types in April form the basis of pattern matching on messages and set-style search operations on a list of structured values;
- Higher order: April allows both programs and codes to be passed from one process to another. The higher order is also used for implementing the static objects.

InterAgent Communications Model (ICM) is a distributed message passing system for the April programs. It controls the messages passing: encoding, decoding, delivering and routing.

Built on a principle of extensional modularity, AAP provides the basic environment where a FIPA agent can be launched and operate. AAP supports yellow pages like directory services DF, white page likes name resolution services AMS and a flexible Message Transport System. Services in AAP are abstracted through an agent wrapper that provides easier integration with the agent systems.

The AAP system was used in the Evening Organizer application for the San Francisco Agentcity site. It is also used in TAGA system as the game server platform.

2.5.5 Jade

Java Agent DEvelopment framework (Jade) [Bellifemine 1999] is a software framework to facilitate the development of agent applications. Complying with FIPA specifications, Jade runs all those mandatory agents that manage the platform, including AMS, DF and ACC. The agent communication is performed through message passing, where FIPA-ACL is the language representing messages.

The goal of Jade is to simplify the agent development while ensuring standard compliance through a comprehensive set of system services and agents. To achieve this goal, Jade offers the following features to the agent programmers:

- Distributed agent platform: The agent platform can be split into several hosts. Agents are implemented as one Java thread. The Java events are used for effective and light-weight communication between agents on the same host;
- Federated DF: Multiple DF can be started at run-time to support the applications crossing multiple domains;
- Light weight transportation: The ACL messages, whose sender and receiver are inside the same agent platform, are encoded as Java objects to avoid a marshalling procedure. Otherwise, the messages are automatically converted to and from the FIPA compliant string format;

- GUI interface: Jade implements a Remote Monitoring Agent (RMA) to provide a Graphical User Interface (GUI) for the remote management, monitoring and controlling of the agent's status.

TAGA uses Jade as the game clients, taking advantage of its GUI feature.

2.6 Agentcities

In recent years, a significant amount of research effort has been directed towards enabling vastly richer and more dynamic interaction between publicly accessible online systems. One of the efforts is Agentcities [Willmott, Nov 2001] [Dale, 2002], an open network of agent platforms designed to help realize the commercial and research potential of agent-based applications. By constructing this worldwide platform hosting diverse agent-based services, Agentcities expect to achieve the goal of enabling the dynamic, intelligent and autonomous composition of services, thereby creating compound services to achieve user and business tasks and address changing needs.

The design of Agentcities is based on three principles [Willmott Nov 2001]:

- The agent and multi-agent systems technology is adopted as a powerful metaphor for defining and describing advanced service environments. One of the long-term goals is to combine technologies such as web services, semantic web and agents into a single coherent model;

- The development of a global test network serves as a real deployment environment for the testing and integration of innovative services, applications and technologies;
- The network is open to anybody wishing to connect to it.

The nodes in the Agentcities network are the agent platforms running on one or more machines hosted by an organization or individual. One agent running on a particular Agentcity node is able to communicate directly with the agents running on other publicly available Agentcity nodes. Applications involving the agents running on multiple different Agentcity nodes can be created through the flexible use of inter-agent communication models, semantic frameworks and shared ontologies. This communication model consists of the following three levels:

- Network level: Platforms in the Agentcities network interoperate and exchange basic communications at the communication and infrastructure level;
- Service composition level: The business components including the services and behavior descriptions are supported in this level;
- Semantic interoperability level: This level turns Agentcities into a test bed for system-system communications in an open environment. The business components are hosted dynamically without requiring human intervention to perform service discovery and invocation.

As of today, Agentcities are largely experimental. There are several workgroups inside Agentcities, such as ontology, services and language. These groups focus on difficult tasks, with the shared goal of developing an open service platforms.

2.7 Trading Agent System

Many trading systems were developed to simulate the real-world trading scenarios and experience the innovative trading theory and algorithms. This section starts with the auction, one of the most widely used trading methods. Then, a list of trading agent systems is discussed.

2.7.1 Auction

Auction, with root "auctio" means increase, is a competition-based method of allocating scarce resources. There are three participants in an auction. A seller, who owns the resources and wishes to obtain as much money as possible, holds an auction to extract information he might not otherwise realize. An auctioneer acts as the agent for the seller. A buyer, who frequently knows more than the seller about the value of the resources, wants to pay as little as necessary. An auction offers the advantage of simplicity in determining market-based prices. It is efficient as an auction usually ensures that the resources accrue to those who value them most highly and also ensures that the sellers receive the collective assessment of the value.

William Vickrey, winner of the 1996 Nobel Prize in Economic Sciences, establishes the basic taxonomy of auctions based upon the order in which prices are quoted and the

manner in which bids are tendered. He suggests the following four major one-sided auction types:

- The English auction, also known as the open-outcry auction or the ascending-price auction, is the auction type most people are familiar with. Paul Milgrom defines the English auction as [Milgrom 1987]: "*Here the auctioneer begins with the lowest acceptable price--the reserve price-- and proceeds to solicit successively higher bids from the customers until no one will increase the bid. The item is 'knocked down' (sold) to the highest bidder.*" ;
- The Dutch auction, also known as the descending-price auction, uses the open format like the English auction. It is the technique used in Netherlands to auction produce and flowers (hence, a "Dutch" auction). In a Dutch auction, bidding starts at an extremely high price and is progressively lowered until a buyer claims an item by submitting a bid;
- The first-price auction, as suggested by its name, awards the auction item to the highest bidder at the bidding price. Each bidder submits one bid in ignorance of all other bids. It has a primary characteristic of being sealed (not open-outcry like the English auction or Dutch auction) and thus hidden from other bidders. A sealed-auction has two distinct steps: a bidding phase in which the bidders submit their bids, and a resolution phase in which the bids are opened and the winner is determined;
- The Vickrey auction, also known as the uniform second-price auction, is named after William Vickrey. Like the first-price auction, the bids are sealed. Each bidder is ignorant of other bids. The item is awarded to the highest bidder at a price equal to the second-highest bid (or highest unsuccessful bid).

In addition to the four one-sided auction types, there is double-sided auction in which both buyer and seller suggest prices. The prices are ranked from highest to lowest to generate demand and supply profiles. From the profiles, the maximum quantity exchanged can be determined by matching selling offers (starting with lowest price and moving up) with demand bids (starting with highest price and moving down). This format allows buyers to make offers and sellers to accept those offers at any particular moment. A continuous double auction is one variant in which many individual transactions are carried on at any single moment and trading does not stop as each auction is concluded. The New York Stock Exchange is an example of the continuous double auction.

The following table is the comparison of the five different auction types.

Name	**Type**	**Feature**	**Rules**
English	Open	Winner's Curse	Seller announces reserve price or some low opening bid. Bidding increases progressively until demand falls. Winning bidder pays highest valuation.
Dutch	Open	One buying bid	Seller announces very high opening bid. Bid is lowered progressively until demand rises to match supply.
First-price	Sealed	Winners' curse	Buyer submits one bid with no knowledge of bids of others. Winner pays the exact amount he bids.
Vickrey	Sealed	Less winner's curse	Buyer submits one bid with no knowledge of bids of others. Winner pays the second-highest amount bid.
Double Auction	Open	Non-Stop	Both buyer and seller submit bids. Item is sold by matching selling offers with demand bids.

FIGURE 13: COMPARISON OF FIVE AUCTION TYPES

Winner's curse refers to the fact that the winner of the auction pays the highest price for the auction item. From Figure 13, we learn that winner's curse is high in the English auction and the first-price auction. The Vickrey auction can decrease the winner's curse by allowing the winner pays the highest losing bid.

2.7.2 Trading Agent Competition Classic

The Trading Agent Competition (TAC) [Wellman, 1999] is a market simulation game initiated at the University of Michigan's Artificial Intelligence Laboratory in 2000. The first Trading Agent Competition was held in Boston in conjunction with ICMAS conference. It is based a travel business scenario. The travel agents, servicing multiple customers, compete to make travel arrangements to a conference. They must book flights, hotels, and any special events (concerts, etc.) that each customer wants to attend, taking into account such constraints as customer travel preferences, dates, flight availability, pricing, etc. The second and third TACs, which were held in the subsequent years, maintained research issues in simultaneous interrelated auction context, and had minor modifications for further research. The fourth competition initiated new research issues in supply chain management context and kept the original TAC framework under the name "TAC Classic". The TAC Classic was designed to promote and encourage research in markets involving auction and autonomous trading agents and has proven to be successful after four consecutive year's competitions.

The TAC Classic focuses on automated strategies for the software trading agent. A trading agent assembles a round-trip travel package for each of its customers by trading

goods in multiple concurrent and interrelated auctions. The TAC Classic runs in client/server mode: the game server generates eight customers for each trading agent, and runs twenty-eight simultaneous auction instances which supply travel goods. On the client side, a participating trading agent acts on behalf of its customers, ordering airline tickets, hotel rooms and buying/selling entertainment tickets. The payout comes from a known utility function. This game is highlighted by its three types of auction mechanisms: eight continuous one-sided auctions on airline ticket (supply is unlimited during the game, and prices tend to rise over time); eight standard English ascending multi-unit auctions on hotel room (with auctions closing in random order); twelve standard continuous double auctions on entertainment ticket (both buying and selling are allowed during the game). The trading agent needs to allocate and buy its customers' travel packages within a limited time. The performance of a trading agent is evaluated by the profit obtained according to the utility function.

2.7.3 TAC Supply Chain Management

In today's global economy, effective supply chain management is vital to the competitiveness of manufacturing enterprises as it directly impacts their ability to meet changing market demands in a timely and cost-effective manner. With annual worldwide supply chain transactions in the trillions of dollars, the potential impact of performance improvements is tremendous.

To capture the complexities of actual supply chains, a team of researchers from the e-Supply Chain Management Lab at Carnegie Mellon University and the Swedish Institute

of Computer Science (SICS) designed and implemented the TAC Supply Chain Management (SCM) game in 2003. The first competition was hold in conjunction with IJCAI 2003. Simulating a dynamic supply chain environment where multiple companies compete to secure customer orders, TAC SCM targets the problems of planning and coordinating of the organization's activities across the supply chain, from raw material procurement to finished goods delivery.

In each TAC SCM game round, six personal computer (PC) assembly agents (game participant) compete for customer orders and for procurement of a variety of components over a period of several months. Each day customers issue requests for quotes and select from quotes submitted by the agents, based on delivery dates and prices. The agents are limited by the capacity of their assembly lines and have to procure components from a set of eight suppliers. Four types of components are represented in the game: CPU, Motherboard, Memory, and Hard Drive. It features a variety of components of each type. For example, CPU and Motherboard are available in two different product families, Pintel and IMD. A Pintel CPU only works with a Pintel Motherboard while an IMD CPU can be incorporated only in an IMD Motherboard. There are a total of 10 different components, which can be combined into 16 different PC configurations. Customer demand comes in the form of requests for quotes for different types of PCs, each requiring a different combination of components.

The PC assembly agents are responsible for negotiating supply contracts, bidding for customer orders and managing daily assembly activities. The most challenging problem

for those agents is to concurrently compete in multiple markets with interdependencies and incomplete information, where both demand and supplies are subject to fluctuations, and where each manufacturer is limited by the capacity of its factories. To succeed, the agents will have to demonstrate their ability to react to variations in customer demand and availability of supplies, as well as adapt to the strategies adopted by other competing agents. With TAC SCM, researchers hope to demonstrate that Internet-based software agents can enable companies to be more flexible in component acquisition, and thus able to better manage inventory, schedule manufacturing, fulfill orders, control costs, and ultimately increase profitability.

2.7.4 FAucS

In recent years, governments around the world are using spectrum auctions as a mechanism for allocating rights to the radio spectrum. In addition to the complexity of these auctions, they have huge commercial importance, each bringing in billions of dollars to governments around the world. FAucS [Csirik 2001] is an FCC Spectrum auction simulator developed by AT&T Lab. It is a software test-bed for studying automated agent bidding strategies in the simulated auctions, specifically the United States FCC wireless frequency spectrum auctions.

The goods available in the FCC spectrum auctions are a set of licenses or blocks of spectrums in a region of the United States. All goods are available at the same time, and bidding occurs in discrete rounds in one FCC spectrum auction. After each round, each bidder's bids are announced publicly. The provisionally winning bids, the highest bid

received up to that point on each license, are also announced. The auction ends immediately after the first round with no new activity. Each license is then sold to its provisional winner, for a price equal to the provisionally winning bid.

The most important rules that make FCC spectrum auctions different from other auctions are activity constraints. In an English auction, the bidders may sit out during the early rounds of bidding and delay placing their serious bids until late in the auction. Such a strategy would result in lower FCC revenues and an unnecessarily long auction. A set of rules is designed to prevent the bidders from employing this strategy. For each round of the auction, each bidder has a certain eligibility and activity, both of which are expressed in terms of bidding units. The bidder is allowed to submit a set of bids that would make its activity higher than its eligibility. Additionally, each bidder is allowed up to five activity waivers, intended to ensure that equipment failures do not put bidders in impossible situations.

The FAucS uses the client/server architecture. The server and the bidding agents use TCP sockets to communicate with each other. The FAucS also allows humans and programs to simultaneously participate in a simulated auction.

2.7.5 The Penn-Lehman Automated Trading Project

The Penn-Lehman Automated Trading (PLAT) Project [Kearns 2003] broadly investigates the algorithms and strategies for automated trading in the financial markets. The centerpiece of the project is the Penn Exchange Simulator (PXS), an automated stock

trading agent that merges automated client limit orders with real-world, real-time order data available via modern Electronic Crossing Networks (ECN). In this environment, multi-client trading simulations match client orders both with each other and with ECN orders, thus effectively blending the internal and external markets. PXS automatically computes the profits and losses of clients, volumes traded, and various other quantities of interest.

The core part of the PXS is the execution engine, which is responsible for maintaining the PXS order books, integrating Island client limit orders into the PXS order books, executing matching orders of PXS clients and computing the share position, cash holdings, profit and loss of PXS clients. The execution engine consists of a number of main steps that are executed repeatedly throughout a simulation. First, the island updates are retrieved and sent to the PXS books. Then, the PXS executes the orders from the island available volume and the PXS books. At the end, the PXS client orders are processed and a new PXS public view is generated.

The PXS can be used as the test-bed for developing automated trading strategies. The real-data, real-time nature of the PXS makes it possible to examine the trading strategies in the intensive, high frequency and high volume market.

CHAPTER 3 AGENT-BASED SERVICES

The goal of this chapter is to specify what the agent's role is in the future service-based web, which is powered by the semantic web technologies, and how the agents and semantic web should and can work together. These discussions will be motivated by the potential problems in a set of user cases.

3.1 Agent's Role on the Semantic Web

Human agents, such as realtors or travel agents, bring together two parties (buyer and seller). Obviously, a single person cannot be an expert in every area. So, people are willing to pay for someone professional and experienced to do the jobs like buying a house. The reason why human agents exist is that they have specified domain knowledge and have the ability to do repetitive tasks.

Similar to the human agents, the software agents also act on behalf of the entity (human user in most cases) and perform the tasks requiring specified domain knowledge. While the software agents existed long before the semantic web idea came up, the following characteristics owned by the software agents make them fit perfectly on the semantic web:

- Autonomous: An agent has authority to autonomously act on behalf of the user and without continuous guidance from the user. In addition to accomplishing assigned

tasks, as a web service program usually does, the agent is able to activate action and create new tasks. The web is a massive information collection. People don't have the opportunity or desire to review all the details. The ideal model for a human is first saying: "I want an air ticket to Boston next Monday", then making some selections and approving the purchase of the travel package. The agent fills the gap by collecting information, extracting knowledge, organizing, planning and committing the tasks based on rules of behaviors. Additionally, the agent is able to react faster than the human. This is useful in case like rescheduling after being notified by the airline agent that the plane being canceled;

- Cooperative: In addition to communicating with the environment to collect data and take action, an agent has the ability to communicate, cooperate and negotiate with other agents to solve problems that are beyond its capacities and knowledge. On the semantic web, a task is usually composed of a set of small tasks accomplished by the service programs spatially distributed around the Internet. The service requester must be smart enough to recognize opportunities for action/inaction, infer new knowledge, and know where and how to look for services. This is possible only with the agent. Given the ability to communicate with other agents, the agent can automatically accomplish extremely intense tasks for the user;
- Adaptive: An agent has the ability to adapt behavior to optimize performance. The web is an extreme unstable environment. New content is added every second. The web server or web pages may be pulled off or changed in any minutes. The agent with adaptive mechanisms (self-monitoring and online learning) ensures robustness and may ultimately accomplish the tasks;

- Social: An agent supports the policies to constrain the agent behaviors and can manage conflicting obligations. The confliction and contradiction on the free-styled web, where anybody can say anything, is accepted and can be resolved by the agent.

Meaningful interaction between the agents requires semantics. The ontology provides a framework for shared definitions of terms, resources and relationships among them. They also provide the models for the agent action and interaction. This makes the ontology-enriched semantic web an ideal platform for the agent technology.

On the semantic web, a human is linked with a personal agent. So he can assign tasks like "buy an air ticket to Boston next Monday" to his personal agent. This sample task requires knowledge of personal schedule (locate where the human is flying from), travel business (check and compare the air ticket prices from multiple travel agents and choose the cheapest one) and personal finance record (paying with credit card). The personal agent may ask the human to provide additional information, choose among multiple options or confirm the transaction. The link between the human and the personal agent can be as simple as a few form-based web pages or as complex as an interface agent.

Besides the personal agent, the agents also serve the following roles on the semantic web:

- Interaction controller: The software agents commit and monitor the interaction in the system. The interaction includes conversation, negotiation, enactment and monitoring;Communication controller: The software agents manage and enrich the communication on the semantic web using the communication act and agent communication language. Compared with the client/server model in the traditional

web, the peer-to-peer model used by the agents fits better with the future service-based web;Service finder: The software agents help locating the right service provider in the massive Internet and web space. This involves discovering the service provider, matching service request with the service description and selecting the most suitable service provider; Organizer: The software agents are able to accomplish complex tasks by utilizing the pre- and post- conditions defined in the service description, recognizing the input and output properties and automatically composing a list of small service together.

As discussed in Chapter 2, the agents are divided into many categories and have many international organizations working on their standards. Among them, FIPA publishes well-developed standards and many widely used FIPA-compliant multi-agent systems. Most important, RDF is already one of FIPA's standard content languages and OWL is widely used within the FIPA community, though not yet formally adopted as a FIPA compliant content language. FIPA system is a good option when choosing the agent system for the semantic web environment. In this thesis, we use FIPA-compliant systems as the agent platforms.

3.2 Problem Statement

Since the agents play important roles on the semantic web, this chapter starts with running multiple personal agents in a semantic web environment (Figure 14) and suggests the following user cases:

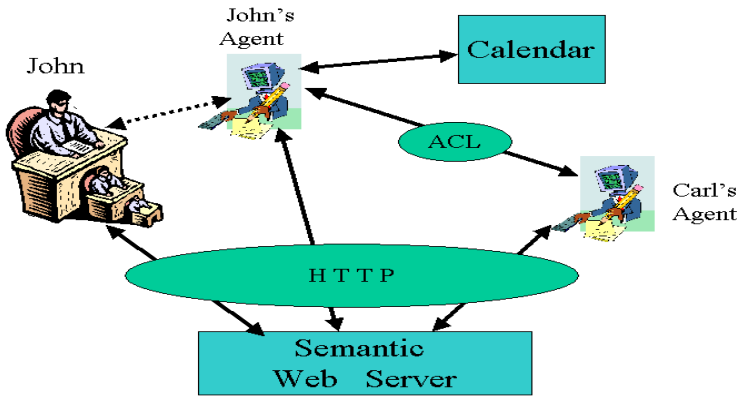

FIGURE 14: SEMANTIC WEB USE CASE

Case 1:

User John visits the xTalk web site via his web browser and determines that this web site is useful. He instructs his personal agent to monitor xTalk web site and inform him whenever there is a talk matching his research interests. As a result, John's agent is informed when a new talk is added to xTalk web site. The agent retrieves the talk announcing OWL page, parses it into triples, and inferences those triples with rules. One of the rules is to match John's research interests with the talk's topic keywords. Because the talk is written in OWL language, the agent can easily get the talk topic keywords from a talk ontology instance. John's interests are available at John's homepage (also written in OWL). When the research interests and topic keywords match, the agent sends an email to John informing him of the new talk and updates the schedule.

Case 2:

John adds a rule specifying that he is interested only in the talks that match his research interests and his busy schedule. The personal agent checks his calendar (Microsoft

Outlook) and informs him only when the talk's topic keywords match his research interests and he is available at the time of the talk.

Case 3:

John wants his agent to be a talk-announce filter and inform him only when the talk is really important to him. So, he assigns the following rules to his agent: John is informed about the talk when the talk's topic keywords match his research interests and his advisor Dr.Carl recommends the talk. John's agent retrieves the research interests from John's homepage and matches them against topic keywords from the talk announcing OWL page. Receiving the query message from John's agent, Carl's agent checks the talk announcing page and generates a score based on the Carl's agent's preference, policies and knowledge. The score is sent back to John's agent. John's agent then decides whether to inform John.

Problems when the agents meet the semantic web

The human browses a web page and understands what it is talking about easily and naturally because the human has a thinking brain. How can John's agent understand the talk announcing web page?

The knowledge in a semantic web page is represented as ontology using the semantic web language like OWL. The agent downloads the web pages, learns the ontology and later inferences about the knowledge. It is clear that learning the ontology is an important step toward understanding the web page. XML, the data exchange language, is based on

the assumption that both parties of the communication use the same DTD or Schema file. The understanding of XML content is achieved by matching string and keyword. Similarly, the agents use a common ontology to define the vocabulary with which queries and assertions are exchanged. However on the semantic web, a common ontology may not exist, because web sites owned by different organizations or people geography distributed around the world are very unlikely to agree on the same vocabulary or knowledge. Even using the same vocabulary, web pages with different context may have different meaning. Pragmatically, the agent-understandable web page based on the agent's ontological commitments is not a good model for the open web environment.

Once having the right semantic web pages, the next step for the agents is to retrieve information from the pages. Various query languages and query systems can be used to retrieve the information from the web pages. However, in some case, the needed information may not be available on the web page. For example in user case 3, John may have a rule: "if more than 20 people join the talk, I will join the talk". John's agent asks xTalk web site:" how many people will join this talk?". The xTalk web site does have this number in its database, but does not have an appropriate web page to publish it.

One solution for this problem is having the web server open all information to other agents. The personal agent is able to query the web server's backend database directly and get whatever it wants. But a new problem arises, as the personal agent needs prior knowledge about the web server's database design. It is also inefficient to force every personal agent to carry all the responsibility to learn all knowledge and understand all

rules and policies at various web sites. Furthermore, this solution brings a security problem since the rules and policies of one website are often considered critical information.

In the above user cases, John's agent monitors the xTalk web site and checks whether there is a new talk being added from time to time. It is a time-consuming job and brings the extra burden to both the personal agent and the xTalk web server. An alternative solution is that the xTalk web site informs the personal agent automatically when there is a new talk added. Subscribed to the xTalk web server's new talk-announcing service, John's personal agent gets notified with the addition of every new talk. This kind of subscribe/broadcast model is common in the agent society, however, not supported by the web.

In user case 3, Dr Carl's agent computes a recommend score for the talk. The score is Dr. Carl's comment to the talk. It would be helpful if the comments and assertions about the talk can be stored somewhere, and other agents can get this score when they visited the talk announcing OWL page. The current web services and agent model do not support this operation.

Problem when agent meet agent

FIPA has already addressed many problems about agent interoperation, including Agent Communication Language and Communication Protocol. FIPA assumes that the agent has prior knowledge about the communication methods and message content. The FIPA

agents are communicating with pre-defined specific terms. Specifically, the referencing of multiple ontologies within the message content is not currently supported by FIPA. However one challenge in the distributed web is that every user and web site might develop and use its own ontologies. The agents running on the semantic web have to deal with multiple ontologies. It is also desirable to make the agents interoperable on the semantic web, which needs supporting of ontology mapping and ontology sharing. Most research in ontology mapping has assumed that mappings are developed manually and focuses on how to represent them. But the agents on the web have no prior knowledge about what ontologies others agents might use when communicating or negotiating with them.

Problem when agents meet web services

In the above user case 2, John's agent may need to compute the driving time when computing availability. This, of course, can use a web services providing a Mapping web site. However the current web services and agents are two separate fields and do not interact with each other well. They also fundamentally differ in the service description, service publishing, service matching and service model. For example, the agents use DF to register and locate the service, while the web services use UDDI. The consequence is that they can't find and interact with each other. It is possible to write a special wrapper program for a specific purpose. But as the web services and the agents want to work together as a whole, a middleware component between them is needed.

3.3 Related Works

The problems suggested in Chapter 3.2 are related to three research fields: semantic web, software agents and web services. The rest of this section lists a number of previous and ongoing works targeting those problems.

Retsina Calendar agent [Payne 2002]

Retsina CALendar agent (RCAL), a project from Carnegie Mellon University, provides interoperability between RDF-based calendar descriptions on the web and Personal Information Manager (PIM) systems.

The Retsina agent consists of two parts: the Retsina semantic web Calendar Parser and the Meeting Scheduling Engine. The Calendar Parser allows the user to browse the schedules and events (for example a conference), and mark up them in RDF with respect to the existing event ontology (iCal). It also allows inserting the schedules and contact details found on the semantic web into the PIM (for example Microsoft Outlook). The Meeting Scheduling Engine assists in organizing and scheduling meetings between several individuals, and coordinates these meetings based on existing schedules maintained by PIM. Calendar Agents automatically obtain new schedules and negotiate possible meeting times with other Retsina agents based on the user's schedule and preferences.

RCAL is an early exploring project to test-drive the semantic web with a commercial product (Microsoft Outlook) and real data (schedule of first semantic web Working

Symposium held at Stanford in 2001). Although it uses the primitive semantic web language (RDF) and does not have much work on ontology, it does demonstrate the attractive features and future of the semantic web.

Onto-Agent Project [Decker 1999]

Onto-Agent focuses on developing a set of tools for ontology management on the semantic web. The ontology defines the terms that can be used to annotate information in the web pages. It provides the OnTo-Agents Ontology Construction Tool to build the ontology and the OnTo-Agents Web page Annotation Tool to select appropriate mark-up terms of the ontology map. The web page annotation process creates a set of annotated web pages, which are available for an OnTo-Agent to achieve its tasks. The OnTo-Agent includes the OnTo-Agents Inference System that supports the evaluation of rules, queries and general inferences. The data from the annotations can be used to construct additional websites: a Community Web Portal that presents a community of interest to the outside world in a concise manner. Personal agents can give specific retrieval tasks to an OnTo-Agent, or they can query a Community Web Portal for immediate access to the information.

ConGolog [Giacomo 2000]

ConGolog, a project from Stanford KSL lab, is a DAML-enabled agent that automates web service discovery, execution, composition and interoperation. The Agent is implemented using the first-order language. It automatically performs the complex tasks such as travel planning by exploiting DAML markup of web services and DAML markup of user constraints. ConGolog assumes that the semantic web activities are constructed as

reusable and high-level generic procedures. These activities are represented as distinguished services in DAML language and archived in sharable generic procedures. A user can then select a task-specific generic procedure from the ontology and submit it to the DAML-enabled agent. Considering the user's DAML-encoded constraints, environment and available services, the agent automatically performs the task by customizing the procedure and executing a sequence of requests to web services.

ConGolog demonstrates a DAML-enabled agent working in a web services environment. The first-order language used in ConGoLog is able to support complex customization and composition tasks. But, without the support of multiple ontologies and the multi-agent systems, there is no communication or cooperation in this system.

Scientific American Scenario

Recently Dr. James Hendler published the implementation comments on the "Lucy and Pete dealing with Mom" scenario suggested in the pioneering semantic web article [Berners-Lee 2001]. In the scenario, Lucy and Pete search a doctor for their mom based on the available time and office location, and make an appointment that fits both of their busy schedules.

The following is a list of components to be implemented:

- Service description and discovery: The web services are described in the OWL-S language and made available for discovery using Universal Plug and Play technology;

- Service agents: The service agent is represented as a web services program based on action;
- Service composition: A service composition tool composes OWL-S descriptions and executes them using the WSDL and UPnP groundings;
- Planning: The web service composition task is mapped to a planning problem. Then, the Simple Hierarchical Ordered Planner is used to arrange a meeting time for multiple persons according to their schedules, also to find the correct order of appropriate services in order to accomplish the task;
- Personal agent: The FIPA complaint Jade system is used as personal agent. RDF is used as the content language;
- Knowledge management: ParkaSW system is used to store semantic data and provide basic query capabilities;
- Trust: A rule-based authentication is applied to build authentication of each agent. Additionally, a distributed trust system using social network analysis is applied to build trust relationship among agents.

The “Lucy and Pete dealing with Mom” scenario suggests how the semantic web might affect our real life. The implementation of this scenario is a challenging job as it covers almost the entire “Semantic Web Vision Cake”.

Agent-based Semantic Web Services [Gibbins 2003]

Inspired by the multi-agent systems techniques of separating the intentional force of messages from the content, this University of Southampton project suggests using this technique within the DAML Services model.

The technique of factoring out the common attributes of message types and ascribing them to different classes of speech acts is commonly used in the design of agent communication languages (ACL) for multi-agent systems, where there is a clear separation made between the domain-specific and domain-independent communications. This separation between intent and domain is beneficial because it reduces the brittleness of a system. If the characterization of the application domain (the ontology) changes, then only the component that deals with the domain-specific information needs to be changed; the agent communication language component remains unchanged. However, the communicative intent of a message (for example, whether it is a request or an assertion) is not separated from the application domain in the conventional web services approach or OWL-S,

To solve the problem, a simple process ontology of message types based on the FIPA ACL is designed. In this ontology, ACL message types are represented as atomic processes, with the content of the message as a parameter of the process. As a proof of concept, a situational awareness system in a humanitarian relief scenario is implemented to demonstrate the use of this ontology.

We agree with the opinion suggested in this project that the ACL component can be integrated into the semantically rich service descriptions and used in the procedure-calling web services environment. However, we argue here that agent itself is a "player" in the semantic web services. ACL may be useful in Java method invocation (as in this project), but ACL is intended for agent communication and can be used to improve the service interaction. A straightforward solution would be introducing a semantic-rich language into ACL.

GraniteNights [Grimnes 2003]

GraniteNights is the work by the University of Aberdeen as part of the Agentcities project. It allows a user to plan an evening's entertainment in the city of Aberdeen, Scotland. The technologies involved in this system include using RDF as agent communication content language, using DAML+OIL as ontology language and using Jade as the agent system.

Agents are organized according to a series of predefined roles:

- Information agent: It acts as wrappers for RDF data resources. Each has a simple, consistent query interface based upon Query By Example language;
- Profile agent: It manages user data, such as ID, password and preferences;
- Constraint-Solver agent: As the name suggests, it maps RDF data to finite domain constraints and produces valid instantiated schedules;

- Evening agent: This is the central agent that coordinates the process of delivering the evening plan. It receives user queries and controls invocation of other agents to generate a solution;
- User interface agent: It connects the web interface and provides an easy way to control the system.

The GraniteNights project makes no attempt to deploy a sophisticated scheduler or profiling mechanism. The Evening Agent uses a pre-defined collaborative strategy. Additionally, this project makes no attempt to discover the service. The addresses of three information agents are foreknown to every agent in the system.

One of the key opinions in this project is that RDF can be a good alternative content language for the agent interaction. We agree with this opinion. Although FIPA-SL may be more expressive, the fact that RDF is based on XML makes RDF a better choice as the content language for the agent-based web services. However, we argue here that the current expressiveness of RDF is not enough for many applications. OWL is rising as the standard semantic web language, and OWL-S may become the choice for service description language. Apparently, OWL fits well in the agent-based web services as the content language.

Semantic Web Services as Agent Behaviors [Buhler 2003]

Paul A. Buhler in the University of South Carolina explores the technique for providing agent software with dynamically configured capabilities. He suggests that the semantic

web and web services can facilitate agent-based workflow management in an open environment.

The experimental system uses a composition language named Piccola and utilizes DAML-S to describe agent's capabilities. The composite service is available as a semantically-described behavior within a FIPA-compliant agent. The workflow is achieved with the cooperation and coordination of a list of sociable agents representing individual services. An individual agent achieves coherent behavior globally through its coordinated interactions with other agents. Together as a system, the agents have the capability to dynamically form social structures through which they share commitments to the common goal of workflow enactment.

3.4 Agent–Based Services for the Semantic Web

This section will discuss our version of running agent in the semantic web environment. We start with the definition of service agent concept. Then we propose a number of novel solutions to resolve the problems listed in Section 3.2: use Semantic Web language as Agent communication language; support multiple query languages; design a set of ontology actions to support multiple ontologies.

3.4.1 The Service Agent

We divide agents into two categories: personal agent and service agent. The function of the personal agent is to assist the human user, such as maintaining a personal calendar, finding the best price or booking an air ticket. The function of the service agent is to

provide specified services to other agents, such as announcing a talk or selling the airline ticket.

The personal agent communicates in Agent Communication Language and cooperates with other agents to finish tasks based on predefined rules or policies. It is capable of learning and inserting new facts into its knowledge base. The personal agent is a "special" service agent in the sense that it provides service to only one customer: its human owner. The research of personal agent is primarily focused on human interface and intelligence.

The service agent publishes "white page" services description content and registers its services at a "yellow page" site so that other agents can search and find it. It is capable of understanding an ontology and answering queries related to it. A semantic web server is linked with a service agent and pushes information to other agents through it.

The web services research activities like WSDL also suggest providing services to programs or agents. The web server publishes WSDL services description pages and registers the services in an UDDI directory. However without reasoning ability, the web service description only indicates a machine's ability to solve a known problem by performing pre-defined operations on existing well-defined data. The most significant difference between a web services program and a service agent is that the latter is capable of initiating actions and constructing plans of action on its own. To solve a given task, the

service agent may contact other semantic web sites and cooperate with other agents to learn an unknown ontology or agent interaction protocol.

The personal agent can learn new knowledge from the semantic web pages; however this learning may be insufficient for the personal agent to make a decision. The personal agent may have additional questions about the ontologies and knowledge on the web pages. A dedicated service agent with reasoning ability, being part of the services web site, can answer questions about a semantic web page because it knows more about the web site and has better understanding of the knowledge than any outside agents. The service agent also supports accepting comments and assertions about the web pages, and publishes those comments as a semantic web page.

An obvious question with this service agent model is " how does the personal agent find whom to ask?" A human may discover a new website by clicking a link on the Google search results, reading the advertisements in the newspaper or being told by a friend. Corresponding to the human models, there are three models of dealing with finding the right service agent.

The first model is that there is a Google-like central service agent. The central service agent is responsible for all knowledge in all web pages. Also, the service agent may register and send information to the central service agent. The central service agent crawls through the Internet, loads all known facts and rules into the knowledge base and inferences about them later. The personal agent communicates with the central service

agent and asks the questions. Both AIFB's onto-server [Motik 2002] and SHOE system [Heflin 2000] belong to this model.

The second model is that the personal agent goes to search the web site and read web pages on its own (like the human reading a newspaper model). Seeing a useful page, the personal agent loads the page, extracts the desired knowledge out from it, and adds the knowledge into its own knowledge base. The agent accomplishes the tasks of retrieving information and managing knowledge. Like the human finding an advertisement in the newspapers, the personal agent can also learn from the service "advertisement" sent by the service agents. The web crawling program and shop-bot [Greenwald 1999] belong to this model.

A problem with those two models is that one agent has to visit all the web pages and reason about all the knowledge inside. It is feasible in the closed Intranet environment. However in the unbounded Internet, the knowledge dependency and knowledge contradiction are common, which make it highly inefficient to run a "Know All" system. On the other hand, to accomplish a specified task like finding a travel agent, only limited domain knowledge is needed. Furthermore, the consideration of privacy and security may stop the web site from opening all information to everyone.

We endorse the third "being told by a friend" model. A web page, marked with a semantic web language like OWL, appoints an existing agent that can be contacted to answer queries about the information on the page or to make statements about the

information. Thus the semantic web providers not only publish viewable web content to the human, but also provide agent-based services to other agents.

The features of this model include:

- There is no centralized agent who has to search all web pages and understand every ontology. The service agent in the distributed agent environment only need knowledge about a set of common ontologies and its domain ontologies;
- By marking up web pages, the best agent to ask questions can be easily located. This is a good amendment to the web services discovery and the agent services searching;
- The non-semantic web site with content encoded in HTML or XML can also join the semantic world and provides content for semantic query by linking to a service agent;
- The non-agent program can be wrapped with a service agent shell, allowing other agents to make use of those external web services, also allowing the web service program like the .NET program to interact with a service agent to exchange information and make use of the agent services;
- Owned by the semantic web site, the service agent can be trusted to access privacy-sensitive file or content, and can act as a proxy interface for outside agents;
- The service agent is able to accept comments and assertions about the web pages and publish those comments as a semantic web page.

The functions of the service agent are divided into four levels:

- The first level provides the requested semantic web page. For a semantic web site, the agent copies the file and sends it out. For a non-semantic web site, the agent finishes additional converting from a HTML/XML page to a RDF/OWL page;
- The second level answers simple questions about the semantic web pages. The inference in this level is based on local rules, limited semantic pages and local ontologies;
- The third level answers complex questions about the semantic web pages. When a service agent has insufficient knowledge to answer the question, it forwards the question to other service agents who have knowledge related to this question. The inference in this level involves multiple ontologies, multiple semantic web sites and multiple agents;
- The fourth level validates trust and delegation. The service agent can be used to verify the signature signed by an agent or prove what it believed to be TRUE. This part involves a set of complex inference rules.

3.4.2 OWL as ACL Content Language

The FIPA standards offer mature and published specifications for multi-agent infrastructure with an emphasis on agent communication languages (ACLs) and interaction protocols. We found the FIPA framework can be augmented with the semantic web languages RDF and OWL [Zou, 2003]. The rest of this section describes the choices we made for the content languages.

The content language is a language used to express the content of messages exchanged between the software agents. A good content language should be able to express rich forms of content, be processed efficiently and fit well with existing technologies. The FIPA communication infrastructure allows agents to communicate using any mutually understandable content language as long as it satisfies a few minimal criteria as a FIPA compliant content language [FIPA, 2003]. Published FIPA specifications provide a library of registered FIPA-compliant content language, including FIPA-SL, XML and RDF.

FIPA-SL is able to express complex knowledge, but there is no efficient way to generate or process the SL message. XML is adequate as a low level language for encoding information but falls short as a semantic language to express information at the knowledge level, even when augmented by more recent components such as XML Schema, XSL or through applications such as WSDL.

FIPA ACL RDF Content Language and Agent toolkit Jade suggest defining the ACL message content ontology and modeling the ACL message with the RDF language. An ontology defines a group of vocabularies and a set of relationships between the vocabulary elements. The Jade system has a software package containing a set of basic concepts (like True, False, Action). Jade also supports user-defined ontology, enabling the agents to create the content of messages, e.g., application-specific predicates and actions. A user-defined ontology is implemented through an object of class and is characterized by a name, a base ontology, a vocabulary, and a set of element schemata.

Jade restricts the description of ontology to the features that support inter-agent communication. To send and receive a message in Jade, both a predefined ontology providing the vocabulary, and the codes handling the syntax of the content language must register with Jade through the content manager before the communication begins. However on the semantic web, the agent may have no prior knowledge about ontologies used by other agents, thus can't register beforehand. Research work by Steven Willmott [Willmott May 2001] suggests building an abstract ontology representation (AOR) for the ACL message using the DAML language to support mapping of different content languages. Messages in KIF, SL, and XML can be translated into and generated from DAML. AOR imports the DAML language into the agent system and provides the mapping among multiple content languages. However, this mapping is neither efficient nor complete.

In this thesis, we define an ACL ontology in OWL language to support adding semantic into the agent communication. Compared with RDF, OWL has well-defined model-theoretic semantics as well as an axiomatic specification that determines the intended interpretations of the language. OWL is unambiguously computer-interpretable, thus making it amenable to agent interoperability and automated reasoning techniques. Adopting a stronger semantically rich content language like OWL facilitates a higher-level of interoperability between the agents. By agreeing on how meaning is conveyed, applications can share meaningful content easily and naturally.

The ACL ontology defines a FIPA-compliant content language. In addition to the basic required classes (such as Agent, ACLMessage and Service) and necessary expressive requirement (such as Proposition, Action and Reification), this ontology supports rules, queries and answers. We believe that OWL is the optimal choice as a generic ACL content language for four reasons. First, its expressive power as a knowledge representation language is adequate for many, if not most, needs of current agent based systems. Second, it offers better support for using terms drawn from multiple ontologies than existing ACL content languages. Third, as the semantic web language, it is designed to fit into and integrate with web-based information and service systems. Fourth, OWL has the potential to be a widely accepted and used representation language, enhancing the potential for interoperability among many systems.

To demonstrate that OWL is an adequate language for ACL content, we consider a list of test cases suggested in [Bothelo 2002]. These test cases are used as an expressive test for a candidate FIPA content language. The testing in [Bothelo 2002] also compares the result of encoding these test cases in SL, KIF, ebXML, Prolog and DAML. OWL might not be as expressive as SL, KIF or Prolog, but the OWL version of those test cases given below shows that OWL has enough expressive for most of tasks it might be asked to serve.

Expression	**Representation**	**Comment**
"Schrödinger's Cat is alive"	<Cat rdf:ID="schrödinger-s_cat"> <owner>Shrodinger</owner> <status> alive </status> </Cat>	There is a live cat in the world whose owner is Shrodinger.
"Cats are animals"	<owl:Class rdf:ID="cat"> <rdfs:subClassOf rdf:resource="#animal"> </owl:Class>	Cat is subclass of animal
"You are making the tea"	<fipaowl:Action rdf:ID="tea_action1"> <fipaowl:act>making-tea </fipaowl:act>	There is a making-tea action, "you" are the actor.

	<fipaowl:actor>you</fipaowl:actor> <fipaowl:Action>	
"Drinking too much is bad for you"	<Behavior rdf:ID="drinktoomuch"> <hasBehavior>excessive_drinking</hasBehavior> <healthy>bad</healthy> </Behavior >	The behavior of drinking too much is bad for your health.
"All red things"	<owl:Class rdf:ID="allredthing "> <owl:intersectionOf rdf:parseType="Collection"> <owl:Classrdf:about="#Thing"/> <owl:Restriction> <owl:onProperty rdf:resource="#hasColor" /> <owl:hasValue rdf:resource="#Red" /> </owl:Restriction> </owl:intersectionOf> </owl:Class>	The things whose colors are red.
"Any color a car might have"	<owl:Class rdf:ID="anycarcolor"> <rdfs:subClassOf> <owl:Restriction> <owl:onProperty rdf:resource="#color" /> <owl:allValuesFrom rdf:resource="#CarColor " /> </owl:Restriction> </rdfs:subClassOf> </owl:Class>	The color that limits the color property value in the car colors. This can also be a query: "Select color where color in Car Color"
"All things are hot"	<owl:Class rdf:about= "#Thing"> <rdfs:subClassOf> <owl:Restriction> <owl:onProperty rdf:resource="#tempterature" /> <owl:hasValue rdf:resource="#hot" /> </owl:Restriction> </rdfs:subClassOf> </owl:Class>	All things's temperatures are hot.
"Something is cold"	<owl:Thing rdf:ID= "cold_thing"> <temperature>cold</temperature> </owl:Thing>	There exist something whose temperature is cold.
"Herring or Perch"	<owl:oneOf rdf:parseType="Collection"> <owl:Thing rdf:about="#Vokda "/> <owl:Thing rdf:about="#Perch"/> </owl:oneOf>	
"Vodka and Tonic".	<owl:union Of rdf:parseType="Collection"> <owl:Class rdf:about="#Vodka " /> <owl:Class rdf:about="#Tonic" /> </owl:unionOf>	
"Not cricket"	<owl:Class rdf:ID="Noncricket"> <owl:complementOf rdf:resource="#Cricket " /> </owl:Class>	
"Success implies Payment"	<fipaowl:Rule> <fipaowl:Implies > <fipaowl:head > Payment</fipaowl:head > <fipaowl:body >Success </fipaowl:body > </fipaowl:Implies> </fipaowl:Rule>	The rule : Payment :- Success.
"Luis has the persistent goal that W"	<Person rdf:ID= "Luis"> <hasPersistentGoals> W </hasPersistentGoals> </Person>	
"Steve Believes X"	<Person rdf:ID="steve"> < hasProposition> < Belief rdf:ID="stevebelief1"> < believe>true</believe> < Statement > X</Statement> </Belief> </hasProposition> </Person >	
"Jonathan Desires Y"	<Person rdf:ID="Jonathan"> <hasProposition> <Desire rdf:ID="jonthandesire11"> <desire >true</desire> <Statement > Y </Statement> </Desire> </hasProposition> </Person >	
"Matthias Intends Z"	<Person rdf:ID="Matthias"> < hasProposition> <Intend rdf:ID="Matthiasintend1"> <intend>true</intend> <Statement > Z </Statement> </Intend> </hasProposition> </Person >	

FIGURE 15: OWL EXPRESSION TEST

Compared with other ACL content languages, OWL provides improved support in modeling, maintaining and sharing ontologies. The standard content languages such as SL and KIF offer no explicit mechanisms for ontology support. FIPA inherits the simple mechanism for ontology specification first used in KQML [Finin, 1992] that essentially requires that all content terms in a particular message be tagged as coming from a single ontology. Although variations and "work arounds" to this constraint have been proposed, implemented and used, none has been formally adopted as part of the stable FIPA specifications. The large scale and open multi-agent systems will benefit from OWL's support of multiple namespaces and multiple ontologies. Moreover, OWL has better support for other services essential to large scale open systems, such as the capability to map information from one ontology to another ontology, the ability to negotiate meaning or otherwise resolve differences between various ontologies.

The FIPA standards framework has a good set of primitive communicative acts, a way for the agents to define communication protocols [Odell, 2000], and a sound mechanism by which the agents can describe their capabilities and the supporting services. This framework will work well with the addition of OWL as a new content language.

3.4.3 Query Support

Among the most important communicative acts used by the agents are those designed to support query. The FIPA ACL has a primitive query model supporting just two acts: *query-if* and *query-ref*. Because the agents on the semantic web may have no prior knowledge about ontology or interaction protocol, an advanced query language is needed

to conduct the complicated query. The following is a list of previous works on the web content query and reasoning.

The eXtensible Stylesheet Language (XSL) proposes a language expressing style-sheets for XML documents. XSL is a transformation language mapping an input data tree (for example, an XML tree) to an output data tree. Although its primary role is to allow users to write transformations from XML to HTML describing the presentation of the XML document, it can also serve as a query language. An XSL program is a set of template rules that are executed in a best-match manner by recursively traversing over the nodes in the input data tree.

XML-QL combines XML syntax with query language. It uses path expressions and patterns to extract data from the input XML data. The variable is used to bind the data. It also provides templates that show how the output XML data is reconstructed.

The XML Query Language (XQL), an extension of the XSL pattern syntax, is a notation for addressing and filtering the elements and text of XML documents. It enhances XSL with Boolean operators, filters and indexing capabilities. XQL extracts data by patterns and path expressions, just like XSL. However, XQL provides easier syntax and more powerful selection mechanisms.

RDFDB by R.V. Guha [Guha 2003] supports a graph oriented API via a SQL-like query language. RDFDB is built on Berkeley's SleepCat database system and offers enough

functionality to do RDF data operation. It can load and reload an RDF file from a URI into the database and supports some basic forms of inference.

Squish by Libby Miller is another RDF query language. The difference between Squish and RDFDB is that Squish can make more complex queries of RDF models. Squish stores RDF in either Postgres SQL databases or in-memory model. The syntax of Squish resembles the syntax of SQL:

SELECT variables

[FROM url, url]

WHERE (pred variable/string variable/string)

[AND constraint]

[USING predString FOR predUrl]

RDF Data Query Language (RDQL) [Seaborne, 2003], an implementation of Squish in the Jena system, views RDF as a logical data model. Data is retrieved by returning a set of bound variables.

Logic programming also provides solutions for RDF query. The RDF statement is stored in files or relational databases. The inference engine then parses a query into a set of triples and returns results after running the query triples in a logic system (such as XSB). SRI DAML group suggests defining DQL language [Fikes 2002] by extending DAML syntax. The DQL query is parsed, interpreted and sent to a search site, which will search indexed DB for all matching objects. DQL also supports recursively searching through imported ontologies, translating to other ontologies with help from the ontology mapping

service and a specialty KB service. TRIPLE [Sintek 2002], a layered and modular query language for the semantic web, is designed to query and transform RDF models. Based on Horn logic, TRIPLE borrows many basic features from F-Logic.

We have experimented with the integration of a number of query solutions in our previous ITTalk work [Cost 2002] and RDF-based TAGA project [Zou 2003]. Since a consensus query system for the semantic web has yet to emerge, we have adopted an approach in which agents can use any one of several query systems and associated protocols. An agent specifies the supporting query languages and protocols as part of its basic service description. Other agents who intend to send queries to this agent are expected to encode the query strings in one of the supporting languages. A mediator agent handling several kinds of query languages offers a query translation service. For example, agent A wishes to send a DQL query to agent B, who understands only RQL. A query translation service supporting both DQL and RQL is able to provide the mediation service – receiving a DQL query from A, sending appropriate RQL queries to B, accepting the response from B, reformulating to fit the DQL protocol and sending back to A.

Similar to the query solution, the rule support takes the same approach, in which the agents can use any of several rule languages and systems. For example, the rule in the user case "if more than 20 people join the talk, I will go also" can be expressed in ruleML, SWRL or Prolog language. The agent extracts the rule content from the communication

message and sends it to the rule engine, which is responsible for parsing and processing of the rules.

3.4.4 Ontology Support

Ontology plays an important role on the semantic web and agents. To use the FIPA agent on the semantic web, the ontology for agent communication is needed.

The ontologies on the semantic web are distributed and managed by multiple parties. This distributed model is a better fit for deployment in the open web environment. There is no centralized site or agent that has to maintain or understand every ontology. Ontologies and rules are designed and implemented by service owners to reflect their business models and meet their requirements. The agent belonging to a service owner is responsible for answering the question related to the ontologies it uses. Ontologies are stored locally and accessible by the local agent. Those local personalized ontologies and rules also have the potential to resolve the problem of security and trust.

To support multiple ontologies in the FIPA system, an ontology service agent is used to enhance the agent's ontology operation on the semantic web. The support of ontology sharing and exchanging is achieved by a set of ontology-related actions:

- *NewInstance:* The receiver creates an instance using the specified ontology and the instance data provided;
- *OntologyQuery:* The sender queries the receiver about the definition of a term in an ontology;

- *OntologyShare:* The sender informs the receiver about the ontology definition, including Class/Property definition, Class-Subclass relationship and Class-Property relationship;
- *OntologyRelation:* The sender informs the receiver about the conversions and relationships among class or property terms defined in different ontologies. The relationships include extension, identical and equivalent. This message can be an inform message informing other agents about the relationships, for example, agent A informs agent B that the class Person is same class as the class Human used by agent B. The message can also be a query message that verifies the relationship or a request message that requests the converting of the ontology terms used in multiple ontologies.

FIPA assumes that the agent uses predefined agent interaction protocol. Since this might not be true in the open semantic web environment, the FIPA agent interaction protocol can also be encoded as ontology using OWL language. The interaction protocol ontology file can be accessed via the web and through the agent interface. The agents are able to learn a new agent interaction protocol by querying the ontology service agent.

3.5 Conclusion

This chapter presents a vision of software agents working in the service-based semantic web environment. The challenges come from three main areas:

- Heterogeneous: the software agents, semantic web and web services are three independent developed research communities. There have different working models and standards;
- Communication: the software agents, semantic web and web services need to discover and understand each other;
- Security/Authorization: working in the unbound Internet sparks more concerns about security and trust;

We further illuminate this vision with following premise: the consumers of the semantic web information is the software agents; the agents, in global scale, need the ontology-based semantic web to promote the agent communication and interaction; both human and software agents can directly access the semantic-marked web pages through web interface; an agent communicates with other agents by exchanging messages encoded in a semantic-rich agent communication language; both web services and agent services are described in a semantic-rich language to improve the interoperability; the personal agents and service agents work together to understand the semantic web content, automate web services and better serve humans.

We suggest an agent-based service model for this vision. We attach a service agent to the semantic web server. The service agent has knowledge about a set of common ontologies and its domain ontologies. It is always the best agent to answer questions about the knowledge and can be easily located. The non-semantic web site and non-agent program can also join the semantic world by linking to a service agent or service agent wrapper.

This also relieves the problems of security and trust because the service agent is owned by the semantic web site and trusted to access privacy-sensitive file or content.

To improve the interaction among the heterogeneous systems, we have designed an ACL ontology to support directly using OWL as the content language for agent communication. OWL has better support for the services essential to large-scale open systems, such as the support of multiple namespaces and multiple ontologies, the capability of mapping information from one ontology to another ontology, the ability to negotiate meaning or otherwise resolve differences between various ontologies. Finally, we propose our solutions for supporting multiple query languages and multiple ontologies.

CHAPTER 4 TRAVEL AGENT GAME IN AGENTCITIES

As a proof of concept discussed the Chapter 3, the Travel Agent Game in Agentcities (TAGA) system is designed and implemented to demonstrate how the agents and semantic web technologies working together. This chapter starts with the system design and implementation of the TAGA system. A list of new ontologies and interaction protocols are then introduced. Finally, the service architecture and F-OWL, an OWL inference engine are discussed.

4.1 System Design

The TAGA system is developed on the foundation of FIPA technology and the Agentcities infrastructure, which provide a stable communication environment where messages expressed in semantic web languages can be exchanged. The agents and services use FIPA supported protocols, semantic web language and web services

interfaces to create a travel market framework. This travel market is the combination of various types of auctions and varying markets including service registries, service brokerage, wholesalers, peer-to-peer transactions and bilateral negotiation. The TAGA system provides a rich test-bed for experimenting with the semantic web and web services technology as well as an interesting scenario to test and challenge the agent technology.

4.1.1 TAGA Architecture

TAGA provides a flexible framework to run the travel market game. As shown in Figure 16, the collaboration and competition among six types of agents who play different market roles simulate the real world travel market.

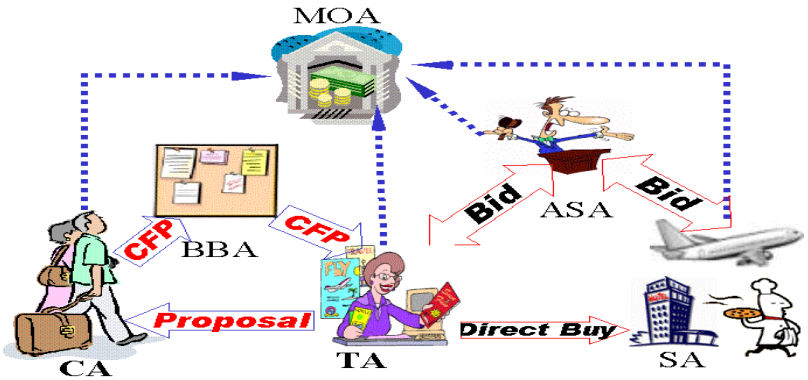

FIGURE 16: TAGA ARCHITECTURE

- *An Auction Service Agent* (ASA) operates all auctions in TAGA. Supporting auction types include English and Dutch auctions as well as other dynamic markets similar to Priceline.com and Hotwire.com;

- A *Service Agent* (SA) offers travel-related service units such as airline tickets, lodging or entertainment tickets. Each travel-related service has multiple providers with different service quality levels and with limited service units. It allows other agents to query its description (e.g. service type, service quality, location) and its inventory (the availability or price of a certain type of service unit). Other agents may directly buy the service units through published service interface. SA also bids intentionally in the auctions to sell its good, e.g. listing its goods in auction and waiting for the proper buyer. SA is responsible for maintaining inventory (mark service unit as sold) and reporting transaction record;

- A *Travel Agent* (TA) is a business that helps customers acquire needed travel service units and organize the travel plans. The service units can be bought either directly from the service agents, or through an auction server;

- A *Bulletin Board Agent* (BBA) provides a mechanism to help the customer agents finding and engaging one or more travel agents;

- A *Customer Agent* (CA) represents an individual customer who has particular travel preferences and constraints. The preferences include departure and return date, minimal hotel star level and a list of preferred entertainment activities. The constraints, for example *"the hotel selected must have swimming pool"*, limit the TA's selection of travel resource and travel

packages. CA's goal is to engage one or more TAs, negotiate with them over price and travel packages, and finally select one TA that is reliable to acquire all needed travel service units;

- The *Market Oversight Agent* monitors the game, validates every reported transaction and updates the financial model. It is responsible for the integrity of the running game.

TAGA is running in the Agentcity environment. From Figure 17, we conclude some of the key features of the TAGA system:

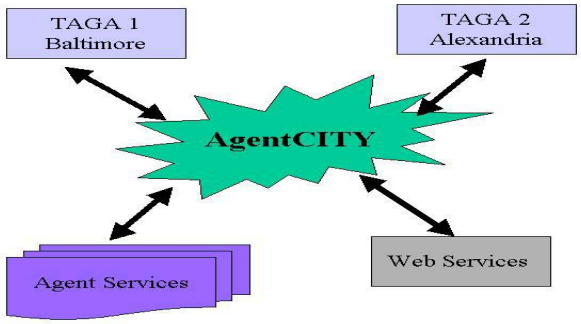

FIGURE 17: TAGA IN AGENTCITIES

- Service re-use: Wherever possible, TAGA attempts to reuse without modification existing agent service in the Agentcities and web service on the Internet;
- Service provider: TAGA publishes a number of generic and thus wholly-reusable agents, including travel service agents and auction server agents;

- Open system: The system is open in the sense that most of the individual agents can be accessed through Internet and invoked directly by external agent platforms.

4.1.2 Game Design

The TAGA system was designed as a general framework for running agent-based market simulations and games. The first use of TAGA was to build a travel competition along the lines that were used in the TAC Classic. In the competition, customers travel from City A to City B and spend several days before flying back. A travel package includes a round-trip airline ticket, corresponding hotel accommodations and tickets to entertainment events. A travel agent (an entrant to the game) competes with other travel agents in making contracts with customers and purchasing the limited travel service units from the travel service agents. The customer agent selects the travel agent who proposes the best travel itinerary. The objective of the travel agent is to acquire more customers, fulfill their travel packages and maximize the profit.

The following rules are used in the TAGA system:

- Game day: One game day is 10 minutes in real time;
- A travel itinerary is complete if it satisfies the following conditions:
    - It includes a round-trip airline ticket and hotel rooms for every night between the arrival and departure dates;

    - There can only be one hotel, i.e., the customer never moves to another hotel;
    - The return date must be later than departure date and cannot be on the same day;
  - The CA may receive travel package proposals from multiple TAs, but only one TA can be selected;
  - In response to the CA's travel request, each TA can submit only one proposal. The proposal cannot be canceled or modified.

The basic cycle of the TAGA game has the following stages:

- Game start: The MOA creates required tables and inserts data into the MySQL database, thus creating the game. The TAs register with the BBA, therefore subscribing to new customer informing services. The SAs register at the ASAs as auction sellers;
- A customer-generating agent creates a new customer with particular travel constraints and preferences chosen from a certain distribution and sends it to the BBA in the form of a Call For Proposal (CFP) message;
- The BBA forwards the CA's CFP message to each of the TAs that has registered with it. Each TA independently considers the CA's CFP and decides whether and how to respond;
- Once deciding to propose a travel package, the TA contacts the necessary ASAs and SAs to assemble a travel itinerary. Note that the TA is free to implement a complex strategy using both aggregate markets (ASAs) as well as direct negotiation with the SAs. The proposal to the CA includes the travel itinerary, a set of travel service units,

the total price and the penalty to be suffered by the TA if failing to complete the transaction;

- The CA negotiates with the TAs, ultimately selecting one from which to purchase an itinerary based on its constraints, preferences and purchasing strategies (which might, for example, depend on a TA's reputation);
- Once having a commitment from the CA, the TA attempts to purchase the service units in the itinerary from the ASAs and SAs. There are two possible outcomes: the TA acquires the needed service units and completes the transaction, resulting in a satisfied CA and a profit or loss for the TA, or the TA is unable or unwilling to purchase all of the units, resulting in an aborted transaction and the invocation of the penalty (which can involve both a monetary and a reputation component);
- All transactions are reported to the MOA, who is responsible for verifying them based on reports from multiple parties. A valid transaction owns two reports from both seller and buyer and has the unique transaction ID. A travel package is valid if the travel itinerary is complete and all related travel resources are reserved with valid transactions. The MOA also controls the finance records and transfers money from buyer to seller;
- The game runs continuously. The transactions and finance records are published at TAGA web site in real time. The TA who has the most money is the current winner of the game.

4.1.3 TAGA Auction

In addition to supporting the standard auction types listed in Chapter 2, TAGA introduces two reality-based auction types.

Priceline auction is used in TAGA's airline auction. It is based on the market model of www.priceline.com. The buyer submits a travel request including the destination and the travel date. The auction server returns a list of constraints and selections. The buyer accepts a few constraints based on his preferences and budget limit, and then offers a price that he can afford. The auction server forwards the travel request to all registered service providers. The service providers decide whether to accept the offer based on their inventories and auction strategies. The first service provider responding with "Accept" wins the auction. If no one accepts the offer, the buyer resubmits the request after increasing the offer price and selecting more constraints. The most important feature of this auction type is that the buyer exchanges the convenience and certainty for lower price. Compared with a direct-buy from the service provider, the buyer may have to wait until a provider willing to accept the offer. The provider's information and quality of the goods are uncertain until the auction is finished. The obvious benefit of participating in the expectation of paying a price lower than the market value.

Hotwire auction, used in TAGA's hotel auction, is based on the market model of www.hotwire.com. The buyer submits a travel request, which includes a destination and preferred travel date. The auction server returns a list of hotel prices in diverse star levels, lower than the market price in most cases. But the buyer receives no information about the hotel like the name, location, or facilities. The buyer may sign the contract right away

and reserves the hotel room. The auction server randomly selects a hotel room from the hotels participating in the auction. Unlike the Priceline auction, the buyer receives the hotel room right away. However the auction server rather than the buyer offers the price. The service provider's information and quality are uncertain until the auction is closed. The obvious benefit of participating in this auction is the possibility of acquiring the hotel room with a price lower than its market value.

4.1.4 Conclusion

This section describes the TAGA system, a framework extending and enhancing the TAC system to work in an open multi-agent systems environment of FIPA compliant systems. Unlike the TAC, who uses a straightforward client-server architecture in which a single TAC server manages all of the travel service suppliers as well as the customers, the TAGA system offers a more realistic model where customers, service providers and various kinds of "middlemen", including market providers, operate as autonomous peer agents. Moreover, agents can develop complex strategies, which involve a combination of direct transactions (e.g., TA buys directly from the hotel agent) as well as auction-mediated transactions of various kinds. Finally, adopting a multi-agent systems approach supports an environment where all aspects of commerce, such as service discovery, information seeking, negotiation, decision-making, commitment and transaction execution, can be integrated in a more natural manner.

4.2 Ontology

Three ontologies are designed to reflect the travel domain and agent communication. The travel ontology covers the basic travel business concepts in TAGA, including the travel itinerary, customers, travel services and service reservations. The auction ontology defines the different kinds of auction types, the roles played by the participants and the interaction protocols. The ACL content language encodes the ACL message in OWL language.

The ontologies may refer to qualified names using one of the following namespace names:

Namespace name	**Namespace**
Rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
Rdfs	http://www.w3.org/2000/01/rdf-schema#
Xsd	http://www.w3.org/2001/XMLSchema#
Owl	http://www.w3.org/2002/07/owl#

Please note that the introduction in this section is an overview, please refer to the appendix for complete ontologies.

4.2.1 Auction Ontology

This ontology describes the auction business. The auction Class defines the auction.

(Auction

:auctionHouse *Agent*

:seller *Agent*

:hasAuctionItem *AuctionItem*

:auctionType English/Dutch/Double/Priceline/Hotwire

:closeTime xsd:Date

:reservePrice xsd:Integer

:status: Open/Close/Failure

:**hasBidding** *Bidding*

)

The two actors creating the auction are the seller who actually owns the goods and the auction house who shouts out for the seller. They are both agents in TAGA. The Auction Item refers to the goods for auction. It may also be the combination of multiple auction items, for example, a travel package includes two airline tickets and one hotel room. Two important properties of the auction are closing time and reserved price. These properties may not present in all auction types, for example, the English auction has no closeTime because it ends when no one is bidding anymore. There are three statuses for the auction. The auction starts with "Open" status. When the auction is closed and items are sold, the status changes to "Close". When the items are unsold, the status turns to "Failure". The hasBidding property links to all submitted bids.

The detail about the auction item is described in the AuctionItem Class.

(AuctionItem

:itemName xsd:String

:itemDescription xsd:String

:itemQuantity xsd:Integer

:itemQuality Thing

:suggestRetailPrice xsd:Integer

)

The itemQuality means the quality of the goods. For hotel, it indicates the star level. For other items, it can be, for example, “broken”, “as-is”, or “brand-new”.

The bidding Class includes the information about the submitted bid.

(Bidding

:bidder *Agent*

:expire xsd:Date

:bidPrice xsd:Integer

:bidQuantity xsd:Integer

:reserveResource Thing

)

The bidder is the agent who participates in the auction and submits the bid. The bid has an expiration time, which is either a duration of time or the ending of the auction (which means never expire). The reserveResource refers to the resource reserved for the bid by the buyer (bidder). Note here that the auctionItem in the auction Class is the resource reserved for the auction by the seller. For example, in a Priceline auction, the customer creates the auction stating the needed airline ticket and the suggested price. The airline company must reserves the related resource (air tickets) when submitting the bid and

make sure the tickets are available when it wins the auction. Of course when the bidder loses the auction, the reserved resource will be immediately released. The reserved resource may also be other resources related to the auction items, for example, a reserved IP address when the bidder bids for a new computer, or the money on hold in the bank account when bidding for the auction item.

4.2.2 ACL Ontology

The agents exist physically on an agent platform and utilize a list of standard agent services, which includes DF, AMS and ACC.

(AgentPlatform

:platformName xsd:String

:hasServices *Services*

)

The agent Class is defined as:

(FIPAAgent

:agentName xsd:String

:agentAddress xsd:anyURI

:atPlatform *AgentPlatform*

:provideService *Service*

:agentStatus waiting/suspended/transit/initiated/active

)

The agentName is the agent's unique identifier on the host agent platform. The agentAddress describes the accessing interface. The most popular access interfaces include IIOP and HTTP. The provideService links the agent to its service function. The agentStatus shows the four different statuses in an agent's life cycle: the status is "initiated" when the agent program is starting up and becomes "active" when starting is finished; the status becomes "suspended" after a suspend action and returns to "active" after a resume action; the status turns to "waiting" after a wait action and returns to "active" after a wakeup action; the status changes to "transit" after running a move action to move the agent to a new platform and returns to "active" after an execute action.

The agent provides services for the human user and other agents. The service Class connects the agent and the services together.

(Service

:serviceName xsd:String

:serviceProfile xsd:anyURI

:serviceType xsd:String

:ownership xsd:String

:serviceProtocol xsd:String

:serviceOntology xsd:String

:serviceLanguage xsd:String

serviceParameter Thing

)

The serviceProfile links the service provider with the service description file. Usually, it is a WSDL or OWL-S file. The agent who wishes to use the service composes an understandable ACL message with the serviceOntology, encoding it in the serviceLanguage. Therefore, the serviceLanguage and serviceOntology help the agents better understand each other.

The ACLMessage Class describes what is inside the message:

(ACLMessage

:performative xsd:String

:sender *FIPAAgent*

:receiver *FIPAAgent*

:content Thing

:reply-to *FIPAAgent*

:language xsd:String

:encoding xsd:String

:ontology xsd:anyURI

:protocol xsd:String

:conversation-id xsd:String

:reply-with xsd:String

:in-reply-to xsd:String

:reply-by xsd:Date

)

The content property contains the exchanged information between the agents. Some of the most frequently used contents include Action, Question and Rule.

(Action

:act xsd:String

:actor *Agent*

:target Thing

:actionParameter Thing

)

An action message from the agent A to the agent B implies that A asks B to take an action. For example, Joe asks John to close the door. The actor "John" is the subject who carries out the action. The act "close" is the action to be taken. The target "door" is the object of the action. Some frequently used act properties in the ACL include: forwardMessage, register, deregister, newInstance and informOntology.

The result of executing the action is defined as:

(Result

:fromAction *Action*

:done xsd:Boolean

:output Thing

)

The done property describes whether the action is finished. The output property includes the output from action, for example, the hotel reservation action has the output reservation ID.

The execution of the action also has effects on the environment of the agent:

(Effect

:fromAction *Action*

:oneEffect Thing

)

For example, the effect of closing the door in the above example is that the status of the door changes from “open” to “close”.

The question and answer Class cover how the agents submit the query and retrieve the answer.

(Question

:questionLanguage xsd:String

:ask Thing

:answerNumber xsd:Integer

)

The questionlanguage is one of the query languages supported by the answering agent. The widely used query languages in FIPA include FIPA-SL and KIF. The RDF query languages like RQL, Squish or RDQL can also be used. The answerNumber limits the expected size of the answer set.

(Answer

:hasQuestion *Question*

:hasAnswer Thing

:questionFailure xsd:Boolean

:failureReason xsd:String

)

The hasAnswer contains the answer set encoded in the question language. In case the query has failed, the failure reason is listed in the failureReason property.

The following is an example of using RDQL language inside a query:

<fipaowl:Answer>

<fipaowl:hasQuestion>

<fipaowl:Question>

<fipaowl:questionLanguage>rdql</fipaowl:questionLanguage>

<fipaowl:ask>

"SELECT ?x,?y

FROM <people.rdf>

WHERE (?x,<dt:friend>,?y),(?y,<dt:friend>,?x)

AND ?x<^gt;?y

USING dt for <http://foo.org#>, rdf for http://www.w3.org/1999/02/22-rdf-syntax-ns#"

</fipaowl:ask>

<fipaowl:result_number>10</fipaowl:result_number>

</fipaowl:Question>

</fipaowl:hasQuestion>

<fipaowl:hasAnswer>

Array ([0] => Array ([?x] => http://foo.org/persons/Carl [?y] =>

http://foo.org/persons/Peter) [1] => Array ([?x] => http://foo.org/persons/Peter [?y] =>

http://foo.org/persons/Carl))

</fipaowl:hasAnswer>

</fipaowl:Answer>

The receiving side is expected to extract the answer from the message and process appropriately based on the query language.

Rule also play a significant role in the agent communication:

(Rule

:ruleLanguage xsd:String

:ruleContent Thing

)

Similar to the query, no individual language is specified for encoding the rule. The popular rule languages in FIPA include FIPA-SL and KIF. Other supporting rule languages include ruleML and SWRL .

The following is an example of using SWRL to express the rule: **animal(X1) :- man(X1).**

<fipaowl:Rule>

<fipaowl:ruleLanguage>swrl </fipaowl:queryLanguage>

```
<fipaowl:ruleContent>
<swrl:Variable rdf:ID="X1"/>
<ruleml:Imp>
  <ruleml:body rdf:parseType="Collection">
    <swrl:individualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="man"/>
      <swrl:argument1 rdf:resource="#x1" />
    </swrl:individualPropertyAtom>
  </ruleml:body>
  <ruleml:head rdf:parseType="Collection">
    <swrl:individualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="animal"/>
      <swrl:argument1 rdf:resource="#x1" />
    </swrl:individualPropertyAtom>
  </ruleml:head>
</ruleml:Imp>
</fipaowl:ruleContent>
</fipaowl:Rule>
```

The receiving agent is expected to extract the rule from the message and insert it into the knowledge base.

4.2.3 Travel Ontology

This ontology defines the travel business. There are three roles involved: customer, travel agent and service provider.

(Customer

:custId xsd:Integer

:budgetLimit xsd:Integer

:hasItinerary *Itinerary*

hasPreference *Preference*

)

The customer's preference is composed of the preference values for hotel, airline and entertainment. The travel agent's task is to organize the travel plan and fulfill the customer's travel itinerary.

(Itinerary

:from xsd:String

:to xsd:String

:departureDate xsd:Date

:returnDate xsd:Date

:travelPlanReady xsd:Boolean

:travelAgent *Agent*

:offerPrice xsd:Integer

:**penaltyPrice** xsd:Integer

:**hasReservation** *Reservation*

)

The offerPrice is the travel agent's asking price for a travel package. The penaltyPrice is the compensation paid to the customer when the travel agent fails to reserve all needed travel service units and fulfill the travel itinerary.

The reservation is a contract between the customer and the service provider:

(Reservation

:forCustomer *Customer*

:hasServiceProvider *ServiceProvider*

:reserveNumber xsd:Integer

:price xsd:Integer

)

The ServiceProvider Class links the provider to its service Class.

(ServiceProvider

:serviceProviderName xsd:String

:serviceType xsd:String

:hasService Service

)

The value of serviceType in TAGA is can be Cinema, Restaurant, Opera, Accommodation or Transportation.

4.2.4 Conclusion

This section presents the three ontologies used by the TAGA system. The travel ontology reflects the travel domain, which covers the basic travel business concepts in TAGA. It has eleven OWL Classes, including the travel itinerary, customers, travel services and service reservations. It also has twenty Object/Datatype properties, covering the property of customer, services and travel planning. The auction ontology defines the term used in the auction market, including the different kinds of auction types, the roles played by the participants and the interaction protocols. It has four OWL Classes and twenty-two Properties. The ACL ontology encodes the ACL message in OWL language, including agent ID, address, message, and services. The supporting of query and rule are also addressed. The ACL ontology includes thirty-one Classes and fifty-one properties, which generates around 230 triples. Lots of OWL's descriptive features are used in these ontologies, for example, UnionOf, InverseOf and OneOf. The ontologies are published at DAML ontology library and SchemaWeb. We hope people working on the trading market and agent communication can reuse them.

4.3 Implementation

As a platform for experimenting with the semantic web and agent technologies, TAGA is designed to be heterogeneous and diverse. The multiple running agent platforms use various agent systems and programming languages. The services are described and published in multiple formats.

4.3.1 Agent Platform

TAGA has two public agent platforms registered at Agentcities.

Name:	umbctac.agentcities.net	taga.agentcities.net
Platform:	April Agent Platform (AAP)	April Agent Platform (AAP)
Location:	Alexandria, USA	Baltimore, USA
MTP-address:	http://youyong.homelinux.net:8080/acc	http://taga.umbc.edu:8080/acc

In addition to hosting the databases and web servers, these two platforms act as the game servers. They publish travel and auction services to Agentcities's global DF, respond to the travel service requests, and provide database and web server functions.

The individual travel agent or service agent may run on an independent agent platform, the Jade system in most cases. Those agents may not register at the Agentcity platform, though they are able to search and locate TAGA agent platform and other services through the Agentcity's global DF and AMS.

Agents

The TAGA system is composed of six types of agents who play different market roles. They collaborate and compete with each other to simulate the real world travel market.

4.3.2.1 Customer Agent

There are two types of customer agents in TAGA. The AAP customer agent provided by the TAGA platform generates a new customer every few minutes and acts as the default customer agent. The Jade customer agents developed by the game participants support more complicated preference values and sophisticated game strategies.

The customer agent is a special type of Personal Assistant Agent. Its sole task is to arrange the travel plan for the customer. Each customer has a personal profile including the name, contact information and travel preference. Once a new customer is created, the Customer Agent (CA) loads the RDF-encoded profile and learns the traveler's information. Then, the CA begins searching for the Bulletin Board Agent (BBA). Once a BBA is located, the CA composes a *CFP* message, wraps it inside a *PROXY* message and sends out. The message content is the action "OrganizeTravel" whose target property is a customer instance with a blank itinerary.

Customer_CFP::Proxy_ACLMessage(CFP_ACLMessage(CustID, Prefer_FlyIn, Prefer_FlyOut,Prefer_Hotel, Prefer_Entertainment, Budget_Limit))

CustID, Prefer_FlyIn, Prefer_FlyOut::Integer

Prefer_Hotel, Prefer_Entertainment, Budget_Limit:: Integer

After receiving the proposals from the Travel Agents (TA), the customer selects one TA to sign the contract. The contract is sent back to the TA and the Market Oversight Agent. The CA has no control of the contract result. If the TA fulfills the travel itinerary, the CA pays the offer price to the TA; otherwise, the CA receives the penalty price for compensation.

One challenging problem for the game players, who choose to play the role of customer agent, is the human interface. This involves designing a convenient GUI interface, learning and managing customer preference automatically and interfacing with existing personal software like Microsoft Outlook. Another problem is the smart decision in

choosing a TA to carry out the job of organizing the travel package. The decision is based on offer prices, penalty prices, suggested itineraries and the reputations of the TAs. The customer may contact the chosen service providers in the proposed itinerary and queries for more information, like availability of a special facility (e.g. valet parking, or smoke-free).

4.3.2.2 Bulletin Board Agent

The multi-agent systems containing a number of heterogeneous agents typically rely on the infrastructure that supports service discovery and agent interoperation. Problems arise when the agents providing the services are running on an unstable global network and may change status (restarting agent platform, host being shutdown or modifying the service description) at any time. A middle-agent, mediating the communication between the agents, provides an efficient way to discover the agents with specific capability descriptions. It supports the flow of information, assisting in locating and connecting the ultimate information requester with the ultimate information provider. The Bulletin Board Agent (BBA) is such a middle-agent between the CA and the TA. It assists the CA in discovering the TAs based on a desired service capability description. The BBA has two functions: the matching function that depends on the capability description and the communication function that determines how to mediates the agents. With the introducing of the BBA, the TAGA system becomes more robust and adaptive.

There are four steps in the BBA's processing model:

i. *The TAs register with BBA:* The registration contains the parameters representing the service description and the location of the service profile. The BBA supports additional actions including deregister and modify.

ii. *The CA queries the BBA:* A query message describes what the CA is looking for, including the service type and additional properties like reputation or location, is received by the BBA. The Customer's profile and preference information are also submitted along with the query message.

iii. *The BBA locates the TAs:* The query is matched against the service description of the registered TAs and a list of active TAs is chosen. The BBA is responsible for maintaining the active TAs list by sending out a ping message periodically.

iv. *The BBA sends CallForProposal Message to the TAs:* The BBA decomposes the query message and retrieves the customer's information. A new CallForProposal message is generated and sent to the chosen TAs.

4.3.2.3 Travel Agent

TAGA supplies Travel Agent Shell written in the April language. This shell provides a set of libraries to free the game player from the tedious communication processing and focus on the game strategies. The OWL library includes multiple functions related to the OWL message: generating an OWL message; parsing an OWL message into triples; querying an OWL message. The service library provides the functions to access the service providers: checking availability and price of a travel resource; checking the auction status; submitting a bid; creating a new auction instance. The database library

includes the functions for connecting to the MySQL database running at TAGA Server and updating the game data. The agent library includes the functions related to the agent platform and communication: registering at DF and AMS; registering at the BBA; searching the service providers based on the service type; wrapping the OWL content inside an ACL message; sending out the message via the HTTP/SOAP interface.

The TA shell uses a RDF-encoded profile to specify a few game strategies. The game strategy Class inside the profile includes two properties: *BidHotel* specifies whether to participate in the hotel auction; *BidAirline* specifies whether to participate in the airline auction. Three dummy travel agents are provided as examples to demonstrate how to design the travel agent using the shell and profile. They run on the TAGA server continuously and serve to enhance the competition in the game. The dummy TA one uses a conservative strategy: it buys only the needed travel resource directly from the service providers. The dummy TA two participates in the hotel auction and tries to find the best deals, but acquires the travel resource only after winning the customer. The dummy TA three is the most aggressive one among the three: it participates in the airline auction and tries to find the best deals; it intentionally buys the available travel resources at low price and later assigns them to the winning customer.

TAGA also supplies the Jade Travel Agent Shell written in Java language. Because of Java, this shell has better support of GUI interface, providing a better way to watch the TA's behavior and monitoring the communication in the system.

Given a description of the market and a set of the agent preferences, there are many issues involved in automatically determining how to organize the travel package and submit the bidding. Some important factors that may affect the game strategies include:

- Auction participating: Usually the auction ends with the buyer paying the price lower than the market price, but the uncertainty (winning of the auction and the duration of the auction) makes the game strategies more complex;
- Buying before having the customer: Acquiring the travel resource when the price is low means bigger profit and may have big advantage in competing for the customer when the supply of travel resource is tight. But without a sound planning theory and an effective resource allocating algorithm, the wasted travel resources will cost money;
- Offer price: The lower offer price may attract more customers but also means less profit for the TA;
- Penalty price: The higher penalty price may give the customer more confidence to select this TA, but it is also more to bear when the supply of the travel resource is tight;
- Reputation: The good reputation, equals to trustworthiness in most cases, helps to attract more customers. To gain the good reputation, the TA is expected to satisfy the customer's travel plan whatever happens; sometime this means paying an extremely high price to acquire the scarce travel resource.

The TA is responsible for organizing the travel package with the services distributed on the heterogeneous platforms. It communicates with the service agents and copes with

ontologies sharing. The TA also coordinates the dependence among multiple services, for example, failing to book the airline ticket will affect the already booked hotel room. There are many interesting auction, game theoretic and planning algorithms available to help design a smart TA.

4.3.2.4 Service Agent

Each service agent in TAGA is a travel service provider. It supports the following actions:

- Querying: An agent sends a *QUERY-REF* message to the service agent with a query about the service, such as available travel resource, available facilities, suggest retail price or location of the service description profile;
- Booking: when a booking *REQUEST* message is received, the service agent reserves a service resource unit as "sold" and returns a reservation ID;
- Reporting: The booking transaction is reported to the Market Oversight Agent, who transfers the money between the bank account of two parties involved in the transaction;
- Bidding: the service agent actively participates in the auctions held by the Auction Server Agent. The airline service agent gets notified once a new auction instance is created by the TA. It independently decides whether to accept the offer price based on the availability and the market price. The service agent also seeks to sell extra service units by creating a new auction instance.

The aim of the game participant who plays the service agent role in TAGA is to gain bigger profits. One of the most important factors is the management of the travel resource inventory. While selling more resources brings more gross income, the service agent can make a bigger profit by adjusting the price dynamically based on the supply and demand of the market. With the appropriate auction strategy and efficient inventory management, the service agent can sell more resources and gain bigger profits.

4.3.2.5 Market Oversight Agent

The Market Oversight Agent (MOA) controls the game and ensures the integrity of the system. It has many roles in TAGA. The first role is the accountant. All the transactions and contracts are reported and saved at the MOA. They can be queried by submitting an appropriate *QUERY* message. The second role of the MOA is the bank. Each agent participating the game has an account with the MOA. The MOA is responsible for transferring money between the buyer and the seller following a successful finished transaction. The failed transaction triggers paying the penalty and being given a bad reputation score. The third role of the MOA is the auditor. It checks each transaction and catches the possible frauds and mistakes by verifying with both parties. The fourth role of the MOA is the reputation server. It maintains the reputation records, updating them after every reported transaction. The TA who fails to fulfill the travel itinerary receives a bad reputation score. The fifth role of the MOA is the game controller. It is the first agent started in the game and is responsible for setting up the game environment. When the game ends, the MOA publishes the game result and announces the winning agent.

Since the TAGA is running on the open Agentcities platform, the agents cannot assume that they are interacting with honest entities or in a safe environment. Thus, security and trust become important in designing the MOA. The dominant standards for security are encryption and X509 certificate. We believe that the security should be considered in the lower level (message passing level) of agent infrastructure and the security should always be invisible to the agent. The MOA is trusted by every agent to manage the bank account and commit the transactions. The reported transactions from the travel agent are verified to help prevent the travel agent from claiming the resource it doesn't own.

4.3.2.6 Auction Server Agent

There are two Auction Server Agents (ASAs) in TAGA system: hotel auction server and airline auction server. The ASA's activities are supported by the following four types of messages:

- Query message: The ASA supplies the bidding agent with the auction information (a more general term is the price-quote);
- Request message: The ASA creates a new auction instance for the requested auction item;
- Propose message: The ASA accepts a new bid. The bid is checked against the auction rules, being rejected if invalid.
- Inform message: The ASA clears the auction. A new transaction is generated and sent to both parties involved in the transaction as well as the MOA. The

message specifies the terms of the transaction including the auction item, two trading partners and the price.

The auction item in the hotel auction is the hotel rooms. The hotel agent creates a new auction instance by sending the hotel ASA a *REQUEST* message with the available date and suggest retail price of the hotel rooms. The ASA decides the auction price based on the average value of the suggest retail prices. After submitting the bid (no bid withdraw allowed), the TA learns more information about the auction item like hotel start level or hotel address.

The auction item in the airline auction is the airline ticket. The TA who creates the auction specifies the reserve price. The airline agent who first submits the bid wins the auction. When no bid is received after a pre-defined timeout period, the TA may create a new auction instance with a higher reserve price. There is no limit on the timeout value or how many auction instances a TA can create.

4.3.3 Agent Communication

The TAGA system uses OWL as the content language for agent communication. The ACL content ontology described in Chapter 4.2 provides the basic required classes (e.g., Agent, ACLMessage and Service) and necessary expressive requirement (such as Proposition, Action and Reification). It also supports expressing the rule, the query and the answer. The most common-used message contents in TAGA include:

- **Statements:** e.g., the price of one room at Hilton Hotel this Sunday night is $100;

- **Requests:** e.g., create an airline auction instance for this Sunday from Boston to New York;
- **Contracts:** e.g., if TA1 is able to organize the travel package, customer Joe pays $400 to TA1, otherwise, TA1 pay $200 compensation to Joe.
- **Policies:** e.g., to win customer Joe's contract, the travel agent must have a better-than-average reputation (computed by comparing customers with fulfilled travel package vs. all served customers).

The receiving agent extracts the OWL-encoded content and sends it to the inference engine (TAGA uses F-OWL system) for further processing. The inference will be discussed in more detail in Section 4.5.

4.3.4 Interaction Protocol

TAGA system favors the ideas of using the standards to promote and improve the agent communication. Lots of FIPA standards and specifications, including agent communication and interaction protocols, are used in the TAGA system. However, in order to support two new novel auction types suggested in Chapter 4.1 and the unique interaction model used by the MOA, a set of new agent interaction protocols are designed and used in TAGA system.

4.3.4.1 Priceline Interaction Protocol

A new interaction protocol (Figure 18) is defined to support the Priceline auction.

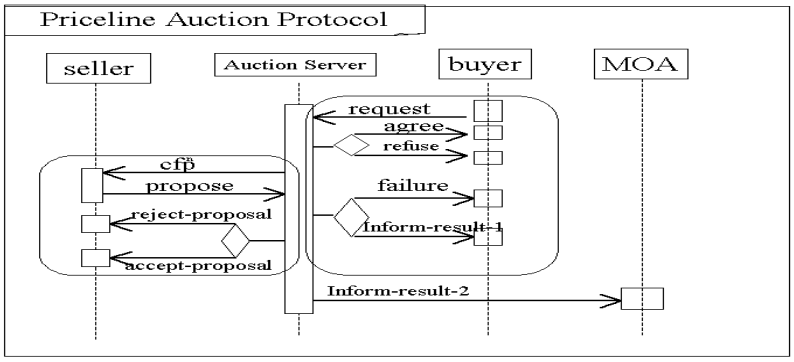

FIGURE 18: PRICELINE AUCTION INTERACTION PROTOCOL

The Priceline Auction Interaction Protocol is composed of two FIPA protocols. The *FIPA Request Interaction Protocol Specification* is used for creating the auction and announcing the auction result. The Auction Server Agent (ASA) creates the auction instance when it receives a *REQUEST* message from the buyer (TA) and sends back an *AGREE* message. A refuse message is sent back if it is unable or unwilling to create the auction. The auction server uses the *FIPA Propose Interaction Protocol Specification* in the bidding process of the auction. It starts with sending out a *CFP* message to all known seller agents. Based on the target retail price and stock number, a seller agent decides whether to accept the offered price. If the seller agent decides to sell the goods, it sends a *PROPOSE* message to auction server. An *ACCEPT-PROPOSAL* message, attached with a signed contract, is sent out when the auction server receives the first valid *PROPOSE* message. Other incoming proposal messages will subsequently be rejected with a *REJECT-PROPOSAL* message. The auction server informs the buyer of the auction result and reports the contract information to MOA. If no proposal message is received, the auction expires after the timeout period and an appropriate *FAILURE* message is sent to the buyer.

4.3.4.2 Hotwire Interaction Protocol

Figure 19 shows the Interaction Protocol of Hotwire Auction:

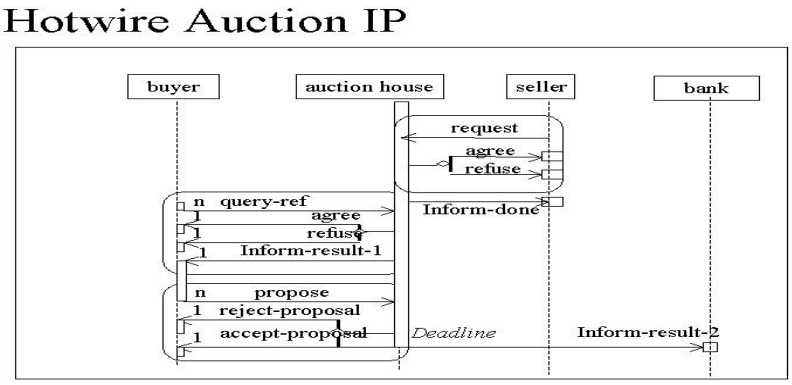

FIGURE 19: HOTWIRE AUCTION INTERACTION PROTOCOL

The seller and the Auction House (AH) start the interaction according to the *FIPA Request Interaction Protocol Specification*:

- The seller sends *REQUEST* message to the AH listing its travel resource unit (a certain type of service on a certain day) for sale;
- Having received the *REQUEST* message, the AH decides whether the unit is valid for the auction. For example, a hotel room on Feb 15 can't be sold after Feb 15. If the unit is valid, the AH sends an *AGREE* message to the seller, otherwise, a REFUSE message is sent to the seller and no further interaction;
- The AH puts the valid unit in the corresponding auction instance based on different properties (TAGA uses the hotel star level), or creates a new auction instance if no such instance is found;

- The asking price of the unit in a particular auction is based on the availability and service quality property. TAGA uses the average price of all listed units in the auction;
- Finally, the AH sends an *INFORM-DONE* message to the seller.

To find a desired travel resource unit, the buyer interacts with the AH following the *FIPA Query Interaction Protocol Specification*:

- The buyer sends *QUERY-REF* message, containing a query encoded in a specified query language, to AH;
- If the query is valid, the AH sends an *AGREE* message to the buyer, otherwise a *REFUSE* message;
- The AH sends an *INFORM*-Result-1 message, which contains the query result (a list of prices), to the buyer;

Agreeing to buy the unit at a specified price, the buyer interacts with the AH following the *FIPA Propose Interaction Protocol Specification*:

- The buyer sends a *PROPOSE* message to the AH;
- The AH makes a decision: if the buyer's offer price is lower than the current asking price in the auction, the proposal is rejected. Otherwise, the auction closes successfully and an *ACCEPT-PROPOSAL* message is sent from the AH to the buyer. (Note that the AH randomly picks one service unit from one of the participating auction sellers);
- The AH sends the bank an *INFORM*-Result-2 message reporting the transaction.

4.3.4.3 Dynamic Contract Interaction Protocol

The Dynamic Contract Interaction Protocol is defined as shown in Figure 20 to facilitate agents in making contracts with other agents dynamically in a mediated system. The recruiter (BBA) helps the initiator (CA) to find the appropriate group of participants (TA). All participants (TA) are candidates who can enter into a contract with the initiator, but only one will be successful. Once the contract is struck, the MOA joins the post-contract activities to ensure the two parties fulfill the contract: either the initiator pays for a successful contract or the participant pays the penalty of being unable to fulfill the contract.

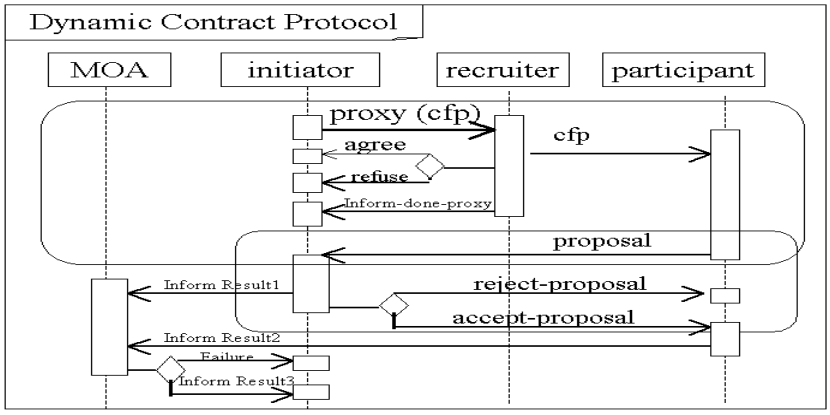

FIGURE 20: DYNAMIC CONTRACT INTERACTION PROTOCOL

This protocol is composed of two standard FIPA interaction protocols. Initially, the *FIPA Recruiting Interaction Protocol Specification* is used for the initiator to find a list of participants with the help of the recruiter: the initiator sends a *PROXY* message to the recruiter with an embedded *CFP* message; if the recruiter can't find any participant, it sends a *REFUSE* message back to the initiator, otherwise it sends an *AGREE* message. The recruiter forwards the *CFP* message to all known participants and sends an

INFORM-DONE message back to the initiator when finishes. Once the *CFP* message is received, the participant evaluates the information and decides whether or not to send a *PROPOSE* message to the initiator.

If the participant decides not to submit a proposal, no further action is required. Otherwise, the participant interacts with the initiator following the *FIPA Propose Interaction Protocol Specification*: the participants send a *PROPOSE* message containing the proposed contract to the initiator; the initiator selects the most profitable proposal and sends an *Accept-Proposal* message to the selected participant; the other participants receive a *Reject-Proposal* message. An *Accept-Proposal* message from the initiator to the participant means a contract signed between the two parties. The participant needs to acquire the resource units and fulfill the contract. The MOA is responsible for monitoring the result of the contract and informing the related parties

4.3.5 Conclusion

This section describes the implementation of the TAGA system. At first, the two platforms registered at the global Agentcities platform are described. Then, the function and processing model of every agents running in the system are explained in detail. The communications among multiple agents, along with the examples of message content, are discussed. Finally, three new interaction protocols are designed to improve the interaction among the trading partners.

The TAGA system started as an entrant to the Agentcities Technology Competition in October 2002. Its initial goal was to play the TAC Classic game in the Agentcities environment and enrich the game with new technologies like the semantic web and web services. At the same time as TAGA was designed, the researchers from CMU and Sicstus designed the TAC SCM game. The following figure compares TAGA with TAC Classic and TAC SCM.

	TAC Classic	TAC SCM	TAGA
Game	Travel	Supply Chain	Travel
Market	Auction	Auction	Market and Auction
Architecture	Client/Server	Client/Server	Heterogeneous Distributed
Play the Game	As travel agent	As retailer	Multiple roles
Message format	XML	XML	FIPA-ACL
Ontology used	No	No	Yes
Service	No	No	Yes
Server	Centralized	Centralized	Distributed
Game rule	Simple	Complex	Complex
Customer	Assigned to TA	Select TA based on proposal	Select TA based on proposal
Multiple supplier for same goods	No	Yes	Yes

FIGURE 21: COMPARISON OF TAGA AND TACS

From the figure above, we learn that TAGA uses the same game scenario as TAC Classic. Both TAGA and SCM enrich the game by adding more rules and flexibility into the system. Some of TAGA's unique features include using the open and distributed service network (Agentcities), using heterogeneous FIPA systems, participating in the game with multiple roles and using the semantic web technology to improve the automation and cooperation among agents.

An important difference between TAGA and the TACs is that TAGA system is designed as a framework for E-Commerce like applications. The framework provides multiple functional levels to support the agent interaction: agent communication level (semantic web), service level (web services and agent services) and system level (agentcities). The TAC-like travel agent game is implemented as a TAGA application based on the TAGA framework.

Another advantage of using TAGA is its stability. TAC is running in client/server model. The game participant is in trouble when there is network interruption. This happens many times during the TAC seeding rounds, even having to cancel some final round games in the first and second TAC competitions. TAGA is much more robust and stable. It has been running continuous in Agentcities for more than 18 months, with average network traffic about 50000 ACL messages/day. The game participants can join and quit the game anytime without the need of continuous TCP connection or waiting for their game round to start (as in TAC). In case there is network problem, the game participants don't lose money. The short network interruption period will have very small, if any, effect on their performances. The final ranking is decided by the auction strategy, algorithms and long-term performance.

4.4 Services

There are three basic roles in TAGA's service architecture: service provider, service requester and service DF. The service provider describes its service capability and service model in OWL-S language. The service is published when this service profile is supplied

to the service DF. The service requester sends the service requirement to the service DF. The service DF uses the knowledge on the published services to locate the services that satisfy the requirements.

4.4.1 Service Description

The FIPA agents are associated with one or more FIPA platforms, each of which offers a Directory Facility (DF) agent that handles the service registration, deregistration and matching. When registering a service in the DF, the agent provides only basic service information like the service type or the owner. However, more specific service information may be needed to provide more accurate matching. For example, a customer wants to book a three-star hotel, which is close to public transportation and offers valet parking. Such specific service matching can be achieved with the introducing of OWL-S. Every travel service provider in TAGA describes its service in the OWL-S language and publishes it as a service profile. This profile includes the declarative description of the service information in terms of processing model, thus providing the necessary descriptors to automate service invocation and service composition.

The travel service agent advertises the properties and capabilities through the profile description. The contact information is defined as follow:

```
<profileHierarchy:Booking rdf:ID="Profile_HotelBarcelona_Agent">
   <!-- reference to the process model specification -->
   <profile:has_process rdf:resource="&travel_process;#Travel_Process"/>
   <profile:serviceName>HotelBarcelona_Agent</profile:serviceName>
   <profile:contactInformation>
```

```
<profile:Actor rdf:ID="HotelBarcelona-Reservation">
  <profile:name>HotelBarcelona Reservation</profile:name>
  <profile:phone>34-93-3025858</profile:phone>
  <profile:physicalAddress>Caspe 1-13, Barcelona, Spain  </profile:physicalAddress>
  <profile:webURL> http://www.accomline.com </profile:webURL>
   <profile:qualityRating rdf:resource="&travel_process;#3-Star "/>
 </profile:Actor>
</profile:contactInformation>
```

The input and output properties associated with the process model enable the services to be used in automating web service invocation. For example, the booking process has three input properties including *Departure Date, Return Date and Buyer*. It also has one conditional output *ReservationNumber*. The following are two of the definitions:

```
<process:hasInput>
            <process:Input rdf:ID="ReturnDate">
            <process:parameterType rdf:resource="&xsd;#Date"/>
            </process:Input>
</process:hasInput>

<process:hasOutput>
        <process:ConditionalOutput rdf:ID="ReservationNumber">
          <process:coCondition rdf:resource="#RoomAvailable"/>
          <process:parameterType rdf:resource="#TravelItinerary"/>
        </process:ConditionalOutput>
 </process:hasOutput>
```

The effect properties enable the services to be used for automated composition and interoperation. The following example shows the effect of the travel reservation is that the traveler has a room at the hotel.

```
<process:hasEffect>
        <process:ConditionalEffect rdf:ID="RoomOccupied ">
                <process:ceCondition rdf:resource="#RoomAvailable"/>
                <process:ceEffect rdf:resource="OccupyRoomEffect#"/>
        </process:ConditionalEffect>
</process:hasEffect>
```

The process model of reservation service is a composite processes with an if-then-else construct which is composed of three atomic processes. The reservation is described in terms of two main steps: checking availability and reserving the travel resource. If there is resource available on a specified date, the resource is reserved and a unique reservation ID is sent to the buyer. Otherwise, the buyer receives the reject message.

```
<process:CompositeProcess rdf:ID="TravelProcess">
 <process:composedOf>
  <process:If-Then-Else>
        <process:ifCondition rdf:about="#CheckAvailability"/>
         <process:then rdf:about="#Booking"/>
        <process:else rdf:about="#BookingRejected"/>
  </process:If-Then-Else>
```

```
  </process:composedOf>
</process:CompositeProcess
```

Auction's process model is a composite process using a sequence of atom processes: CreateAuction, ReceiveBidding, AuctionEnd.

```
<process:CompositeProcess rdf:ID="Auction_Process">
  <process:composedOf>
    <process:Sequence>
      <process:components rdf:parseType="Collection">
                  <process:AtomicProcess rdf:resource="#CreateAuction"/>
                  <process:AtomicProcess rdf:resource="#ReceiveBidding"/>
                  <process:AtomicProcess rdf:resource="#AuctionEnd"/>
          </process:components>
    </process:Sequence>
  </process:composedOf>
```

The CreateAuction process accepts two inputs, *GameDay* and *Seller*, and generates the output *auctionID*. The ReceiveBidding process receives the bidding price from the bidder and returns the current high (winning) bidder. The AuctionEnd process generates a unique reservation ID and has the effect that the travel resource is occupied.

4.4.2 Service Publishing

A fundamental step toward the web of services is publishing the services to the Internet. The FIPA system provides a service-directory service to the agents. The agent who wishes to publicize its services to other agents finds an appropriate service-directory facility in the agent platform and registers its service description as a service-directory-entry. The entry is the service description consisting of a list of key-value pairs containing the service-name, service-type, service-locator and a set of optional service-attributes. The service-name is a globally unique name for the service. The service-type is the categorized type of the service. The service-locator, a typed structure composed of service signature and service address, is the interface to access the service. Additional service attributes may include other descriptive properties of the service, such as the cost associated with using the service or restrictions on using the service.

Additional functions provided by the service-directory service include de-registration and modification. The de-registration function has the consequence of removing the service information relating to an agent. The modification function enables the agent to update its service description at any time and for any reason.

The service-directory service is running inside each agent platform. To improve the service accessibility across multiple platforms distributed around the world, Agentcities designs a Global Service Directory that is able to track any of the services available in the network. It scans the active agent platforms and collects the service description details from every local DF. Two interfaces are provided to access the compiled result: a web page for human browsing and an agent interface for agent querying. The recorded service

description includes the service name, service type, supported languages, ontologies, protocols and other properties. One example is as follow:

Name	TAGA_Hotel_Barcelona
Type	Travel_Hotel
Protocol	FIPA-Request, FIPA-Query
Ontology	Travel, Auction
Language	OWL
Ownership	UMBC
Properties	(property :name damlsProfile :value http://taga.umbc.edu/hotel1.owl)

TAGA imports the OWL-S profile into the standard FIPA DF and Agentcities global DF by utilizing the provided "Properties" in the service description entities. This enables the other agents to acquire the needed service profile and feed appropriate input parameters using the service model described in the profile.

4.4.3 Service Discovery

FIPA's DF and Agentcities's global DF allow the discovery of the active services via querying the service descriptions. However, the DF supports only the query of the keyword in the service description entities and makes no use of the semantic information, therefore failing to match on the bases of the service capabilities and the service locations, consequently failing to address the problem of service discovery. The introducing of OWL-S into DF suggested in Section 4.4.2 enables other agents to retrieve the service

profiles. But the standard DF does not own the ability to query the detailed service model and service profile inside the OWL-S profile. [Dale 2003] suggests adding such ability to DF; however, we don't have control of the global Agentcities DF or others' FIPA agent platform. So, TAGA chooses not to embed the OWL-S profile service match function in the DF, but provides a standalone OWL-S service matchmaker.

Service matching is inherently a semantic problem because it has to abstract from the differences between the representations of the provided services and the requested services to recognize the semantic similarities between the two. For example, a service provider may advertise as "Travel Agent", while a service requester may need a "Travel Organizer". OWL-S fills the need for semantic representation of services through its tight connection with OWL. OWL allows the definition of relationships and supports further reasoning. So, it is possible to define that the "Travel Organizer" is same-instance-as the "Travel Agent". The task of matching becomes the task of reasoning over well-expressed advertisements and requests. TAGA uses F-OWL as the inference engine to manage the advertised service profiles and find the best-matching services.

The OWL-S based matching engine works as follow:

- Upon receiving a register request, the engine locates the OWL-S profile and loads the parsed triples as facts. Each service is an instance of service Class;
- Upon receiving a deregister request, the engine removes the related facts and discards all the tables (which may contain computed facts related to removed facts);

- Upon receiving a query, the engine returns a list of matching service instances.

4.4.4 Conclusions

This section covers TAGA's service architecture. The service description describes the service capability and service model. The service is published at the service DF. The service requester sends the service requirement to the service DF. The service DF uses the knowledge on the published services to locate the services that satisfy the requirements. There are two technologies involved: services provided by the software agent and semantic web services. This section suggests using OWL-S based matching engine and combining the best of two research communities: taking advantage of the DF support by importing the OWL-S representations into DF representations, while publishing semantically grounded descriptions of services that can be used to perform capability based search for services.

4.5 F-OWL Engine

One of the advantages of using OWL language in the TAGA system is that the knowledge encoded in the ACL message is machine understandable. Further complex reasoning technologies can be used to extract hidden knowledge. To achieve this goal, we designed and implemented the OWL reasoning engine F-OWL system. Built on First-Order Logic system XSB [Sagonas, 1994] and Frame-Logic system Flora-2 [Yang 2000],

F-OWL supports the representation and reasoning above the knowledge represented in RDF and OWL.

4.5.1 OWL Inference Engine

An inference engine is needed for the processing of the knowledge encoded in the semantic web language OWL. An OWL inference engine should have following features:

- Ontology consistency check: An OWL concept ontology (Tbox) imposes a set of restrictions on the model graph. The OWL inference Engine should check the syntax and usage of the OWL terms, ensures that the OWL instance (Abox) meets all the restrictions.
- Entailment: Entailment, including satisfiability and subsumption, is the key inference task for an OWL inference engine.
- Query: The OWL inference engine needs a powerful, yet easy-to-use, language to support knowledge query.
- Rule: Rules can be used to control the inference capability, describe business model, or express complex constrictions and relations not directly supported by OWL, like the most-referenced Uncle Class example. The OWL inference engine should provide a convenient interface to access the rules.
- Datatype: XML datatype can be used directly in OWL to support basic data types, e.g, integers, floats, strings and date. The new complex type can be created out of base types. The OWL inference Engine should be able to test the satisfiability of conjunctions of thus constructed datatypes.

The OWL language is rooted in Description Logic. Description Logic is a subset of First-Order Logic. Therefore, different approaches based on those logics have been used to design the OWL inference engine:

- Using Description Logic (DL) reasoner: The OWL language is rooted in description logic, it is not surprised that Description Logic reasoners are the most widely used tools for owl reasoning. The DL reasoner is used to specify the terminological hierarchy and support subsumption. It has the advantage of being decidable. Three well-known systems are FaCT, Racer and Pellet. They implement different types of description logic. Racer system implements SHIQ(D) using Tableaux algorithm. It is a complete OWL-DL reasoner, support both Tbox and Abox. FaCT system implements SHIQ, but only support Tbox. Pellet system implements SHIN(D). It includes a complete owl-lite consistency checker and support both Abox and Tbox queries.
- Using Full First Order Logic (FOL) theorem prover: The OWL ontology can be parsed into FOL syntax and use the existing FOL automated theorem prover to do the inference. The systems include Hoolet (using Vampire theorem prover) and Surnia (using Otter theorem prover). For example, the OWL ontology in the Hoolet system is translated to collection of axioms, and this collection of axioms is then given to the Vampire prover for reasoning purposes.
- Using other subset of FOL: Other fragment of FOL and general logic based inference engine can also be used to design the OWL inference engine. Horn Logic is most-widely used because of its simplicity and availability of tools,

including Jena, Jess, Triple and F-OWL (using XSB). Other logics, like higher-order logic in F-OWL (using Flora), can also be used. The advantage is obvious: lots of FOL system are available, and easy to understand, use and programming.

4.5.2 XSB and Flora

F-OWL system is based on First-Order Logic system XSB and F-Logic system Flora. This section gives a brief review of those two systems.

XSB is a Logic Programming system developed at State University of New York at Stony Brook. In addition to providing all the functionality of Prolog, XSB contains several features not usually found in Logic Programming systems, including SLG resolution and HiLog terms. They significantly extend XSB's capabilities beyond those typical Prolog systems.

To understand the SLG resolution, recall that Prolog is based on a depth-first search through trees that are built using program clause resolution. As such, Prolog is susceptible to getting lost as in the following example, where it may loop infinitely.

ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y).

ancestor(X,Y) :- parent(X,Y).

SLG evaluation, available in XSB, can correctly evaluate such a logic program. The user can declare that SLG resolution is to be used for a predicate by using table declarations:

:- table ancestor/2.

The query will terminate in XSB, since ancestor/2 is compiled as a tabled predicate. This power to solve recursive queries has proven very useful in a number of areas, including deductive databases, language processing, program analysis, model checking and diagnosis.

The second important extension in XSB is HiLog programming [Chen 1995], which allows a form of higher-order programming. For example, definition and execution of generic predicates like this generic transitive closure relationship are allowed:

closure(R)(X,Y) :- R(X,Y).

closure(R)(X,Y) :- R(X,Z), closure(R)(Z,Y).

Here the closure(R)/2 is a second-order predicate which, given any relationship R, returns its transitive closure relationship closure(R). XSB supports reading and writing HiLog terms, converting them to or from internal format as necessary. Special meta-logical standard predicates are also provided for inspection and handling of HiLog terms.

XSB supports the extensions of normal logic programs through preprocessing libraries. In particular, XSB supports a sophisticated object-oriented interface called Flora [Kifer, 1995]. Flora itself is a compiler that compiles from a dialect of Frame logic into XSB, taking advantage of the tabling, HiLog and well-founded semantics for negation features found in XSB. Flora-2 is implemented as a set of run-time libraries and a compiler that translates a united language of F-logic and HiLog into tabled Prolog code. HiLog is the default syntax that Flora-2 uses to represent function terms and predicates. Flora-2 is also a sophisticated application development platform. The programming language supported

by Flora-2 is a dialect of F-logic with numerous extensions, which include a natural way to do meta-programming in the style of HiLog and logical updates in the style of Transaction Logic.

4.5.3 F-OWL

The semantic of OWL refers to the fact that it is unambiguously about what the language means. An inference engine is responsible for assuring the accurate semantic of knowledge, finding out the possible inconsistency and deriving new information from the known information. To demonstrate the power of inference, we have this example: customer Joe wants to find an Italian restaurant; Restaurant goodPizza has cuisine type Pizza. They cannot match by the keywords. An ontology definition solves the problem: *Pizza rdfs:SubClassOf ItalianCuisine*. With an inference engine, customer Joe can successfully determine that restaurant goodPizza is what he is looking for.

F-OWL is an OWL inference engine that uses a F-Logic system to inference about OWL ontologies. F-OWL is accompanied with a simple OWL importer that reads the OWL ontology from a specified URI and extracts RDF triples out of the ontology. The extracted RDF triples are converted to the format appropriate for F-OWL's frame style and fed into the F-OWL engine. It then uses inference rules defined in Flora-2 language to check the consistency of the ontology and extract hidden knowledge via resolution.

The model theory is a formal theory that relates expressions to interpretation. The RDF model theory [Hayes 2002] [Hayes 2003] formalizes the notion of inference in RDF and

provides a basis for computing deductive closure of RDF graphs. The semantics of OWL [Peter 2003], an extension of RDF semantics, defines bindings and extensions of OWL interpretations that map variables to elements of the domain:

> The vocabulary *V* of the model is composed of a set of *URI*'s.
>
> *LV* is the set of *literal values* and *XL* is the mapping from the literals to *LV*.
>
> A *simple interpretation I* of a vocabulary *V* is defined by:
>
> 1). A non-empty set *IR* of resources, called the domain or universe of *I*.
>
> 2). A mapping *IS* from *V* into *IR*
>
> 3). A mapping *IEXT* from *IR* into the power set of *IR X (IR union LV)* i.e. the set of sets of pairs <*x,y*> with *x* in *IR* and *y* in *IR* or *LV*. This mapping defines the properties of the triples. *IEXT(x)* is a set of pairs which identify the arguments for which the property is true, i.e. a binary relational extension, called the *extension* of x.

Informally those definitions have the following meaning: every URI represents a resource that might be a page on the Internet but not necessarily: it may also be a physical object; A property is a relationship that is defined by an extension mapping from the property into a set; this set contains pairs where the first element of a pair represents the subject of a triple and the second element of a pair represents the object of a triple; With this system of extension, mapping the property can be part of its own extension without causing paradoxes.

Take the triple :*goodPizza :cuisineType :Pizza* from the pizza restaurant as an example. In the set of URI's there will be things like: *#goodPizza, #cuisineType, #pizza,*

#Restaurant, #italianCuisine, etc. These are part of the vocabulary *V*. The set IR of resources include resources on the Internet or elsewhere. For example *#Restaurant* might represent the set of all restaurants. The URI refers to a page on the Internet where the domain IR is defined. Then there is the mapping *IEXT* from *#cuisineType* to the set {*(#goodPizza, #Pizza),(#goodPizza, #ItalianCuisine)*} and the mapping *IS* from *V* to *IR*: *:goodPizza → #goodPizza, :cuisineTYpe → #cuisineType.*

A rule $A \rightarrow B$ is satisfied by an interpretation *I* iff every binding that satisfies the antecedent *A* also satisfies the consequent *B*. An ontology *O* is satisfied by an interpretation *I* iff the interpretation satisfies every rule and fact in the ontology. A model is satisfied if none of the statements included in the model contradict with each other. An ontology O is consistent iff it is satisfied by at least one interpretation. An ontology O_2 is entailed by an ontology O_1 iff every interpretation that satisfies O_1 also satisfies O_2.

The principal task in OWL reasoning is the ontology entailment. Lots of the OWL reasoning engines (for example Pellet and SHOQ) take an approach suggested by Ian Horrocks [Horrocks 2003]. By taking advantage of the similarity between the OWL and the Description Logic, the OWL entailment can be reduced to knowledge base satisfiability. As a result, a list of existing mature DL reasoning engines (like SHOIN(D), SHIF(D)[Horrocks 2003] or Racer [Haarslev 2001]) can provide reasoning services for OWL. Ora Lassila suggested another approach called*"True RDF processor"* in [Lassila 2002] and implemented it in the Wilbur system [Lassila 2001]. The entailment is defined

via the generation of a deductive closure from an RDF graph composed of triples. The proving of entailment becomes the building and searching of closure graph.

With the support of forward/backward reasoning from XSB and frame logic from Flora, F-OWL takes the second approach to compute the deductive closure. The closure is a graph consisting of every triple *<subject, predicate, object>* that satisfies *{subject, object } ∈ IEXT(I(predicate))*. This is defined as:

<subject, predicate, object> ∈KB <==> {subject, object } ∈ IEXT(I(predicate)).

Where *KB* is the knowledge base, *I(x)* is the interpretation of a particular graph, and *IEXT(x)* is the binary relational extension of property as defined in [Hayes 2002].

4.5.4 Implementation

F-OWL is written in flora-2 language and includes the following rules:

- A set of rules that reason about the data model of RDF/RDF-S and OWL;
- A set of rules that map XML DataTypes into XSB terms;
- A set of rules that perform ontology consistency checking;
- A set of rules that "glue" together the upper Java API calls to the lower layer Flora-2/XSB rules.

F-OWL provides command line interface, GUI and Java API to fit different requirements. Using F-OWL to reason about the ontology consists of four steps:

- Loading application-related rules into the engine;

- Adding a new ontology into the engine: The triples *(subject, predicate, object)* are translated into 2-ply frame style: *subject(predicate, object)@model*;
- Querying the engine: The RDF and OWL rules are recursively applied to generate all legal triples. If the question is Boolean type, a TRUE answer is returned when an interpretation of the question is found. If the question includes variables, the variables will be replaced with values from the interpretation;
- Removing the ontology and triples if desired: This saves the memory usage. Otherwise, the XSB system saves the computed triples in an indexed table, which save significant time and space for future usage.

W3C published a set of OWL test cases as the proof of OWL implementability. The F-OWL is applied to the OWL test cases as follows:

1) Tests for incorrect use of OWL namespace: The OWL syntax checker in F-OWL validates the usage of OWL namespace and gives a warning when the word followed by owl namespace is not defined in the OWL language.

   For example, http://www.w3.org/2002/03owlt/I3.2/bad001.rdf is a NotOwlFeature test file. F-OWL loads it into the engine and parses it into 4 triples:

   - triple(arp58433ARP0,rdf_type,owl_Restriction),
   - triple(arp58433ARP0,owl_cardinalityQ,'1').
   - triple(arp58433ARP0,owl_onProperty,'http://www.w3.org/2002/03owlt/I3.2/b")
   - triple(arp58433ARP0,owl_hasClassQ,'http://www.w3.org/2002/03owlt/I3.2/b")

Then a query “inconsist (X)” is submitted to the F-OWL engine. All the legal OWL terms are defined in the F-OWL rules using the owlterm() function. For example: owlterm(owl_AnnotationProperty) and owlterm(owl_Class). Since owl_hasClassQ is not a legal owl term, the F-OWL engine returns “Inconsist: Found owl atom not defined in OWL Namespace” and removes all the triples related to the test file.

2) Non-Entailment tests: The premise ontology is loaded into the engine first. The non-conclusion ontology is parsed into triples and queried as a conjunction of triples. The engine returns FALSE when failing to find the interpretation that satisfies the graph composed by the triples conjunction;

3) Entailment tests: The premise ontology is loaded into the engine first. The conclusion ontology is parsed into triples and queried as a conjunction of triples. The engine returns TRUE when finding one interpretation that satisfies the graph composed by the triples conjunction;

For example, the PositiveEntailment test disjointWith-001 assures that the disjoint classes have different members. The premises001.rdf file is loaded into the F-OWL engine and parsed into following 7 triples:

- triple('http://www.w3.org/2002/03owlt/disjointWith/premises001#A', rdf_type,owl_Class).
- triple('http://www.w3.org/2002/03owlt/disjointWith/premises001#B', rdf_type,owl_Class).

- triple('http://www.w3.org/2002/03owlt/disjointWith/premises001#A', owl_disjointWith, 'http://www.w3.org/2002/03owlt/disjointWith/premises001#B').
- triple('http://www.w3.org/2002/03owlt/disjointWith/premises001#a', rdf_type,'http://www.w3.org/2002/03owlt/disjointWith/premises001#A').
- triple('http://www.w3.org/2002/03owlt/disjointWith/premises001#a',rdf_type, owl_Thing).
- triple('http://www.w3.org/2002/03owlt/disjointWith/premises001#b',rdf_type, 'http://www.w3.org/2002/03owlt/disjointWith/premises001#B').
- triple('http://www.w3.org/2002/03owlt/disjointWith/premises001#b',rdf_type, owl_Thing).

Then the conclusion file conclusions001.rdf is also parsed and converted into a prolog style query:

```
myquery55114(fowl):-
    fowl_triple(http://www.w3.org/2002/03owlt/disjointWith/premises001#a,
    rdf_type,owl_Thing),
    fowl_triple(http://www.w3.org/2002/03owlt/disjointWith/premises001#b,
    rdf_type,owl_Thing)
    fowl_triple(http://www.w3.org/2002/03owlt/disjointWith/premises001#a,
    owl_differentFrom,
    http://www.w3.org/2002/03owlt/disjointWith/premises001#b).
```

In this example, the query is success, so the entailment is positive.

4) OWL for OWL tests: The tests are intended to illustrate how OWL Full can be used to describe its own properties and classes. The engine is queried against the conclusion ontology and return TRUE;
5) Inconsistency tests: The consistency checking discovers the inconsistency in the ontology and reports a warning;

For example, the test Nothing-001 asserts something of type owl:Nothing, however that is the empty class.

The F-OWL engine loads the file inconsistent001.rdf and parses it into one triple:

triple(arp75398ARP1832,rdf_type,owl_Nothing).

Since neither class nor class can be type of owl:Nothing in the OWL, we have the following inconsistency checking rules:

inconsist('Inconsist: Class rdf_type Owl_Nothing Exist!') :-

A[owl_equivalentClass->>owl_Nothing],_X[rdf_type->>A],!.

The inconsistency checking process finds the inconsistency and prints the error message " Inconsist: Class rdf_type Owl_Nothing Exist!".

6) Consistency: Every fact and rule in the ontology is checked using consistency checking. The ontology is announced consistent when no inconsistency is found;

7) Import level tests: After loading an ontology file, F-OWL locates and loads all ontology files defined in the file's *owl:imports* section;

8) Import entailment tests: F-OWL handles the imports closure by keeping a record of all loaded ontology files. The ontology is skipped if it is already loaded;

For example, the imports-003 ImportEntailment test is a document that imports another document which in turn imports a third document, then it entails anything which is entailed by the conjunction of the statements from the three documents. That means that the import is transitive.

The F-OWL engine loads the premises003.rdf file and parses it into 2 triples:

- triple('http://www.w3.org/2002/03owlt/imports/premises003',owl_imports,'http://www.w3.org/2002/03owlt/imports/support003-A').
- triple('http://example.org/data#Socrates',rdf_type,'http://www.w3.org/2002/03owlt/imports/support003-A#Man').

The first triple tells the F-OWL engine to load the support003-A.rdf file, which includes another 5 triples:

- triple('http://www.w3.org/2002/03owlt/imports/support003A',rdf_type, owl_Ontology).
- triple('http://www.w3.org/2002/03owlt/imports/support003-A', owl_imports, 'http://www.w3.org/2002/03owlt/imports/support003-B').
- triple('http://www.w3.org/2002/03owlt/imports/support003-A#Man',rdf_type, owl_Class).
- triple('http://www.w3.org/2002/03owlt/imports/support003-A#Man', rdfs_subClassOf, 'http://www.w3.org/2002/03owlt/imports/support003-A#Person').

- triple('http://www.w3.org/2002/03owlt/imports/support003-A#Person',rdf_type,owl_Class).

Those triples ask the engine to load support003-B.rdf files, which includes following 4 triples:

- triple('http://www.w3.org/2002/03owlt/imports/support003-B', rdf_type, owl_Ontology).
- triple('http://www.w3.org/2002/03owlt/imports/support003-A#Person', rdf_type, owl_Class).
- triple('http://www.w3.org/2002/03owlt/imports/support003-A#Person', rdfs_subClassOf,'http://www.w3.org/2002/03owlt/imports/support003-B#Mortal').
- triple('http://www.w3.org/2002/03owlt/imports/support003-B#Mortal',rdf_type,owl_Class).

After loading the three ontology files, the query " fowl_triple(http://example.org/data#Socrates,rdf_type,http://www.w3.org/2002/03owlt/imports/support003-B#Mortal) " is submitted and queried in the F-OWL engine.

F-OWL's test result is published in the W3C OWL test results document. Following table shows the result of eight types of OWL tests. We compare F-OWL with other three OWL inference engines. FaCT is a SHIQ DL reasoner written in Lisp. Pellet targets SHION(D)

DL, implementing the Tableau algorithm in Java language. Hoolet uses full FOL theorem prover Vampire system.

	F-OWL	FaCT	Pellet	Hoolet
Use of OWL Namespace	100%	0	100%	0
Entailment test (Lite)	65%	4%	82%	82%
Entailment test (DL)	6%	10%	89%	62%
Entailment (Full)	48%	0	82%	0
Non-Entailment test (Lite)	100%	0	62%	0
Non-Entailment test (DL)	83%	33%	83%	0
Non-Entailment test (Full)	100%	0	57%	0
OWL for OWL	100%	0	0	0
Consistency test (Lite)	64%	40%	100%	76%
Consistency test (DL)	44%	44%	92%	81%
Consistency test (Full)	87%	0	79%	0
Inconsistency test (Lite)	6%	82%	100%	93%
Import Entailment test (lite and Full)	100%	0	100%	0
Import level test (Lite, DL and Full)	85%	0	100%	0

FIGURE 22: COMPARISON OF OWL TEST CASE RESULT

F-Owl system performs well in non-entailment test, however in entailment-DL test, it doesn't perform as well as the DL reasoner Pellet and FaCT. That is because the DL reasoner's decidability helps in the DL type testing. F-OWL fails some of the DL tests after using up the memory in looping or timeout. F-OWL performs well in consistency tests, because the consistency tests are expressed with a predicate file and a result file, which fits well with Horn logic and FOL. The inconsistency test needs a number of rules to check the consistency. The F-OWL's test result in this category, being poor because it has no enough consistency rules, should be improved by adding more such rules. FaCT system doesn't perform well in lots of tests because it only supports Tbox reasoning.

As shown in the result table, F-OWL demonstrates that it has employed a complete strategy and returns a definitive result in many situations

4.5.5 Using F-OWL in TAGA

When an agent receives an incoming ACL message, it computes the meaning of the message from the ACL semantics, the protocols in effect, the content language and the conversational context. The agent's subsequent behavior, both internal (e.g., updating its knowledge base) and external (e.g., generating a response) depends on the correct interpretation of the message's meaning. Thus, a sound and, if possible, complete understanding the *semantics* of the key communication components (ACL, protocol, ontologies, content language, context) is extremely important. In TAGA, the service providers are independent and autonomous entities, which making it difficult to enforce a design decision that all use exactly the same ontology or protocol. For example, the Delta Airline service agent may have its own view of travel business and uses class and property terms that extend an ontology used in the industry. This situation parallels that for the semantic web as a whole – some amount of diversity is inevitable and must be panned for lest our systems become impossibly brittle.

Many of the agents implemented in TAGA system use F-OWL to represent and reason about the message content presented in RDF or OWL. Upon receiving an ACL message with content in RDF or OWL, a TAGA agent parses the content into triples, which are then loaded into the FOWL engine for processing.

The message's meaning (communicative act, protocol, content language, ontologies and context) all play a part in the interpretation. For example, when an agent receives a query message that uses the query protocol, the agent searches its knowledge base for matching

answers and returns an appropriate inform message. TAGA uses multiple models to reflect the multiple namespaces and ontologies used in the system. The agent treats each ontology as an independent model in the F-OWL engine.

F-OWL has many usages in TAGA:

- As knowledge base: upon receiving an ACL message with content encoded in OWL, the TAGA agent parses the content into triples and feeds them into the F-OWL engine. The information can be easily retrieved by submitting queries in various query languages;
- As reasoning engine: The agent can answer more questions with the help of F-OWL engine, for example, the restaurant can answer the question "what is the average price of a starter" after it understands that "starter" is *sameAs* "appetizer";
- Matchmaker: F-OWL manages the service profiles and tries to find the best match based on description in the service request;
- Agent interaction coordinator: the interaction protocol can be encoded into an ontology file using OWL language. F-OWL will advise the agents what to respond based on received messages and context.

4.5.6 Conclusion

This section describes the design and implementation of F-OWL, an inference engine for OWL language. F-OWL uses a Frame-based System to reason with OWL ontologies. F-OWL supports consistency check, extracts hidden knowledge via resolution and supports further complex reasoning by importing rules. By using it in TAGA user case, we find

that F-OWL is a full functional inference engine and easy to use with the support of multiple query languages and rule languages.

There have been lots of works on the OWL inference engine, from semantic web research community and description logic community. The following table compares F-OWL with some of them:

	F-OWL	Racer	FaCT	Pellet	Hoolet	Surnia	Triple
Logic	Horn, Frame, HigherOrder	Description Logic	DL	DL	Full FOL	Full FOL	Horn Logic
Support	OWL-Full	OWL-DL	OWL-DL	OWL-DL	OWL-DL	OWL-Full	RDF
Based on	XSB/Flora	Lisp	Lisp	Java	Vampire	Otter	XSB
XML Datatype	Yes	Yes	No	Yes	No	No	No
Decidable	No	Yes	Yes	Yes	No	No	Yes
Complete consistency checker	No	Yes (OWL-Lite)	Yes	Yes(OWL-Lite)	No	No	No
Interface	Java, GUI, Command Line	DIG, Java, GUI	DIG, Command Line	DIG, Java	Java	Python	Java
Query	Frame style, RDQL	Racer query language		RDQL			Horn logic style
Known Limitation	Not scale well		Not support Abox		Not scale well	Not scale well	Only support RDF

FIGURE 23: COMPARISON OF F-OWL AND OTHER OWL INFERENCE ENGINE

First thing we notice from Figure 23 is that the description logic based system can only support up to the OWL-DL, because their semantics and computation are controversial. OWL-Full is a full extension of RDF, which needs the supporting of terminological cycle. For example, a class in OWL-Full can also be an individual or property. The cyclic terminological definitions can be recognized and understood in horn logic or frame logic system.

The figure shows that only three DL-based owl inference engines, which all based Tableau algorithm [Baader 2000], are decidable and support complete consistency checking (at least in OWL-Lite). However, [Balaban 1993] argues that DL only forms a subset of F-Logic. The three kinds of formulae in the description logic can be transformed into first class objects and n-ary relationships. F-Logic is able to provide a full account for DL without losing any semantics and descriptive nature. We understand that current F-OWL approach is neither decidable nor complete. However, a complete F-Logic based OWL-DL reasoner is feasible.

The figure also shows that F-OWL system doesn't scale well when dealing with large dataset, because of incomplete reasoner. Actually, none of the OWL inference engines listed here scales well when dealing with the OWL test case wine ontology (with thousands of Class and Property). Further research is needed to improve the performance and desirability.

Comparing with other OWL inference engines, F-OWL has following unique features:

- Table: The table in Flora system can greatly improve the performance and desirability;
- Multiple logics: F-OWL supports Horn Logic, Frame logic and Higher-Order logic. Working together, they improve the performance and enhance the capabilities. For example, the F-logic supports non-monotonic (default) reasoning. Another example is Higher-Order logic. The semantic of Higher-Order logic may

not necessary very useful, its Higher-Order syntax (allowing variable appears as predicate) simplifies the expression of the statements.

- Practical: The aim of F-OWL system is to be a practical OWL reasoner, not necessary a complete OWL reasoner. So F-OWL system provides various interface to access the engine and supports multiple query and rule languages.

In the open web environment, it is generally assumed that the data are not complete and not all facts are known. We will research how this fact affects the implementation of inference engine. In the semantic web an inference engine may not necessarily serve to generate proofs but should be able to check proofs. We will work on using F-OWL to resolve trust and proof in semantic web.

CHAPTER 5 CONCLUSION

5.1 Review

This thesis presents a vision of agents working on the semantic web. Research involves the fields of web services, semantic web, and multi-agent systems. One of the challenges of this thesis is to design the service agent for the semantic web. The service agent will deal with problems of query, ontology mapping and OWL reasoning. The service agents need to supply other agents with data and meaning of the data, help other agents to understand the semantic web content, automate web services and serve humans better. We suggest using OWL language as the agent language for agent interaction and message passing. Each agent or web service site on the semantic web may define and use its own ontologies. Compared with the FIPA Content language like XML, SL or RDF, OWL has improved support in the ontologies and ontology mapping. The TAGA system is developed as the proof of concepts. Running on the open Agentcities platform, multiple agents with different market roles in TAGA cooperate or compete with each other to simulate the real-world travel market. We use F-OWL as the inference engine. It is a Flora-2 program that interprets RDF and OWL represented as a collection of RDF triples.

The common adoption of following technologies helps the TAGA system to achieve meaningful interaction:

- HTTP and SOAP as a message-passing protocol guarantee that messages can be exchanged by interacting parties;
- OWL-S and WSDL as a service definition language guarantees that the abstract and concrete syntax of the messages exchanged can be validated and understood.
- OWL as content language guarantees that the semantics of the messages exchanged is explicit and unambiguous between the sender and receiver. The message supports further inferences to match information objects and extract hidden information;
- FIPA agent technologies ensure the agents realize the behavior they are expected to realize.

TAGA demonstrates how the agents and the semantic web work together and answers the following research questions:

❖ What are the agent's roles on the semantic web?

The thesis lists the unique characteristics of the agents that make it a perfect fit for the semantic web environment: autonomous, cooperative, adaptive and social. An agent is capable of assisting human users, controlling agent interaction, controlling agent communication, finding interested services and planning a task.

❖ How to use the semantic web technology in the agent?

The thesis suggests using OWL as the agent communication content language. The thesis also suggests the solution about ontology mapping and sharing to support multiple ontologies in the agent system,

- How does the agent work on the semantic web?

The agent serves as the service agent and runs along with the web server. It is able to access the local ontologies and sensitive data. The thesis also suggests the interface between the agent and web services programs.

5.2 Discussion

5.2.1 Ontology

Ontology plays a prominent role on the semantic web. The semantic web is built on the assumption that everyone can abstract and publish their view as ontology files on the web. The semantic web languags like OWL provide a means to encode the ontology and, more importantly, provide a way to connect multiple ontologies to support ontology sharing and mapping.

Communication benefits when people are talking the same language. This is also true on the semantic web. When sharing the same ontology, the agents on the semantic web can discover the knowledge easier and faster. The future “common” ontology will be the one who is most widely accepted. Currently, the WordNet and IEEE SUMO ontology have many supporters, while CYC, as a traditional KR system, also draws lots of attention.

Many works have addressed automatic ontology mapping. The Anchor-PROMPT system [Noy 2001] explores a heuristic approach. [Peng 2002] uses Bayes Network. Glue system [Doan 2002] uses machine-learning techniques to find the mappings. There is limited work on ontology mapping in the context of the agent because the traditional agent system was designed with common ontology. The Agentcities ontology working group has started addressing some of the potential problems and user cases in the environment where the agent uses multiple ontologies.

5.2.2 Inference Engine

In the open web environment, it is generally assumed that the data is not complete and not all facts are known. When a query cannot be answered, the system should respond: "I don't know" rather than "No". This fact has some effects on the implementation of the inference engine. As to the OWL inference engine, it means that it is impossible to design a complete OWL-Full inference engine.

Another important aspect for the semantic web is inconsistency. In a stand-alone system inconsistencies are dangerous but can be controlled to a certain degree. However, controlling the inconsistencies on the semantic web is a lot more difficult. During the communication, unknown ontology definition origins from other agents may be asserted. Therefore special mechanisms are needed to deal with inconsistent and contradictory information on the semantic web. There are two steps: detecting the inconsistency and resolving the inconsistency.

The detection of the inconsistency is based on the declaration of inconsistency in the inference engine. The restriction, which imposes the possible values and relationship that the ontology elements can have, leads to the inconsistency. For example, the *owl:equivalentClass*: imposes a restriction on the resource which the subject is same class as. The *owl:disjointWith* imposes a restriction on the resource which the subject is different from. The triples *(a owl:equivalentClass b)* and *(a owl:disjointWith b)* is not lead directly to an inconsistency until applying the detecting rule: *(A owl:equivalentClass B) & (A owl:disjointWith B)* ➔*inconsistency.*

When inconsistencies are detected, Namespaces can help trace the origin of the inconsistencies. John posted "all dogs are human" at a web site, while "all dogs are animal" appears in daml.org's ontology library. It is clear that the second assertion is more accurate. The web sites are identified and treated unequivocally on the semantic web. The inference engine contacts the trust system to evaluate the creditability of the namespaces. [Klyne 2002] and [Golbeck 2003] enlist lots of works and brilliant ideas about how to maintain the trust system on the semantic web. Once having the trust evaluation result, the agent can take three different actions: (a) accept the one suggested by the inference engine; (b) reject both, as none of them is trustable; (c) ask the human user to select.

On the semantic web, an inference engine may not necessarily serve to generate proofs but should be able to check proofs. A proof explainer converts the proofs inside the inference engine into more understandable/readable formats.

5.2.3 Web Services

The "web services" promoted by Microsoft, IBM and other industry companies share the same vision about the future of the web with the semantic web. The success of the semantic web and the agent may depend on how they cooperate and integrate with the web services technologies, which have intense support in the industry.

With regard to the web services, OWL-S provides a semantically higher level and richer way to describe the services than what appears in either WSDL or in the descriptions in a UDDI registry. The WSDL interface descriptions correspond to a description of the message needed to invoke a service that is conceptually closer to the ServiceGrounding in OWL-S. The ServiceProfile of OWL-S provides a semantically meaningful description of services, as compared with that put forth by the data structures in the UDDI registry. However, OWL-S has no means to express the binding information that WSDL captures. Likewise, WSDL is unable to express the semantics. OWL-S/WSDL grounding defined by OWL-S group uses OWL classes as the abstract types of message parts declared in WSDL, and relies on WSDL binding constructs to specify the formatting of the messages. The descriptions in WSDL are augmented with OWL-S to describe business processes over a number of web services. The OWL-S is also related to the efforts of BPEL4WS in

the world of web services since the use of component services in BPEL4WS is described by WSDL type documents.

The agent-based services and web-based services can enjoy the following advantages if they can interoperate:

- If agents can access and use web services, then agent developers will profit from this new functionality. Agent applications can combine these web services and offer them to other agents as extended services.
- If web service clients and servers can access and use agent services, then agent developers will be able to offer the benefits of agent services to a web service environment. This can lead to the intelligent web services and will open the agent world and its potential to web services developers.

In web services, application designers compose their applications from services available on the web, as they compose them today from reusable Class on a user's machine. The WSDL file stores the description of what the service does and how to access it. The web services seamlessly connect the application programs (such as a .NET object) to incoming HTTP GET/POST request or SOAP request sent by web services clients.

In this thesis, we suggest having a wrapper for the web service program. This is an agent that reads the WSDL/OWL-S file and behaves as described. Agent-based services are advertised with the Directory Facilitator (DF), a mandatory agent on every FIPA-compliant agent platform providing the service registry functions. By using the OWL as the content language, the content with ontological support can be reasoned.

As to how web service program makes use of the agent services, the Agentcities service working group [Agentcity 2003] suggests building a WS-Agent service Gateway. The UDDI is mapped to the FIPA Directory Facilitator (DF). A FIPA ACL is mapped into a SOAP method invocation. The gateway queries known FIPA DF and produces the UDDI registry descriptions. When a SOAP method invocation from web service clients arrives, the gateway translates the data into the FIPA ACL inside a REQUEST message. The agent performs the necessary agent services and returns an ACL message, which is translated into the SOAP message and sent back.

Some problems remain unsolved. The first problem is that agent and web service are using different models: ACL message communication is performed in an asynchronous fashion, so needs to keep track of the agent conversation. The second problem is that within FIPA, it is an agent that offers the agent-based service. At present, the scope of Service Description registration is on the agent's home platform and on any other federated FIPA-compliant platform. This is a more restricted forum than that proposed by either the web services or the semantic web (OWL-S) visions. However, it is recognized that early users of web services may be the Intranets within organizational units. This is consistent with the scope of agent-based services on agent platforms.

5.3 Contributions

This thesis presents a vision of software agents working in the service-based semantic web environment. The TAGA system is a framework extending and enhancing the TAC

system to work in an open multi-agent systems environment of FIPA compliant systems. The challenges come from three main areas:

- Heterogeneous: the software agents, semantic web and web services are three independent developed research communities. There have different working models and standards;
- Communication: the software agents, semantic web and web services need to discover and understand each other;
- Security/Authorization: working in the unbound Internet sparks more concerns about security and trust.

We see following contributions in our work. First, this thesis presents an enhanced semantic web vision involving software agents, semantic web and web services. Consumers of the semantic information are the software agents. The agents, running in an open environment, use the ontology-based semantic web to promote agent communication and interaction. Both human and software agents can directly access the semantically annotated web pages through web interface. An agent communicates with other agents by exchanging messages encoded in a semantically rich agent communication language. Both web services and agent services are described in a semantically rich language to improve their interoperability. The personal agents and service agents work together to understand the semantic web content, automate web services and better serve humans.

Second, the TAGA system attests the semantic web vision and provides an environment to explore aspects of multi-agent system technology based on the mature and published FIPA standards. Research on multi-agent system technology is best done within a rich yet easily understood problem domain. We have found that the travel agent scenario as originally put forth by TAC provides both the richness as well as accessibility, especially when opened up to be peer-to-peer. We are using TAGA as a test-bed for research on the use of the semantic web languages (e.g., RDF and OWL) as content languages and as service description languages. Future work is planned to add more sophisticated negotiation and ontology mapping to our TAGA environment.

Third, we hope that TAGA can serve as an interesting framework and test-bed for experiments with automated markets and trading. By adding autonomous service provider agents (e.g., hotels), one can experiment with a dynamic market with both "shopbots" and "pricebots" [Greenwald, 1999] or investigate the role of intermediation in the form of agents performing a wholesale function.

Fourth, we hope that others will find TAGA useful as a test, demonstration and teaching environment, both in technology classes focused on multi-agent systems, FIPA standards or the semantic web and in business or e-commerce classes focused on automating commerce and trading, auctions or agent-based simulations.

5.4 Future Directions

Lots of improvement is needed to enhance the functions of TAGA system in the future. Many important and interest research areas are not covered by this thesis. The ontology sharing and mapping are very important for the semantic web itself, as well as for the agents to work together. The web is an unbounded environment, which means an agent has only partial knowledge at any time. The same or related information appears on the web in multiple places, often in different forms, which suggests agents need to deal with knowledge dependence and outdated knowledge. Web pages may come from an unknown website, so an agent must know who can trust and can evaluate the information provider and information itself. Communication between human users and agents, e.g. graphical user interfaces, is very important in attracting the potential "customer" to buy the idea of agent working on the semantic web environment. Co-ordination between the distributed services, i.e. making complex distributed decisions and planning in the agents, is not covered yet and should be considered in the future.

The Agentcities project is exploring the delivery and use of agent-based services in an open, dynamic and international setting. We are working to increase the integration of TAGA and emerging Agentcities components and infrastructure and will include agents running on handheld devices using the LEAP system [Bergenti, 2001].

APPENDIX A: FIPAOWL ONTOLOGY

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF
   xmlns="http://taga.umbc.edu/ontologies/fipaowl#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://taga.umbc.edu/ontologies/fipaowl">
  <owl:Ontology rdf:about="http://taga.umbc.edu/ontologies/fipaowl"/>
<!--
   OWL content language for FIPA acl message.
   Used in TAGA game (http://taga.umbc.edu)

   Author: Youyong Zou  yzou1@cs.umbc.edu
   Date:  Apr 30,2004
-->

  <owl:Class rdf:ID="Cancel">
   <rdfs:subClassOf>
     <owl:Class rdf:ID="ACLMessage"/>
   </rdfs:subClassOf>
```

```
</owl:Class>
<owl:Class rdf:ID="Inform">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="Services"/>
<owl:Class rdf:ID="Subscribe">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="Agree">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="Question"/>
<owl:Class rdf:ID="InteractionProtocol"/>
<owl:Class rdf:ID="Request-whenever">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="Request-when">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="Query-ref">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="Refuse">
```

```
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="Result"/>
<owl:Class rdf:ID="AgentPlatform">
  <owl:disjointWith>
    <owl:Class rdf:about="#FIPAAgent"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:ID="Query-if">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="Rule"/>
<owl:Class rdf:ID="FIPAAgent">
  <owl:disjointWith rdf:resource="#AgentPlatform"/>
</owl:Class>
<owl:Class rdf:ID="Effect"/>
<owl:Class rdf:ID="Request">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="Propose">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="Failure">
```

```
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="Action"/>
<owl:Class rdf:ID="Accept_Proposal">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="Proxy">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="Confirm">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="Propogate">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="CFP">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="Disconfirm">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:Class rdf:ID="Reject-proposal">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
```

```
</owl:Class>
<owl:Class rdf:ID="Answer"/>
<owl:Class rdf:ID="Not-understood">
  <rdfs:subClassOf rdf:resource="#ACLMessage"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="fromAction">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Effect"/>
        <owl:Class rdf:about="#Result"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="#Action"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="participant">
  <rdfs:domain rdf:resource="#InteractionProtocol"/>
  <rdfs:range rdf:resource="#FIPAAgent"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="sender">
  <rdfs:range rdf:resource="#FIPAAgent"/>
  <rdfs:domain rdf:resource="#ACLMessage"/>
```

```
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="target">
  <rdfs:range rdf:resource="#FIPAAgent"/>
  <rdfs:domain rdf:resource="#Action"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="receiver">
  <rdfs:domain rdf:resource="#ACLMessage"/>
  <rdfs:range rdf:resource="#FIPAAgent"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="oneEffect">
  <rdfs:domain rdf:resource="#Effect"/>
  <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="atPlatform">
  <rdfs:domain rdf:resource="#FIPAAgent"/>
  <rdfs:range rdf:resource="#AgentPlatform"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="KB_724709_Slot_9"/>
<owl:ObjectProperty rdf:ID="content">
  <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:domain rdf:resource="#ACLMessage"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="actor">
```

```
  <rdfs:range rdf:resource="#FIPAAgent"/>
  <rdfs:domain rdf:resource="#Action"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="output">
  <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:domain rdf:resource="#Result"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="serviceProvider">
  <rdfs:domain rdf:resource="#Services"/>
  <rdfs:range rdf:resource="#FIPAAgent"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#provideServices"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="in-reply-to">
  <rdfs:range rdf:resource="#ACLMessage"/>
  <rdfs:domain rdf:resource="#ACLMessage"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasServices">
  <rdfs:range rdf:resource="#Services"/>
  <rdfs:domain rdf:resource="#AgentPlatform"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ask">
```

<rdfs:domain rdf:resource="#Question"/>

<rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>

</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="provideServices">

<rdfs:domain rdf:resource="#FIPAAgent"/>

<rdfs:range rdf:resource="#Services"/>

<owl:inverseOf rdf:resource="#serviceProvider"/>

</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="ruleContent">

<rdfs:domain rdf:resource="#Rule"/>

<rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>

</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasQuestion">

<rdfs:domain rdf:resource="#Answer"/>

<rdfs:range rdf:resource="#Question"/>

</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="reply-to">

<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>

<rdfs:range rdf:resource="#FIPAAgent"/>

<rdfs:domain rdf:resource="#ACLMessage"/>

</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="protocolFlow">

<rdfs:range>

```
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <owl:Class rdf:about="#ACLMessage"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
  <rdfs:domain rdf:resource="#InteractionProtocol"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="done">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
  <rdfs:domain rdf:resource="#Result"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="serviceParameter">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Services"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="agentAddress">
  <rdfs:domain rdf:resource="#FIPAAgent"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="questionLanguage">
  <rdfs:domain rdf:resource="#Question"/>
```

```
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="serviceType">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Services"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hasBelief">
  <rdfs:domain rdf:resource="#FIPAAgent"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="serviceProtocol">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Services"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="performative">
  <rdfs:domain rdf:resource="#ACLMessage"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="questionFailure">
  <rdfs:domain rdf:resource="#Answer"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="KB_724709_Slot_30">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
```

```
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="reply-with">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#ACLMessage"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="encoding">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#ACLMessage"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="serviceName">
  <rdfs:domain rdf:resource="#Services"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="agentStatus">
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf rdf:parseType="Resource">
        <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >Waiting</rdf:first>
        <rdf:rest rdf:parseType="Resource">
          <rdf:rest rdf:parseType="Resource">
            <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >Active</rdf:first>
```

```
          <rdf:rest rdf:parseType="Resource">
            <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >Transit</rdf:first>
            <rdf:rest rdf:parseType="Resource">
              <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >Initialized</rdf:first>
              <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
            </rdf:rest>
          </rdf:rest>
        </rdf:rest>
        <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >Suspended</rdf:first>
      </rdf:rest>
    </owl:oneOf>
  </owl:DataRange>
 </rdfs:range>
 <rdfs:domain rdf:resource="#FIPAAgent"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="conversation-id">
 <rdfs:domain rdf:resource="#ACLMessage"/>
 <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="actionParameters">
```

```
  <rdfs:domain rdf:resource="#Action"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="failureReason">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Answer"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ruleLanguage">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Rule"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="language">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#ACLMessage"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="protocolName">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#InteractionProtocol"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="act">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Action"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="serviceOntology">
```

```
  <rdfs:domain rdf:resource="#Services"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hasDesire">
  <rdfs:domain rdf:resource="#FIPAAgent"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="platformName">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#AgentPlatform"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="answerNumber">
  <rdfs:domain rdf:resource="#Question"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ownership">
  <rdfs:domain rdf:resource="#Services"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="serviceProfile">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI"/>
  <rdfs:domain rdf:resource="#Services"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ontology">
```

```
    <rdfs:domain rdf:resource="#ACLMessage"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="reply-by">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
    <rdfs:domain rdf:resource="#ACLMessage"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasIntend">
    <rdfs:domain rdf:resource="#FIPAAgent"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="agentName">
    <rdfs:domain rdf:resource="#FIPAAgent"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 1.1, Build 135)  http://protege.stanford.edu -
->
```

APPENDIX B: TRAVEL ONTOLOGY

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF
```

```
  xmlns="http://taga.umbc.edu/ontologies/travel#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
xml:base="http://taga.umbc.edu/ontologies/travel">
<owl:Ontology rdf:about="">
  <owl:imports rdf:resource="http://www.domain2.com"/>
</owl:Ontology>
<owl:Class rdf:ID="Itinerary"/>
<owl:Class rdf:ID="Preference"/>
<owl:Class rdf:ID="HotelPreference">
  <rdfs:subClassOf rdf:resource="#Preference"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#AirlinePreference"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#EntertainmentPreference"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:ID="HotelReservation">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Reservation"/>
  </rdfs:subClassOf>
```

```
  <owl:disjointWith>
    <owl:Class rdf:about="#EntertainmentReservation"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#AirlineReservation"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:ID="Customer"/>
<owl:Class rdf:ID="ServiceProvider"/>
<owl:Class rdf:ID="AirlineReservation">
  <owl:disjointWith>
    <owl:Class rdf:about="#EntertainmentReservation"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#HotelReservation"/>
  <rdfs:subClassOf rdf:resource="#Reservation"/>
</owl:Class>
<owl:Class rdf:ID="AirlinePreference">
  <owl:disjointWith>
    <owl:Class rdf:about="#EntertainmentPreference"/>
  </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#Preference"/>
  <owl:disjointWith rdf:resource="#HotelPreference"/>
</owl:Class>
```

```
<owl:Class rdf:ID="EntertainmentPreference">
  <owl:disjointWith rdf:resource="#AirlinePreference"/>
  <owl:disjointWith rdf:resource="#HotelPreference"/>
  <rdfs:subClassOf rdf:resource="#Preference"/>
</owl:Class>
<owl:Class rdf:ID="EntertainmentReservation">
  <rdfs:subClassOf rdf:resource="#Reservation"/>
  <owl:disjointWith rdf:resource="#AirlineReservation"/>
  <owl:disjointWith rdf:resource="#HotelReservation"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="forCustomer">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#hasItinerary"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#Itinerary"/>
  <rdfs:range rdf:resource="#Customer"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasServiceProvider">
  <rdfs:range rdf:resource="#ServiceProvider"/>
  <rdfs:domain rdf:resource="#Reservation"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasReservation">
  <rdfs:domain rdf:resource="#Itinerary"/>
```

```
  <rdfs:range rdf:resource="#Reservation"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasPreference">
  <rdfs:domain rdf:resource="#Customer"/>
  <rdfs:range rdf:resource="#Preference"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="travelAgent">
  <rdfs:domain rdf:resource="#Itinerary"/>
  <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasItinerary">
  <rdfs:range rdf:resource="#Itinerary"/>
  <rdfs:domain rdf:resource="#Customer"/>
  <owl:inverseOf rdf:resource="#forCustomer"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="byCustomer">
  <rdfs:range rdf:resource="#Customer"/>
  <rdfs:domain rdf:resource="#Reservation"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="to">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Itinerary"/>
</owl:DatatypeProperty>
```

<owl:DatatypeProperty rdf:ID="travelPlanReady">

<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>

<rdfs:domain rdf:resource="#Customer"/>

</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="serviceProviderName">

<rdfs:domain rdf:resource="#ServiceProvider"/>

<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="custId">

<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

<rdfs:domain rdf:resource="#Customer"/>

</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="penaltyPrice">

<rdfs:domain rdf:resource="#Itinerary"/>

<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>

</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="returnDate">

<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>

<rdfs:domain rdf:resource="#Itinerary"/>

</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="reserveNumber">

<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

<rdfs:domain rdf:resource="#Reservation"/>

```
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="preferValue">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Preference"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="departureDate">
  <rdfs:domain rdf:resource="#Itinerary"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="serviceType">
  <rdfs:domain rdf:resource="#ServiceProvider"/>
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf rdf:parseType="Resource">
        <rdf:rest rdf:parseType="Resource">
          <rdf:rest rdf:parseType="Resource">
            <rdf:rest rdf:parseType="Resource">
              <rdf:rest rdf:parseType="Resource">
                <rdf:rest rdf:parseType="Resource">
                  <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                  >Others</rdf:first>
                  <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                </rdf:rest>
```

```
            <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >Transportation</rdf:first>
           </rdf:rest>
           <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
           >Accomodation</rdf:first>
          </rdf:rest>
          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >Opera</rdf:first>
         </rdf:rest>
         <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
         >Restaurant</rdf:first>
        </rdf:rest>
        <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >Cinema</rdf:first>
       </owl:oneOf>
    </owl:DataRange>
  </rdfs:range>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="offerPrice">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
  <rdfs:domain rdf:resource="#Itinerary"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="from">
```

```
    <rdfs:domain rdf:resource="#Itinerary"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="price">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
    <rdfs:domain rdf:resource="#Reservation"/>
  </owl:DatatypeProperty>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 1.1, Build 135)  http://protege.stanford.edu -
->
```

APPENDIX C: AUCTION ONTOLOGY

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF
    xmlns="http://taga.umbc.edu/ontologies/auction#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://taga.umbc.edu/ontologies/auction">
  <owl:Ontology rdf:about="http://taga.umbc.edu/ontologies/auction"/>
  <owl:Class rdf:ID="Bidding"/>
```

```
<owl:Class rdf:ID="Auction"/>
<owl:Class rdf:ID="AuctionItem"/>
<owl:Class rdf:ID="Participant"/>
<owl:ObjectProperty rdf:ID="seller">
  <rdfs:range rdf:resource="#Participant"/>
  <rdfs:domain rdf:resource="#Auction"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="status"/>
<owl:ObjectProperty rdf:ID="reserveResource">
  <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:domain rdf:resource="#Bidding"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="auctionHouse">
  <rdfs:domain rdf:resource="#Auction"/>
  <rdfs:range rdf:resource="#Participant"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasAuctionItem">
  <rdfs:range rdf:resource="#AuctionItem"/>
  <rdfs:domain rdf:resource="#Auction"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasBidding">
  <rdfs:domain rdf:resource="#Auction"/>
  <rdfs:range rdf:resource="#Bidding"/>
```

```
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="reservePrice">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain rdf:resource="#Auction"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hasStatus">
  <rdfs:domain rdf:resource="#Auction"/>
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf rdf:parseType="Resource">
        <rdf:rest rdf:parseType="Resource">
          <rdf:rest rdf:parseType="Resource">
            <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >Failure</rdf:first>
            <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
          </rdf:rest>
          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >Close</rdf:first>
        </rdf:rest>
        <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >Open</rdf:first>
      </owl:oneOf>
    </owl:DataRange>
```

```
  </rdfs:range>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="itemDescription">
  <rdfs:domain rdf:resource="#AuctionItem"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="suggestRetailPrice">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
  <rdfs:domain rdf:resource="#AuctionItem"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="itemName">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#AuctionItem"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="name">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Participant"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="itemQuantity">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain rdf:resource="#AuctionItem"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hasAuctionType">
```

```
<rdfs:range>
  <owl:DataRange>
    <owl:oneOf rdf:parseType="Resource">
      <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >English</rdf:first>
      <rdf:rest rdf:parseType="Resource">
        <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >Dutch</rdf:first>
        <rdf:rest rdf:parseType="Resource">
          <rdf:rest rdf:parseType="Resource">
            <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >Priceline</rdf:first>
            <rdf:rest rdf:parseType="Resource">
              <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
              <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >Hotwire</rdf:first>
            </rdf:rest>
          </rdf:rest>
          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >Double</rdf:first>
        </rdf:rest>
      </rdf:rest>
    </owl:oneOf>
```

```
    </owl:DataRange>
  </rdfs:range>
  <rdfs:domain rdf:resource="#Auction"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="itemQuality">
  <rdfs:domain rdf:resource="#AuctionItem"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="closeTime">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >when the auction will be closed</rdfs:comment>
  <rdfs:domain rdf:resource="#Auction"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>
<owl:FunctionalProperty rdf:ID="expire">
  <rdfs:domain rdf:resource="#Bidding"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="bidQuantity">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#Bidding"/>
```

```
 </owl:FunctionalProperty>
 <owl:FunctionalProperty rdf:ID="bidder">
  <rdfs:domain rdf:resource="#Bidding"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
 </owl:FunctionalProperty>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 1.1, Build 135)  http://protege.stanford.edu -
->
```

APPENDIX D: TAGA MESSAGE

This is a Proxy message sent by Customer Agent. The Bulletin Board Agent accepts the message and forwards it to all registered Travel Agents

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema#"
xmlns:owl="http://www.w3.org/2002/7/owl#"
xmlns:travel="http://taga.umbc.edu/travel#"
xmlns:tagaQL="http://taga.umbc.edu/tagaQL#"
```

xmlns:auction="http://taga.umbc.edu/auction#"

xmlns:fipaowl="http://taga.umbc.edu/fipaowl#">

<fipaowl:Action rdf:ID="proxy534">

<fipaowl:actor><fipaowl:FIPAAgent

rdf:about="http://taga.umbc.edu:8080/acc/bulletinboard">

<fipaowl:agentName>bulletinboard</fipaowl:agentName>

<fipaowl:agentMTP>http://taga.umbc.edu:8080/acc</fipaowl:agentMTP>

</fipaowl:FIPAAgent>

</fipaowl:actor>

<fipaowl:act>SendMessage</fipaowl:act>

<fipaowl:target><fipaowl:FIPAAgent>

<owl:Restriction>

<owl:onProperty rdf:resource="travel:serviceType"/>

<owl:hasValue>TravelAgent</owl:hasValue>

</owl:Restriction>

</fipaowl:FIPAAgent>

</fipaowl:target>

<fipaowl:target><fipaowl:ACLMessage>

<fipaowl:performative>CFP</fipaowl:performative>

<fipaowl:sender><fipaowl:FIPAAgent

rdf:about="http://taga.umbc.edu:8080/acc/bulletinboard">

<fipaowl:agentName>bulletinboard</fipaowl:agentName>

<fipaowl:agentMTP>http://taga.umbc.edu:8080/acc</fipaowl:agentMTP>

</fipaowl:FIPAAgent>

</fipaowl:sender>

<fipaowl:reply-to><fipaowl:FIPAAgent

rdf:about="http://taga.umbc.edu:8080/acc/CustomerAgent">

<fipaowl:agentName>CustomerAgent</fipaowl:agentName>

<fipaowl:agentMTP>http://taga.umbc.edu:8080/acc</fipaowl:agentMTP>

</fipaowl:FIPAAgent>

</fipaowl:reply-to>

<fipaowl:content><fipaowl:Action rdf:ID="newcustomer534">

<fipaowl:act>OrganizeTravel</fipaowl:act>

<fipaowl:target><travel:Customer rdf:ID="cust534">

<travel:budgetLimit>1599</travel:budgetLimit>

<travel:hasItinerary><travel:Itinerary rdf:ID="Itin534">

<travel:returnDate>20</travel:returnDate>

<travel:departureDate>10</travel:departureDate>

<travel:from>City_A</travel:from>

<travel:to>City_B</travel:to>

<travel:hasPreference><travel:HotelPreference rdf:ID="HotelPref534">

<travel:preferValue>4</travel:preferValue>

</travel:HotelPreference>

</travel:hasPreference>

<travel:hasPreference><travel:EntertainmentPreference rdf:ID="EnterPref534">

<travel:preferValue>3</travel:preferValue>

</travel:EntertainmentPreference>

</travel:hasPreference>

</travel:Itinerary>

</travel:hasItinerary>

<travel:custId>534</travel:custId>

</travel:Customer>

</fipaowl:target>

</fipaowl:Action>

</fipaowl:content>

</fipaowl:ACLMessage>

</fipaowl:target>

</fipaowl:Action>

</rdf:RDF>

This is a propose message from Travel Agent to Customer Agent.

<?xml version="1.0"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

xmlns:xsd="http://www.w3.org/2000/10/XMLSchema#"

xmlns:owl="http://www.w3.org/2002/7/owl#"

xmlns:travel="http://taga.umbc.edu/travel#"

xmlns:tagaQL="http://taga.umbc.edu/tagaQL#"

xmlns:auction="http://taga.umbc.edu/auction#"

xmlns:fipaowl="http://taga.umbc.edu/fipaowl#">

<fipaowl:Result rdf:ID="actionresult780">

<fipaowl:done>true</fipaowl:done>

<fipaowl:action><fipaowl:Action rdf:resource="newcustomer534"/>

</fipaowl:action>

<fipaowl:output><travel:Customer rdf:resource="cust534">

<travel:custId>534</travel:custId>

<travel:hasItinerary><travel:Itinerary rdf:ID="Itin1534tagaTA2457">

<travel:returnDate>20</travel:returnDate>

<travel:departureDate>10</travel:departureDate>

<travel:AirlineReservation>

<travel:hasServiceProvider>airlinewsa1@taga.agentcities.net</travel:hasServiceProvider

>

</travel:AirlineReservation>

<travel:hasReservation><travel:HotelReservation>

<travel:hasServiceProvider>HotelAuctionServer</travel:hasServiceProvider>

</travel:HotelReservation>

</travel:hasReservation>

<travel:offerPrice>1083</travel:offerPrice>

<travel:penaltyPrice>2</travel:penaltyPrice>

<travel:travelAgent>tagaTA2</travel:travelAgent>

</travel:Itinerary>

</travel:hasItinerary>

</travel:Customer>

</fipaowl:output>

</fipaowl:Result>

</rdf:RDF>

APPENDIX E: DESIGN AN AGENT FOR TAGA

TAGA provides a Travel Agent Shell written in the April language. This shell provides a set of libraries to free the game players from the tedious communication processing and focus on the game strategies.

The OWL library includes multiple functions related to the OWL message:

- parseOWLFile(owlfile, defaultNS): load an OWL file and parse it into a list of triples
- CreateOWL(content): create a new OWL ACL message with content inside
- parseOWLString(owlStream, defaultNS): parse an OWL stream data into a list of triples.
- OneClass(): retrieve the specified Class instance information

- OneProperty(): retrieve the specified property information
- AidtoOwl(): process the agent AID encoded in OWL format

The service library provides the functions to access the service providers:

- SearchForAgents(): search for service providers adhere to specified constraints
- ParseProfile: process an OWL-S/WSDL service description profile
- CheckHotelPrice: send query message to all known hotel service providers and return the current prices
- CheckAirlinePrice: send query message to all known airline service providers and return the current prices
- SubmitPriceline: submit a new bid to the airline auction server
- CheckHotelAuctionPrice: send query message to hotel auction server and return the current price

The database library includes the following functions:

- TagaProfile(): load taga profile from a file or URI
- TagaDB(): connect to the MySQL database running at TAGA Server and updating the game data

The agent library includes the functions related to the agent platform and communication: registering at DF and AMS; registering at the BBA;

- searchForAgents(): searching the service providers based on the service type
- createOWL(): wrapping the OWL content inside an ACL message

- tagaHttpMtp(): send out the message via the HTTP/SOAP interface

APPENDIX F: F-OWL USER GUIDE

F-OWL provides a command line interface. It supports following commands:

- rule <filename>: load a XSB/Flora rule file
- load <filename> [>file.P]: load RDF/OWL/NTriple
- file into KB [or save triples in file]
- unload <filename>: remove the loaded triples from KB
- query <filename>: Query RDF/NTriple File
- command <xsb command>: run xsb command F-OWL
- xsbquery <xsb query>: send xsb command to XSB and print result
- pp className: pretty print Class and its property
- goflora: change F-OWL to Flora2, use flEnd to return F-OWL.
- reset: abolish all tables
- rdftest testNumber: test the RDF test case
- owltest testNumber: test the OWL test case
- namespace [namespacePair]: namespace management
- help: show the help message
- quit: quit F-OWL

BIBLIOGRAPHY

[Agentcities 2003] Agentcities Service group: "Integrating Web Services into Agentcities Recommendation", Nov 2003.

[Alexaki 2001] Sofia Alexaki, Vassilis Christophides, Gregory Karvounarakis, Dimitris Plexousakis, Karsten Tolle: "The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases", Semantic Web Working Symposium, Stanford CA, May 2001.

[Anthony 2001] P. Anthony, W. Hall, V.D. Dang, and N. Jennings: "Autonomous Agents for Participating in Multiple Online Auctions", Proceeding of the IJCAI Workshop on EBusiness and the Intelligent Web, Seattle WA, July 2001.

[Baader 2000] Franz Baader and Ulrike Sattler: "An Overview of Tableau Algorithms for Description Logics", Proceeding of Tableau 2000.

[Balaban 1993] Mira Balaban: "The F-Logic Approach for Description Languages", Ben-Gurion University of Negev Technical Report FC-93-02, 1993.

[Bellifemine 2001] F. Bellifemine, A. Poggi, G. Rimassa: "Developing Multi Agent Systems with a FIPA-Compliant Agent Framework", Software - Practice And Experience, 2001 N31, page 103-128, 2001.

[Bergenti 2001] F. Bergenti and A. Poggi: "LEAP: A FIPA Platform for Handheld and Mobile Devices", ATAL, 2001.

[Berners-Lee 2001] Tim Berners-Lee, James Hendler and Ora Lassila: "The semantic web", Scientific America May 2001.

[Bothelo 2002] L. Bothelo, S. Willmott, T. Zhang, J. Dale: "A Review of Content Languages Suitable for Agent-Agent Communication", EPFL I&C Technical Report #200233, 2002.

[Broekstra 2000] Jeen Broekstra, Michel Klein, Stefan Decker, Dieter Fensel, Ian Horrocks: "Adding Formal Semantics to the Web: Building on top of RDF Schema", Proceedings of the Workshop ECDL 2000 Workshop on Semantic Web, 2000.

[Buckle 2000] Phil Buckle: "FIPA and FIPA-OS Overview", Joint Holonic Manufacturing Systems and FIPA Workshop, London, September 2000.

[Buhler 2003] Paul Buhler and Jose M. Vidal: " Semantic Web Services as Agent Behaviors", Proceeding of Agentcities: Challenges in Open Agent Environments workshop, Springer 2003.

[Chen 1995] Weidong Chen, Michael Kifer and David Warren: "Logical Foundations of Object-Oriented and Frame-Based Languages", Journal of ACM, May 1995.

[Cost 2002] Scott Cost, Tim Finin, Anupam Joshi, Yun Peng, Charles Nicholas, Ian Soboroff, Harry Chen, Lalana Kagal, Filip Perich, Youyong Zou, Sovrin Tolia: "ITTALK: A Case Study on the semantic web and DAML+OIL", IEEE Intelligent Systems January/February 2002.

[Csirik 2001] János A. Csirik, Michael L. Littman, Satinder Singh, Peter Stone: " FAucS: An FCC Spectrum Auction Simulator for Autonomous Bidding Agents", Electronic Commerce: Proceedings of the Second International Workshop, Page 139-151, Springer, 2001.

[Dale 2002] J. Dale, S. Willmot, and B.Burg: "Agentcities: Challenges and Deployment of Next-Generation Service Environments", Proceeding of Pacific Rim Intelligent Multi-Agent Systems, Tokyo Japan, August 2002.

[Dale 2003] Jonathan Dale, Luigi Ceccaroni, Youyong Zou, Avigail Agam: "Implementing Agent-Based Web Services", AAMAS 2003 Workshop on Challenges in Open Agent Environments, Melbourne Australia, July 2003.

[DAML-S 2002] The DAML Services Coalition (Anupriya Ankolenkar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne and Katia Sycara): "DAML-S: Web Service Description for the Semantic Web", The First International Semantic Web Conference (ISWC), Sardinia Italy, June 2002.

[Dean 2002] Mike Dean and Guus Schreiber: "OWL Web Ontology Language 1.0 Reference", W3C Working Draft.

[Decker 1999] Stefan Decker, Michael Erdmann, Dieter Fensel, and Rudi Studer: "OnTobroker: Ontology Based Access to Distributed and Semi-Structured Information", Semantic Issues in Multimedia Systems, Page 351-369, Boston MA, 1999.

[Decker 2000] Stefan Decker, Dieter Fensel, Frank van Harmelen, Ian Horrocks, Sergey Melnik, Michel Klein and Jeen Broekstra: "Knowledge Representation on the Web", Proceedings of the International Workshop on Description Logics 2000.

[Denker 2001] Grit Denker, Jerry R. Hobbs, David Martin, Srini Narayanan, Richard Waldinger: "Accessing Information and Services on the DAML-Enabled Web", Semantic Web Working Symposium, Stanford CA, May 2001.

[Doan 2002] Anhai Doan , Jayant Madhavan , Pedro Domingos , Alon Halevy: “Learning to Map between Ontologies on the Semantic Web”, WWW 2002, Honolulu Hawaii, 2002.

[Eriksson 2002] Joakim Eriksson and Sverker Janson: “The Trading Agent Competition - TAC 2002”, ERCIM News, October 2002.

[Fensel 2000] D. Fensel, I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann, M. Klein: “OIL in a Nutshell”, EKAW 2000, October 2000.

[Finin, 1992] Tim Finin, Rich Fritzson, and Don McKay: “A Knowledge Query and Manipulation Language for Intelligent Agent Interoperability”, Fourth National Symposium on Concurrent Engineering, CE & CALS Conference, Washington DC, June 1992.

[Fikes 1997] R. Fikes and A. Farquhar: “Large-Scale Repositories of Highly Expressive Reusable Knowledge”, Knowledge Systems Laboratory, 1997.

[Fikes 2002] R. Fikes, P. Hayes and I. Horrocks: “DQL - A Query Language for the Semantic Web”. Knowledge Systems Laboratory, 2002.

[Genesereth 1992] Michael R. Genesereth: “Knowledge Interchange Format version 3.0”, Report Logic 92-1, Stanford University, Stanford CA, June 1992.

[Giacomo 2000] G. De Giacomo, Y. Lesperance and H. Levesque: “ConGolog: A Concurrent Programming Language Based On the Situation Calculus”, Artificial Intelligence, 2000.

[Gibbins 2003] N. Gibbins, S. Harris, and N. Shadbolt: “Agent-Based Semantic Web Services”, Journal of Web Semantics: Science, Services and Agents on the World Wide Web 2003.

[Golbeck 2003] Jennifer Golbeck, Bijan Parsia, and James Hendler: “Trust Networks on the Semantic Web”, Proceeding of Cooperative Intelligent Agents 2003, Helsinki Finland, August 2003.

[Greenwald 1999] Amy Greenwald and Jeffrey Kephart: “Shopbots and Pricebots”, International Joint Conferences on Artificial Intelligence, Stockholm, August 1999.

[Greenwald 2001] Amy Greenwald and Peter Stone: “Autonomous Bidding Agents in the Trading Agent Competition”, IEEE Internet Computing, March/April 2001.

[Greenwald, 2003] Amy Greenwald: “The 2002 Trading Agent Competition: An Overview of Agent Strategies”, AI Magazine.

[Grimnes 2003] G. Grimnes, S. Chalmers, P. Edwards and A. Preece: “GraniteNights: A Multi-Agent Visit Scheduler Utilising Semantic Web Technology”, Seventh International Workshop on Cooperative Information Agents, 2003.

[Gruber 1993] T. R. Gruber: “A Translation Approach to Portable Ontologies”, Knowledge Acquisition, Volume 5(2) page 199-220, 1993.

[Guha 2003] R.Guha, Rob McCool, Eric Miller: “Semantic Search”, The Twelfth International World Wide Web Conference, Budapest Hungary, May 2003.

[Haarslev 2001] Volker Haarslev and Ralf Moller: “Racer System Description”, Proceeding of International Joint Conference on Automated Reasoning, Volume 2083, page 701-705, Springer 2001.

[Harmelen 2000] F. van Harmelen and I. Horrocks: “FAQs on OIL: the Ontology Inference Layer”, IEEE Intelligent Systems: Trends and Controversies, November/December 2000.

[Hayes 2002] Pat Hayes: “RDF Model Theory”, W3C Working Draft, Feb 2002.

[Hayes 2003] Pat Hayes: “RDF Semantics”, W3C Working Draft, 2003.

[Heflin 2000] Jeff Heflin and James Hendler: “Searching the Web with SHOE”, AAAI-2000 Workshop on AI for Web Search, 2000.

[Hendler 2000] James Hendler and Deborah L. McGuinness: “The DARPA Agent Markup Language”, IEEE Intelligent Systems: Trends and Controversies, page 6-7, November/December 2000.

[Hendler 2001] James Hendler: “Agents and the Semantic Web”, IEEE Intelligent Systems Journal, March/April 2001.

[Horrocks 1998] Ian Horrocks: “The FaCT system”, Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98, number 1397 in Lecture Notes in Artificial Intelligence, page 307-312. Springer, May 1998.

[Horrocks 2003] Ian Horrocks: “Reducing OWL Entailment to Description Logic Satisfiability”, Proceeding of ISWC 2003, page 17-24, Octomber 2003.

[Karvounarakis 2001] Gregory Karvounarakis, Vassilis Christophides, Dimitris Plexousakis and Sofia Alexaki: “Querying RDF Descriptions for Community Web Portals”, BDA-01, Nov 2001.

[Karvounarakis 2002] Gregory Karvounarakis, Sofia Alexaki , Vassilis Christophides , Dimitris Plexousaki and Michel Scholl: “RQL: A Declarative Query Language for RDF”, WWW 2002, Honolulu Hawaii, 2002.

[Kearns 2003] Michael Kearns and Luis Ortiz: “The Penn-Lehman Automated Trading Project”, 2003.

[Kifer 1995] M. Kifer, G. Lausen, and J. Wu: "Logical Foundations of Object-Oriented and Frame-Based Languages", Journal of ACM, Volume 42(4), page 741-843, Jul 1995.

[Klyne 2002] G.Klyne: "Framework for Security and Trust Standards", SWAD-Europe, December 2002.

[Lassila 1999] Ora Lassila and Ralph R. Swick: "Resource Description Framework (RDF) Model and Syntax Specification", W3C Recommendation, February 1999.

[Lassila 2001] Ora Lassila: "Enabling the Semantic Web Programming by Integrating RDF and Common Lisp", Semantic Web Working Symposium, Stanford CA, May 2001.

[Lassila 2002] Ora Lassila: "Taking the RDF Model Theory Out for a Spin", First International Semantic Web Conference, Sardinia Italy, June 2002.

[Luke 2000] Sean Luke and Jeff Heflin: "SHOE 1.01. Proposed Specification", February, 2000.

[McIlraith 2001] Sheila A. McIlraith, Tran Cao Son and Honglei Zeng : "Mobilizing Semantic Web with DAML-Enabled Web Services", Semantic Web Working Symposium, Stanford CA, May 2001.

[Melnik 2000] Sergey Melnik and Stefan Decker: "A Layered Approach to Information Modeling and Interoperability on the Web", Proceedings of ECDL 2000 Workshop on the Semantic Web, 2000.

[Milgrom 1987] P.R.Milgrom: "Auction Theory", Advances in Economic Theory: Fifth World Congress, Cambridge University Press, 1987.

[Motik 2002] B. Motik, A. Maedche and R. Volz: "A Conceptual Modeling Approach for building semantics-driven enterprise applications", Proceedings of the First International Conference on Ontologies, Databases and Application of Semantics (ODBASE-2002), Springer, LNAI, California USA, 2002.

[Noy 2001] N. Noy and M. Musen: "Anchor-PROMPT: Using Non-Local Context for Semantic Matching", Proceedings of the IJCAI Workshop on Ontologies and Information Sharing, 2001.

[Odell 2000] J. Odell, H. Parunak and B. Bauer: "Extending UML for Agents", Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, 2000.

[O'Brien 1998] P. O'Brien and R. Nicol: "FIPA - Towards a Standard for Software Agents", BT Technology Journal, Volume 16-3, page 51-59, 1998.

[OWL-S 2004] Semantic Web Services group: "OWL-S 1.0", Jan 2004.

[Payne 2002] Terry R. Payne, Rahul Singh and Katia Sycara: "Calendar Agents on the Semantic Web", IEEE Intelligent Systems Volume 17(3), page 84-86, May/June 2002.

[Peng 2002] Yun Peng, Nenad Ivezic, Youyong Zou and Xiaocheng Luan: "Semantic Resolution for E-Commerce", First GSFC/JPL Workshop on Radical Agent Concepts, NASA Goddard Space Flight Center MD, January 2002.

[Peter 2003] Peter F. Patel-Schneider, Pat Hayes and Ian Horrocks: "OWL Web Ontology Language Semantics and Abstract Syntax", W3C Working Draft, 2003.

[Poslad 2000] S. J. Poslad, Rachel Bourne, Alex L. G. Hayzelden: "Agent Technology for Communications Infrastructure: An Introduction", Agent Technology for Communications Infrastructure, 2000.

[Russell 1995] S. Russell and P. Norvig: “Artificial Intelligence: A Modern Approach”, New York Prentice-Hall 1995.

[Sagonas 1994] Kostantinos Sagonas, Terrance Swift and David S. Warren: “XSB as an Efficient Deductive Database Engine”, ACM Conference on Management of Data (SIGMOD), 1994.

[Seaborne 2003] A. Seaborne: “RDQL – RDF Data Query Language”, HP Labs Semantic Web activity.

[Sintek 2002] Michael Sintek and Stefan Decker: “TRIPLE-A Query, Inference and Transformation Language for the Semantic Web”, First International Semantic Web Conference (ISWC), Sardinia Italy, June 2002.

[Staab 2000] Steffen Staab, Michael Erdmann, Alexander Maedche and Stefan Decker: “An Extensible Approach for Modeling Ontologies in RDF(S)”, Proceedings of ECDL 2000 Workshop on the Semantic Web, 2000.

[Stone 2000] Peter Stone and Amy Greenwald: “The First International Trading Agent Competition: Autonomous Bidding Agents”, Electronic Commerce Research Journal Page 1-36, 2000.

[Wellman 1999] Michael P. Wellman and Peter R. Wurman: “A Trading Agent Competition for the Research Community”, IJCAI-99 Workshop on Agent-Mediated Electronic Commerce, Stockholm Sweden, 1999.

[Wellman 2001] Michael P. Wellman, Peter R. Wurman, Kevin O'Malley, Roshan Bangera, Shou-de Lin, Daniel Reeves and William E. Walsh: “A Trading Agent Competition”, IEEE Internet Computing, Volume 5(2), page 43-51, March/April 2001.

[Wellman 2002] Michael P. Wellman, Amy Greenwald, Peter Stone and Peter R. Wurman: “The 2001 Trading Agent Competition”, Fourteenth Innovative Applications of Artificial Intelligence Conference (IAAI-2002), page 935-941, Edmonton Canada, August 2002.

[Willmott May 2001] S. Willmott, I. Constantinescu and M. Callisti: “Multilingual Agents: Ontologies, Languages and Abstractions”, Proceedings of the First International Workshop on Ontologies in Agent Systems, Autonomous Agents 2001, Montreal Canada, May 2001.

[Willmott Nov 2001] S. Willmott, J. Dale, B. Burg, P. Charlton and P. O'Brien: “Agentcities: A Worldwide Open Agent Network”, AgentLink News, Issue 8, November 2001.

[Wong 1999] H.C. Wong and K. Sycara: “Adding Security and Trust to Multi-Agent Systems”, Proceedings of Autonomous Agents-Workshop on Deception, Fraud and Trust in Agent Societies, 1999.

[WSA 2003] Web Services Architecture Group: “Web Services Architecture”, W3C Working Draft, August 2003.

[Yang 2000] Guizhen Yang and Michael Kifer : “FLORA: Implementing an Efficient DOOD System Using a Tabling Logic Engine”, Proceedings of Computational Logic 2000, page 1078-1093, Springer, July 2000.

[Zou, 2003] Youyong Zou, Tim Finin, Li Ding, Harry Chen and Rong Pan: “TAGA: Trading Agent Competition in Agentcities”, IJCAI Workshop on Trading Agent Design and Analysis, Acapulco MX, August 2003.