



**HAL**  
open science

# Autonomous semantic grid: middleware for supporting the interoperability of agents and web services

Maruf Pasha

► **To cite this version:**

Maruf Pasha. Autonomous semantic grid: middleware for supporting the interoperability of agents and web services. Software Engineering [cs.SE]. Université de Bretagne Sud, 2010. English. NNT : . tel-00519183

**HAL Id: tel-00519183**

**<https://theses.hal.science/tel-00519183>**

Submitted on 18 Sep 2010

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



THESE / Université de Bretagne-Sud  
*sous le sceau de L'Université Européenne de Bretagne*

pour obtenir le titre de :  
**DOCTEUR DE L'UNIVERSITE  
EUROPEENNE DE BRETAGNE**  
*Mention : Sciences et Technologies de l'Information et de la  
Communication*  
École Doctorale Informatique et Applications

présentée par

**Maruf PASHA**

Préparée Au laboratoire VALORIA  
Université de Bretagne-Sud

## **Grille sémantique autonome : un intergiciel pour l'interopérabilité d'agents et services web**

**Thèse soutenue le 4 Février 2010**

devant le jury composé de :

**Richard McCLATCHEY**

Professeur, University of the West of England, Royaume Uni /  
rapporteur

**Hiroki SUGURI**

Professeur, Miyagi University, Japon / *rapporteur*

**Arshad ALI**

Professeur, National University of Science and Technology –  
SEECS, Pakistan / *examineur*

**Pierre-François MARTEAU**

Professeur Université de Bretagne-Sud / *examineur*

**Danny WEYNS**

Chargé de recherche, Katholieke Universiteit Leuven,  
Belgique / *examineur*

**H.Farooq AHMAD**

Professeur associé, National University of Science and  
Technology – SEECS, Pakistan (Co-encadrant)

**Flavio OQUENDO**

Professeur, Université de Bretagne Sud, France (Directeur de  
thèse)



**Dedicated to my loving parents.**

## Acknowledgements

First of all I am thankful to Allah almighty for blessing me the courage and spirit to undertake PhD studies. I would like to express my most sincere gratitude to my supervisors Professor Flavio Oquendo and Dr. H. Farooq Ahmad. I am indebted to Professor Flavio Oquendo, whose continuous guidance helped me in achieving my goals and his ideas have always been a source of inspiration for my research. In spite of his extraordinary workload, he always managed to spare time for helping me whenever I needed his guidance. Without his efforts and valuable advice, it would not have been possible to achieve this dream. It has been a pleasure for me to work under his supervision. I am also thankful to Dr. Hafiz Farooq Ahmad who helped me clarify my research direction and to overcome the difficulties occurred during my research. His unlimited support helped me survive the desperate moments during my studies. This thesis work is an output of his vision and ideas.

I would like to express special thanks to Professor Richard McClatchey, Professor Hiroki Suguri, Professor Pierre-François Marteu, Professor Arshad Ali and Dr. Danny Weyns for their detailed reviews and comments that have lead to completion of this thesis.

I express my special gratitude to my parents who kept my spirits high to complete my PhD studies and were always there for me in my hard times, no matter how far they were. I want to express my sincere appreciation to my family for believing in me, for their patience, for their suffering throughout the years, for their love, sacrifices, and continued encouragement for counting the days and years to have me back. They have comforted me when I was frustrated, encouraged me when I was doubtful, and supported me throughout everything. I would also like to thank very special people in my life: my brothers, my sisters who were with me in every step throughout my journey.

I am highly thankful to all of my teachers who have been guiding me throughout my academic studies. Their knowledge, guidance and training helped me a lot to carry out this research work. I would like to offer my appreciation to all the project team, my close colleagues and all my friends especially Bilal, Haider, Jibrán, Nadeem, Zawar, who have been encouraging me throughout my research work

## Table of Contents

<b>RÉSUMÉ .....</b>	<b>12</b>
<b>ABSTRACT .....</b>	<b>13</b>
<b>CHAPTER 1 .....</b>	<b>15</b>
INTRODUCTION.....	16
1.1 SCOPE OF RESEARCH AREA.....	18
1.2 RATIONALE FOR RESEARCH.....	21
1.3 RESEARCH HYPOTHESIS .....	21
1.4 APPROACH TO THE RESEARCH QUESTIONS .....	22
1.5 SUMMARY .....	23
1.6 THESIS ORGANIZATION .....	24
<b>CHAPTER 2 .....</b>	<b>25</b>
PROBLEM DOMAIN.....	26
2.1 SEMANTIC INTEROPERABILITY .....	26
2.2 SEMANTICS FOR GRID COMPUTING .....	30
2.2.1 <i>Enabling semantics in Grid systems</i> .....	30
2.2.2 <i>Amalgamation of Information</i> .....	31
2.3 RESEARCH QUESTIONS .....	32
2.4 SYNERGY OF TECHNOLOGIES .....	33
2.5 SUMMARY .....	35
<b>CHAPTER 3 .....</b>	<b>37</b>
LITERATURE REVIEW .....	38
3.1 SOFTWARE AGENTS.....	39
3.1.1 <i>Personalizing Web Services through Agents</i> .....	39
3.2 MULTI-AGENT SYSTEMS (MAS) .....	40
3.3 AGENT PLATFORM .....	41
3.4 FIPA (FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS) .....	42
3.4.1 <i>Agent Management System (AMS)</i> .....	43
3.4.2 <i>Message Transport Service (MTS)</i> .....	43
3.4.3 <i>Agent Communication Language (ACL)</i> .....	44
3.4.4 <i>FIPA Semantic Language (FIPA SL)</i> .....	44
3.4.5 <i>Directory Facilitator</i> .....	45
3.4.6 <i>Visual Management Agent (VMA)</i> .....	45
3.5 GRID COMPUTING .....	46
3.6 SEMANTIC WEB .....	48
3.7 SEMANTIC GRID .....	50
3.8 WEB SERVICES .....	51
3.9 ROLE OF ONTOLOGIES.....	52
3.10 NEGOTIATIONS BETWEEN AGENTS AND WEB SERVICES.....	52
3.11 $\pi$ -ADL (ARCHITECTURE DESCRIPTION LANGUAGE) .....	53
3.12 STATE OF THE ART SOLUTIONS .....	54

3.12.1 <i>Web Services Integration Gateway (WSIG)</i> .....	54
3.12.2 <i>WS2JADE</i> .....	55
3.12.3 <i>OWL -P</i> .....	56
3.12.4 <i>Middleware Ontology Service (MOS)</i> .....	56
<b>3.13 COMPARISON OF STATE OF ART SOLUTIONS</b> .....	<b>57</b>
<b>3.14 SUMMARY</b> .....	<b>57</b>
<b>CHAPTER 4</b> .....	<b>59</b>
<b>PROPOSED SOLUTION</b> .....	<b>60</b>
<b>4.1 ONTOLOGY GATEWAY</b> .....	<b>60</b>
<b>4.2 ISSUES IN SEMANTIC INTEROPERABILITY</b> .....	<b>61</b>
<b>4.3 EXPRESSIVENESS OF OWL AND FIPA SL</b> .....	<b>62</b>
<b>4.4 REPRESENTING AGENT ATTITUDES</b> .....	<b>63</b>
<b>4.5 COMPARISON BASED ON LOGIC</b> .....	<b>64</b>
<b>4.6 PROPOSED SYSTEM ARCHITECTURE OF ONTOLOGY GATEWAY</b> .....	<b>66</b>
4.6.1 <i>Agents Communicating with OWL Based Services</i> .....	68
4.6.2 <i>OWL Based Service Communicating with Agent</i> .....	69
<b>4.7 DETAILS OF ARCHITECTURE COMPONENTS</b> .....	<b>70</b>
4.7.1 <i>Control Unit</i> .....	72
4.7.2 <i>Ontology Agent</i> .....	73
4.7.3 <i>OWL to FIPA SL Module</i> .....	75
4.7.4 <i>FIPA SL to OWL Module</i> .....	75
4.7.5 <i>SOAP to ACL Module</i> .....	75
4.7.6 <i>ACL to SOAP Module</i> .....	75
4.7.7 <i>Transformation Component</i> :.....	76
4.7.8 <i>Negotiation Module</i> :.....	78
4.7.9 <i>Decision Engine</i> .....	81
<b>4.8 SUMMARY</b> .....	<b>82</b>
<b>CHAPTER 5</b> .....	<b>83</b>
<b>IMPLEMENTATION AND VALIDATION</b> .....	<b>84</b>
<b>5.1 COMPARATIVE ANALYSIS OF SERVICE DISCOVERY AND COMMUNICATION PROTOCOLS</b> .....	<b>85</b>
5.1.1 <i>Analysis of service registry and discovery</i> .....	85
5.1.2 <i>Analysis of communication protocols</i> .....	86
<b>5.2 IMPLEMENTATION DETAILS OF THE PROPOSED SYSTEM</b> .....	<b>88</b>
<b>5.3 AGENTS INTERACTING WITH THE SEMANTICALLY ENABLED SERVICES</b> .....	<b>88</b>
<b>5.4 SEMANTICALLY ENABLED SERVICES/GRID CLIENT COMMUNICATING WITH THE AGENTS</b> .....	<b>90</b>
<b>5.5 NEGOTIATION BETWEEN AGENTS AND WEB SERVICES</b> .....	<b>92</b>
<b>5.6. VALIDATION</b> .....	<b>94</b>
5.6.1 <i>FIPA to OWL Translation</i> .....	96
5.6.2 <i>OWL to FIPA Translation</i> .....	97
<b>5.7. PERFORMANCE EVALUATION</b> .....	<b>99</b>
5.7.1 <i>FIPA SL to OWL Translation Analysis</i> .....	99
5.7.2 <i>OWL to FIPA SL Translation Analysis</i> .....	100
5.7.3 <i>Ontology Translation Comparison between OWL and FIPA</i> .....	101
<b>5.8 CLASSES OF APPLICATIONS</b> .....	<b>102</b>
5.8.1 <i>Flight reservation using Ontology Gateway</i> .....	103
5.8.2 <i>Semantically Enriched Agent Application using Ontology Gateway</i> .....	105
<b>5.9 SUMMARY</b> .....	<b>108</b>

<b>CHAPTER 6 .....</b>	<b>109</b>
<b>CONCLUSION.....</b>	<b>110</b>
<b>6.1 CONCLUSIONS.....</b>	<b>110</b>
6.1.1. <i>Road map towards addressing the Research Questions</i> .....	111
6.1.2. <i>Contribution</i> .....	114
<b>6.2 FUTURE WORK.....</b>	<b>115</b>
6.2.1 <i>Short-term objectives</i> .....	116
6.2.2 <i>Mid-term objectives</i> .....	117
6.2.3 <i>Long-term objectives</i> .....	117
<b>6.3. CONCLUDING REMARKS.....</b>	<b>118</b>
<b>APPENDIX A .....</b>	<b>128</b>
<b>APPENDIX B.....</b>	<b>133</b>
<b>APPENDIX C.....</b>	<b>138</b>
<b>APPENDIX D .....</b>	<b>141</b>
<b>APPENDIX E .....</b>	<b>143</b>
<b>APPENDIX F .....</b>	<b>153</b>



## List of Figures

FIGURE 1.1: AUTONOMOUS SEMANTIC GRID CLASSIFICATION AMONG OTHER TECHNOLOGIES. ....	20
FIGURE 1.2: INTEGRATION OF AGENTS AND WEB SERVICES, A MIDDLEWARE BASED APPROACH .....	22
FIGURE 2.1: AUTONOMOUS SEMANTIC GRID.....	33
FIGURE 2.2: SYNERGY OF SOFTWARE AGENTS WITH WEB SERVICES AND GRID COMPUTING .....	35
FIGURE 3.1: FIPA ARCHITECTURE MODEL (FIPA, 2002) .....	42
FIGURE 3.2: COMPONENTS OF FIPA COMPLIANT AGENT FRAMEWORK.....	43
FIGURE 3.3: AGENT COMMUNICATION LANGUAGE .....	44
FIGURE 3.4: STACK VIEW FOR SERVICE ORIENTED GRIDS.....	47
FIGURE 3.5: THE SEMANTIC WEB STACK.....	49
FIGURE 3.6: NEGOTIATION PROCESS SPECIFIED BY FIPA .....	53
FIGURE 4.2: AGENTS COMMUNICATING WITH OWL BASED SERVICES .....	68
FIGURE 4.3: OWL BASED SERVICES INTERACTING WITH AGENTS .....	69
FIGURE 4.4: THE $\pi$ -ADL MODEL OF THE OVERALL ARCHITECTURE .....	71
FIGURE 4.5: CONTROL UNIT SPECIFICATIONS IN $\pi$ -ADL .....	73
FIGURE 4.6: ONTOLOGY AGENT SPECIFICATIONS IN $\pi$ -ADL.....	75
FIGURE 4.7: TRANSFORMATION COMPONENT.....	76
FIGURE 4.8: NEGOTIATION AMONG AGENTS AND WEB SERVICES.....	78
FIGURE 4.9: NEGOTIATION MODULE.....	79
FIGURE 4.10: RUNTIME AGENT .....	80
FIGURE 4.11: DECISION ENGINE .....	82
FIGURE 5.1: ACL TO SOAP CONVERSION STEPS .....	87
FIGURE 5.2: SOAP TO ACL CONVERSION STEPS .....	88
FIGURE 5.3: AGENTS INTERACTING WITH THE SEMANTICALLY ENABLED WEB SERVICES .....	89
FIGURE 5.4: SEMANTICALLY ENABLED SERVICES/GRID CLIENT COMMUNICATING WITH THE AGENTS .....	90
FIGURE 5.5: NEGOTIATION BETWEEN AGENTS AND WEB SERVICES. ....	92
FIGURE 5.6: FIPA TO OWL CONVERSION STEPS .....	96
FIGURE 5.8: FIPA TO OWL CONVERSION STEPS .....	98
FIGURE 5.9: UNIVERSITY ONTOLOGY IN PROTÉGÉ .....	98
FIGURE 5.10: ACL QUERY TO RETRIEVE INFORMATION FOR A PARTICULAR ENTITY.....	99
FIGURE 5.11: FIPA SL TO OWL TRANSLATION.....	100
FIGURE 5.12: OWL TO FIPA SL TRANSLATION.....	101
FIGURE 5.13: ONTOLOGY TRANSLATION COMPARISON BETWEEN OWL AND FIPA .....	102
FIGURE 5.14: FLIGHT RESERVATION USING ONTOLOGY GATEWAY.....	103
FIGURE 5.15: $\pi$ -ADL CODE FOR A SIMPLE WEB SERVICE .....	105
FIGURE 5.16: SUPPLY CHAIN MANAGEMENT SYSTEM USING ONTOLOGY GATEWAY. ....	106
FIGURE 6.1: FUTURE RESEARCH DIRECTIONS .....	116

## List of Tables

TABLE 3.1 COMPARISON OF STATE OF THE ART SOLUTIONS .....	57
TABLE 4.1. COMPARISON OF OWL AND FIPA-SL AS CONTENT LANGUAGE .....	63
TABLE 4.2. OWL CLASS MAPPINGS .....	65
TABLE 4.3. OWL PROPERTY MAPPINGS .....	66
TABLE 5.1. COMPARISON BETWEEN DIRECTORY FACILITATOR AND UDDI SERVICE DESCRIPTIONS .....	85
TABLE 5.2. COMPARISON BETWEEN COMMUNICATION PROTOCOLS.....	86
TABLE 5.3. EXTENDED MAPPINGS BETWEEN FIPA ONTOLOGY AND OWL ONTOLOGY TAGS .....	95
TABLE 6.1 CONTRIBUTION TO STATE OF ART .....	114

## List of Abbreviations

MAS	Multi-agent Systems
FIPA	Foundation for Intelligent Physical Agents
AI	Artificial Intelligence
AMS	Agent Management System
MTS	Message Transport Service
ACC	Agent Communication Channel
MTP	Message Transport Protocol
DF	Directory Facilitator
ACL	Agent Communication Language (ACL)
SL0	Semantic Language 0
SL1	Semantic Language 1
SL2	Semantic Language 2
OWL	Ontology Web Language
XML	Extensible Markup Language
HTTP	Hyper Text Transfer Protocol
SOAP	Simple Object Access Protocol
WSDL	Web Services Description Language
UDDI	Universal Description and Discovery Integration
RDF	Resource Description Framework
OWL-S	Ontology Web Language for Services
API	Application Programming Interface
DAML	DARPA Agent Markup Language
DAML-S	DARPA Agent Markup Language for Services
OIL	Ontology Inference Layer
OGSA	Open Grid Service Architecture
WSRT	Web service Resource transfer
WSRF	Web Service Resource Framework
ASG	Autonomous Semantic Grid
OG	Ontology Gateway



## Résumé

Le développement des nouveaux paradigmes et technologies comme le Web sémantique, le calcul sur grille et les services Web ouvrent de nouvelles perspectives et défis pour la conception d'une nouvelle génération d'applications. Cette nouvelle génération peut être conçue comme des systèmes multi-agents opérant sur les grilles de calcul tout en s'appuyant sur les services offerts par les services Web sémantiques.

D'une part, les avancées dans la standardisation des langages et technologies des services Web ont permis un déploiement rapide d'applications fondées sur l'invocation dynamique de services découverts à la volée. Ainsi, grâce aux services Web, les applications peuvent être vues comme un ensemble de services logiciels, explicitement décrits, découverts et invoqués à l'aide des standards internationaux définis par le W3C (World Wide Web Consortium).

D'autre part, le Web sémantique vise à rendre le contenu des ressources du Web accessible et utilisable par les agents logiciels, grâce à un système de métadonnées. Ces métadonnées sont notamment exprimées à l'aide d'ontologies définies en OWL – Web Ontology Language, un langage standardisé par le W3C. L'extension du Web sémantique aux services Web donne ainsi lieu à des services Web sémantiques.

Le Web sémantique est également à la base des grilles sémantiques fournissant une infrastructure virtuelle constituée d'un ensemble de ressources potentiellement partagées, distribuées, hétérogènes, délocalisées et autonomes. Ces ressources sont abstraites en tant que services grille.

La conjonction des services Web et des systèmes multi-agents déployés sur les grilles sémantiques donne naissance aux grilles sémantiques autonomes. Dans cette vision, des agents logiciels exploitent les services des grilles de calcul mais également utilisent des services Web au-delà de la grille, permettant un usage à très large échelle. Ces systèmes multi-agents sont construits en se reposant souvent sur les standards IEEE de la FIPA (Foundation of Intelligent Physical Agents).

Néanmoins, un problème se pose : celui de l'interopérabilité des agents logiciels, de nature sémantique, avec les services Web sémantiques, tout en respectant les standards définis par la FIPA et le W3C.

Cette thèse s'attaque à cette problématique. L'approche développée dans la thèse est fondée sur la conception d'un intergiciel permettant l'interopérabilité entre agents logiciels et services Web de manière transparente tout en s'appuyant sur les standards actuels promus par la FIPA pour les systèmes multi-agents et pour le W3C pour les services Web sémantiques. Elle définit l'architecture logicielle et implémente en intergiciel pour la médiation entre agents logiciels et services Web sémantiques, en prenant en compte la dimension ontologique. La solution proposée est validée par des études de cas et utilise  $\pi$ -ADL, un langage formel basé sur le  $\pi$ -calcul, pour spécifier l'application des mécanismes développés.

## Abstract

Technological advancements in Web services standards have led to the development and deployment of a large number of applications in open and dynamic environments. These standards enable the services to be discovered and invoked dynamically. The current Web however lacks the ability to specify the complete semantic annotations and is thus limited, to be utilized by intelligent applications. Semantics can therefore play a vital role in the automation and enhancement of Web services. Agents are intelligent entities that can be integrated with the current infrastructure thus influencing the negotiation, coordination and cooperation among the heterogeneous environments. We believe that the vision of the Semantic Web can be realized if the agents are intelligent enough to process and interpret semantic content based on the understanding which they have developed about the contents through the use of ontologies, since the agents have well defined reasoning, decision making, and interaction mechanisms. Our objective is to propose a new middleware based approach for interactions among Semantic Web services and software agents.

The Semantic Web is an extension of the current World Wide Web aimed at marking web contents with richer metadata so that they can be processed by machines without human intervention. The Semantic grid is a new initiative to apply Semantic Web technologies to traditional Grid computing for automating the registration, discovery, composition and orchestration of grid services and resources. Ontologies can play an essential role for providing semantics in the Semantic Web and the Semantic grid and to lay down the foundation for interoperability among heterogeneous distributed systems. Semantics is thus a key component for achieving autonomy in the process of efficient service provision and utilization. Software agents can play an important role in the Semantic grid based on their capabilities to realize virtual organizations and virtual services that are a part of any Grid based system.

The main focus of this research is to propose a middleware based approach for the communication of agents and Web Services and to clearly specify the sequence of steps involved in this communication process and how ontologies can play a role in assisting this communication process. These issues have not previously been addressed all together as a single solution. The issues of heterogeneity among these technologies are managed by providing architectural specifications and a prototype implementation in a middleware to achieve semantic interoperability between these technologies without violating the standards for both these technologies (i.e. FIPA and W3C respectively). As the development and implementation of such a complex system is cumbersome,  $\pi$ -ADL a formal language has been used to develop the specifications and formal verification of the overall system at an abstract level using a  $\pi$ -ADL.NET compiler.

## Reading Guidelines

### Guidelines for Reading



The use of this symbol within the margin represents a precise definition.

### Guidelines for Reading



The use of this symbol within the margin represents additional information on a concept, and that the understanding of this concept is not compulsory.

# Chapter 1



## Introduction

---

The aim of the first chapter is to introduce the reader to the domain of agents and Web Services. A brief introduction to the domain is given by highlighting the facts that agents and Web services are different technologies but can lead to useful applications in the future by the convergence of their interest. The research domain has been explored among the various existing technologies and how the other technologies can affect the scope of this project. The aim is to highlight the limitations of the existing system's need of a solution that can bridge the gap between agents and Web service communication. A hypothesis has been presented with the overview of the questions that need to be resolved to investigate the hypothesis. A middleware based approach is adopted as solution to provide meaningful communication and interaction between agents and Web services.

Information and knowledge are becoming the focus of economic activity for the entire world. Global information systems, such as the Internet and World Wide Web, are no longer just pathways for digital data; rather they generate new information and knowledge for commercial activity, education, business and research. Since the advent of mankind this is the first time that resources such as services and information have become an integral and key component of the information based society. For the first time, in the history of human kind, non-material resources such as software and information services have become the real wealth of the knowledge-based society. This poses a great challenge to processing this extremely large set of data and to extract new knowledge autonomously. A number of new technologies such as Grid computing, the Semantic Web and software agents are emerging that can help to cope with this challenge through the synergies of their integration (Ahmad et al.,2003)(Foster et al.,2004). Our aim is to focus on these technologies step by step and to design a new system through the integration of these technologies in order to cope with this new challenge.

The World Wide Web provides an infrastructure for deploying Web services for providing useful applications to the end users. However, their usefulness and popularity have posed a number of challenges. The static and human understandable contents in its underlying architecture limit its capability to cope with the scale of information in the near future. The World Wide Web community has responded to find the solutions to these challenges in its Semantic Web initiative. The Semantic Web concept is to create computer understandable information by marking the contents with richer meta-data so that Web contents can be

consumed and processed by intelligent software. Grid computing provides virtualization of computation and resources to create dynamic virtual organizations by extending Web services to Web Services Resource Framework (WSRF,2004), (Shafiq et al.,2005) and Web Services Resource Transfer (WSRT,2006). It attempts to provide a single image of networked computers to users. However, the problems of the current Web as described above are inherited in Grid computing. The Semantic Grid is a new initiative aimed at applying Semantic Web on Grid computing and provides a platform where contents and services are exploited on the basis of their semantics rather than syntax.

Computer networks today provide a transparent access to the resources or services, but the main aim from the beginning has always been the same, that is to extract information from distributed locations and represent it in the desired form no matter what heterogeneities exist among the sources or where they are located. Typical examples of these networks can be mobile networks, factory networks and home networks. Therefore, if a person is connected to a network through his personal computer or a mobile device from a remote location, the whole scenario will be considered under the umbrella of distributed computing.

At this point it is important to define and highlight the importance of distributed computing in compliance with the research that is being done. A distributed system can be explained as a networked environment where the resources (software or hardware) can coordinate by messages passing to one another, but these resources can be located anywhere on the globe i.e. they can be in the same room or on a different continent. A distributed system is usually responsible for sharing heterogeneous resources like information repositories, hardware components or software services in a transparent manner and independently from each other's failure. Despite its heterogeneous nature the aim is to offer continuous services to the users of Internet with high performance. The solutions that exist to form a distributed system are comprised of policies to spread out its components over a network and hardware/software technology to implement it. These solutions for the distributed technologies that are well know and popular among the users have come a long way in becoming ubiquitous as the distributed systems have faced a lot of serious challenges since their origin.

Originally the Internet was designed to share information between a limited numbers of users without taking into consideration the quality of service. In addition to quality of service, we identify that users have two more basic views of customization and situation regarding information services utilization but these do not exist on the current information service systems as well. However, due to technological advancements and the evolution of real time applications and the emergence of the electronic commerce, there is a dire need to alter the fundamental viewpoint of the underlying infrastructure. Therefore, using information services on the Internet is a provoking experience for most of the users.

It is therefore, a vital task to develop new models in accordance with the information services for the real time applications and electronic commerce on the internet, the delay in providing the necessary technology and framework can lead to the decline of the electronic commerce or its growth may even become dubious. This fosters an immediate requirement for designing a system that can provide the necessary information to culminate the requirements that have been mentioned above. The next section explains the scope of research and which other technologies will come together for achieving the ultimate Goal of Autonomous Semantic Grid.

## 1.1 Scope of Research Area

This research can be categorized as part of distributed computing systems. Distributed systems encompass in itself, very popular and promising technologies like Web services, Grid systems and Multi-agent Systems. The aim of the Web services endeavor is laid on the dogma of Service Oriented Computing as to provide an implementation neutral environment and support for accessing heterogeneous resources. However, the goal of Grid systems is to provide focus on sharing resources in a coordinated way between virtual organizations.

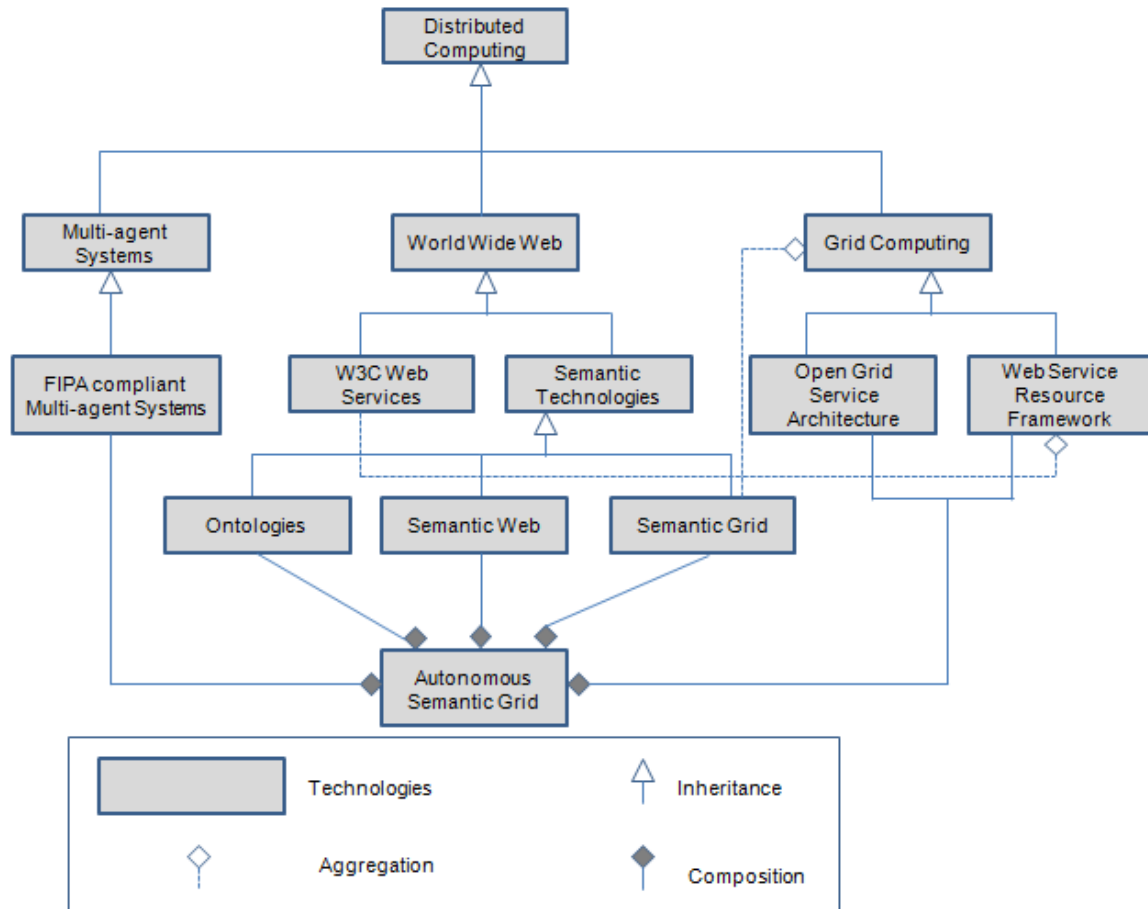
The Semantic Web is an idea presented by World Wide Web inventor Tim Berners-Lee to make the current Web intelligent enough to provide the results to end users an accurate and efficient way. According to Berners-Lee the search engines maintain the capability of indexing most of the content that is present on the Web, but lacks the ability to retrieve the results or information required by the user in a precise manner. Therefore, he proposes techniques by which the developers can associate self-descriptive semantic annotations so that the context for a particular query is well understood and as a result only relevant information is retrieved.

The foremost endeavor of current research in Semantic Web (Lee, 2001) aims at incorporating comprehensive semantic descriptions so that the software entities like agents can understand them and can eventually perform registration, discovery, invocation and composition of services (Foster et al.,2004). Web services are a loosely coupled, self-governing computational body that enables the development and deployment of services in distributed environments. According to the World Wide Web Consortium (W3C) working group software “agents play a key role for the Web services implementation and in retrieval of resources (hardware or software) according to the preferences/requirements of the user” (Foster et al.,2004). In Semantic Grid, Web services further combines with Semantic Web technologies to enable dynamic Web service discovery, invocation, composition, interoperation and execution monitoring.

The Web has provided common grounds to share the information all around the globe. The Grid technologies (Foster et al.,1999) build on this by facilitating the global sharing of not just information, but also of physical resources (that are in terms of computational and data storage resources) to be used at a distance. E-mail and the Web provide vital means that allow communities that cover different states, countries and continents to work together in collaboration. Visualize a scenario if everyone could link their data, computers and other resources into a single virtual office. Grid technology provides the basic support in terms of protocols, services and development environment to make it possible to share resources on a broader level.

At the core of the Grid is the concept of virtual organization. It is a dynamic collection of individuals, institutions and resources bundled together in order to share resources as they tackle common goals. This resource sharing is not only providing the facility of file exchange, but rather direct, controlled (i.e. within the authorization, security, copyright, etc. restrictions) access to resources which can be computational resources, storage or data resources and can be utilized for solving complex problems by collaborative efforts of different industry partners.

The goal while developing Multi-agent Systems (Wooldridge et al., 2002) is the interaction and communication of software agents with one another and the focus is on the maturity of the languages used for communication, protocols used for interaction etc. Agents possess properties, like autonomy, and thus can maintain their own state in order to cope with the situation that is being faced. It is not compulsory that one agent has to perform only one goal, but a number of agents can follow the same goal and can interact with each other for its fulfillment. The agents utilize the semantic representation provided in terms of ontologies for providing meaningful communication between the agent groups. The agent technology provides the autonomy and intelligence capabilities that could facilitate the development of electronic commerce and business applications, where the requirement is to introduce independent entities to handle the communication as the environment is dynamic in nature. Incorporating such entities will reduce human intervention, increase production, minimize the chances of errors and ambiguities, and, last but not the least, will enable the handling of the scalability issue on a large scale (Negri et al., 2006).



**Figure 1.1:** Autonomous Semantic Grid classification among other technologies.

Figure 1.1 refers to the categorization of Autonomous Semantic Grid and how it evolved from the existing technologies. The ultimate aim of Autonomous Semantic Grid is to develop a framework that could provide solution for integrating technologies Web services, Grid Computing and Multi-agent Systems. The development of such a framework will require the integration of capabilities of all three of these technologies without making any changes to the existing standard formats. Once the building blocks of this framework are in place the agents will be able to discover the resources in grid, identified by their semantic annotations and will enable the semantic interoperability between the technologies. The next section highlights the rational for developing such a system that can solve the issues that are not considered by the current technology vendors.

## 1.2 Rationale for Research

The goal of the Autonomous Semantic Grid is that software agents would be able to autonomously and dynamically discover, compose and invoke some particular Web services. Information retrieval in the current Web is very difficult as it is based on conventional information processing in which just the text of the documents is processed and there are no semantics. Foundation of Intelligent Physical Agents (FIPA, 2003) is responsible for developing the specifications of agents and Multi-agent Systems, whereas the standard governing body for the Web services is World Wide Web Consortium (W3C, 2001). The differences in structure and specifications of both technologies act as an obstacle between the communication of software agents and the Web services.

We assume that software agents will play a significant role in the automation of managing Web services and will thus reduce complexity. Therefore, the aim is to integrate both technologies, i.e. agents and Web services. Keeping into consideration the aspect that both these technologies are based on distinct standards and have specified their own vocabulary, thus there are chances of misinterpretation while one technology wants to retrieve information from the repository of the other. We believe that associating semantics can facilitate the interoperability between agents and Web services. Since this interoperability requires the integration of several technologies and raises a number of challenges. Based on these challenges a research hypothesis has been stated with a summary of associated research questions.

## 1.3 Research Hypothesis

The research hypothesis stated in this thesis is:

“It is possible to develop system architecture for interoperability between agents with the Web services without changing existing specification of both the governing bodies i.e. FIPA and W3C?”

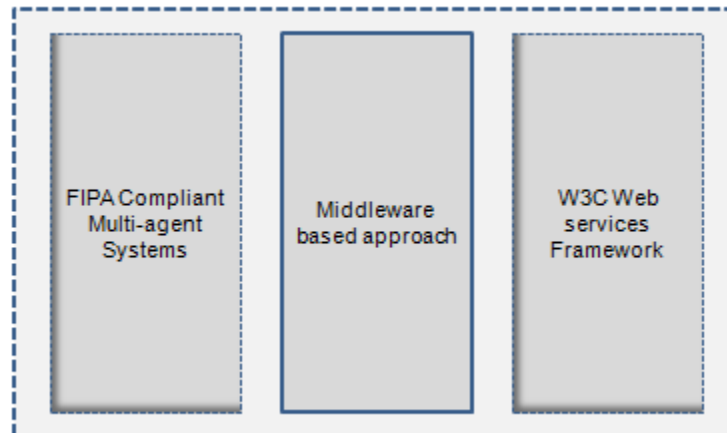
To address the research hypothesis we need to answer a number of research questions which are summarized as follows:

What is the role of ontologies in the communication between agents and Web services? How will the task delegation be performed once a certain piece of information is requested? How can the agents negotiate to a Web service once it is invoked? A detailed list of questions and sub questions is explicitly mentioned in the next chapter. In order to answer these sub questions and achieve this interoperability we have proposed the design of a middleware based approach, referred to as Ontology Gateway, as a solution, that will enable

communication between agents and Web services without tampering or altering the existing standards and will facilitate in service registration its discovery and invocation. Designing and implementing such a system is a complex, time consuming and cumbersome task, therefore clear and unambiguous specifications are required for the architectural components. To provide the architectural specifications of all the participants, a formal language called  $\pi$ -ADL (Oquendo, 2004) has been used and these specifications are validated by using  $\pi$ -ADL.NET compiler (Qayyum, 2008). Keeping into consideration the varying technologies, a middleware based solution is suggested which is highlighted in the next section.

## 1.4 Approach to the Research Questions

To develop a system architecture for interoperability between agents with the Web services without changing the existing specification of both the governing bodies of FIPA and W3C. This transformation will facilitate the autonomous agents to dynamically discover and translate the Web services. The approach proposed in this thesis is to design and develop an Ontology Gateway (middleware based approach) as a solution that could act as a bridge for eliminating the gap between Multi-agent System and Web services communication and without any changes to standards specified by Foundation of Intelligent Physical Agents (FIPA) and World Wide Web Consortium (W3C) (agents and Web services) respectively. The middleware based approach is the most optimal as there are currently a lot of systems that have been implemented and deployed already.



**Figure 1.2:** Integration of agents and Web services, a middleware based approach

Taking into consideration the existing systems, it is not feasible to deploy them as services or to use some other alternative for rebuilding them. Therefore, the proposal is to use a middleware based approach for such systems as depicted in Figure 1.2.

It is important to have some insight of the limitations in the current systems and how different researchers are trying to cope with them. The data retrieved from the current Web as a result of certain query is usually unstructured and is based on the processing of the text layout without keeping into consideration the real meaning of data that is being processed. Incorporating semantics with the published services can enhance their better understanding by the machines, and entities like agents can fully utilize them thereby eliminating human involvement. Associating semantics with the services not only make them meaningful, but also improve the chances of acquired knowledge through the application of simple tools and techniques. Three different methodologies have been identified by (Shafiq et al.,2006) for developing effective communication between the agents and Web services that are:

- Compliance of one technology with the other.
- Introducing a middleware between technologies
- Integration that will result in least changes.

Web Services Integration Gateway Service (WSIGS) (Greenwood et al.,2004) is an example of integration with least changes and is almost in accordance with the gateway that was adapted by the AgentCities (Agentcities,2003).

We have already proposed AgentWeb Gateway (Shafiq et al.,2005) and Middleware Ontology Service (Khalid et al.,2007) for reducing the communication gap between technologies like software agents and Web services. The goal of this research is to integrate technologies like software agents, Web services and Grid without altering the specifications that have been defined by the governing bodies in their respected areas by developing system architecture, i.e. middleware based approach, to enable the communication of agents and Web services that have the capabilities of registration, searching, translation, and querying for a particular service.

## **1.5 Summary**

This chapter outlines the necessary details that act as a prelude to the chapters yet to appear. An overview of the current technologies is presented, but despite all the advancements that have been made in these domains, no one has formulated a precise solution that can address the issues raised during agent and Web services communication. The objective in this thesis is to provide a transparent integration of FIPA compliant Agents with Web services as specified by W3C by developing a middleware “Ontology Gateway”. The middleware based approach facilitates the required integration without changing existing specifications. By integration, I mean enabling two way service discovery, service publishing



and service invocation. A brief discussion has also been done on the state of the art solutions and their limitations that have lead to the proposed investigation in this thesis.

## **1.6 Thesis Organization**

The thesis is organized as follows: Chapter 2 gives an overview of the problem domain and the precise research questions that are dealt with in this thesis. Chapter 3 presents a detailed overview of the involved technologies including Agents, Grid computing and Semantic Grid. Chapter 4 discusses the semantic interoperability and issues involved along with the proposed system architecture of Ontology Gateway in detail and also its formal representation in  $\pi$ -ADL. Chapter 5 discusses the implementation and evaluation of the proposed architecture along with the case studies as a proof of concept for the proposed architecture. Finally, contribution and future directions are presented in chapter 6.

# Chapter 2

## Problem Domain

---

The introduction presented in the first chapter highlighted the concepts that have been used throughout the thesis and the categorization of this research to fit the existing technologies along with the necessity for developing a solution to overcome the limitations in the current systems. Based on the research questions, a middleware based approach has been suggested to tailor the existing technologies for overcoming the limitations in existing systems. This chapter explains the problem domain in depth. The first section of this chapter presents in detail the semantic interoperability from the perspective of various researchers. The second section illustrates the need of semantics in systems like Grids. A precise list of research questions and sub questions have been discussed that needs to be addressed in this thesis. The last section pinpoints the technology mergers that are required to realize the Autonomous Semantic Grid vision.

### 2.1 Semantic Interoperability

Interoperability has been an important research area since the advent of computing. Initially growth of user needs and advancements in networking and computing technology gave rise to network interoperability issues. According to (Heiler, 1995) interoperability among components of large scale distributed systems is the “ability to exchange services and data among one another”. Interoperability among distributed applications is an active research area to develop large scale distributed applications. Different forms of interoperability have been defined in information systems and are categorized as syntactic interoperability, structural interoperability, and semantic interoperability.

#### Semantic Interoperability



Semantic interoperability is an enterprise capability derived from the application of special technologies that infer, relate, interpret, and classify the implicit meanings of digital content, which in turn drive business processes, enterprise knowledge, business rules and software application interoperability.

## Ontology



Ontology is a formal, explicit specification of a shared conceptualization.

## Web Services



W3C defines Web services as a software system designed to support the machine to machine interaction over a network, with its interface defined in WSDL and other system interacts with it using SOAP messages.

Syntactic interoperability is required to deal with application level differences in data representation and is provided by standards such as XML, SOAP, UDDI, WSDL, etc. Structural interoperability involves schematic heterogeneity among information sources (Heiler,1995). Significant research and progress has been carried out to achieve syntactic and structural interoperability but comprehensive solutions to semantic interoperability, remain an active research area.

Semantic interoperability can be achieved by resolving the ambiguities and heterogeneities in terms and concepts thereby providing more useful information (Park,2004). Ontology and Taxonomy Coordinating Working Group defines semantic interoperability as the ability of two independent programs with reasoning capability to come to the same conclusions from the same data (SI,2005). This is a more relevant definition in the context of Autonomous Semantic Grid. Semantic interoperability ensures that communicating individuals or organizations share common understanding of meanings of requested service and data (Heiler,1995).

XML provides syntax for exchanging data among entities. It separates contents from presentation so it is easier to reformat the data. The explosion of industry and organization specific Document Type Definition (DTD) poses great challenges for integration of heterogeneous resources. To overcome these challenges, the Semantic Web tries to address these issues by incorporating semantic annotations so that the intelligent agents can utilize them to their full potential.

Ontologies provide semantic descriptions for Web services related concepts and can enable diverse business entities to be integrated with software agents and will lead to automation (Pasha et al.,2006).

## Semantic Web



The Semantic Web is the extension of the World Wide Web that provides a common framework in order to allow data to be shared and reused across application, enterprise, and community boundaries.

It is important to have a standardized communication mechanisms or language that can be reused for information sharing on the Web.

Web Ontology Language is a World Wide Web Consortium recommendation for representing knowledge on the Semantic Web (Dean et al.,2004). The foundation stone of Web Ontology Language is based on Description Logic which explains the constructs for defining restrictions, expressions and constraints on properties. FIPA Semantic Language is the standard content language used in conjunction with FIPA Agent Communication Language, specified by IEEE FIPA based on modal logic.

Researchers in agent and Grid domains are trying to address the issues that are raised in developing open systems, by their own perspectives. Distributed computing gives rise to concerns like complexity, dynamism and the characterization of entities that can play vital roles in the development of open distributed systems (Park,2004). Both Grid computing and Multi-agent Systems are emerging technologies and each of them can benefit each other to overcome issues like scalability in Grid applications and the limitation of the agents to be implemented as part of distributed systems. The main objective of the Grid community is to provide resource sharing and solving problems through coordinated behaviour in virtual organizations (VOs). Middleware for the Grid systems provides the necessary protocols and specification in order to utilize the basic characteristics of services like registration, discovery, invocation etc. The progression of the Grid system includes:

- Earlier ad-hoc solutions were provided
- Standards and specifications depending on Globus Toolkit (GT)
- Open Grid Services Architecture (OGSA)

## Virtual Organizations (VOs)



Virtual Organizations are a dynamic collection of individuals, institutes, resources that have well defined, flexible, secure, coordinated resource sharing mechanisms.

The Grid Services Architecture is adopting the standards of Web services to integrate service descriptions with the Grid systems and the invocation would be done by their interface. The ever growing research in the field of Grid technology has led to the evolution of Web Service Resource Transfer, that specifies a uniform way for defining, monitoring and accessing Grid services which can be available anywhere around the globe and thus is a crucial task most of the time. Most of the Grid technology research communities are trying to define the standards for Grids that can support open systems like Globus Toolkit.

The latest trends in Web technologies and continuous research in the field of distributed systems is getting attention by most of the organization to use electronic commerce and improve their businesses (Fensel,2003).The role of agents can be of great importance for such systems as they provide a well defined communication infrastructure, prone to changes and able to take immediate actions. It is very important to take into consideration that whenever an application for e-commerce is developed it must have the mechanism to provide a shared knowledge for a particular domain; this idea could be supported through the use of ontologies for adding semantic descriptions and can be of great importance in many ways:

- Utilizing ontologies for incorporating semantics in agents will have a large effect on how the information can be used by enterprises in a more meaningful way.
- Semantic descriptions can be used for defining the service interfaces and can make the services easily discoverable and interoperable without much effort.
- Agents can utilize the ontologies for semantically based interaction with one another.

Semantic interoperability between agents and Grid services leads to a number of tangible advantages. Agents will be able to perform search or consume grid services. It will also result in agents being more robust and integrating information from multiple heterogeneous semantic services (Negri et al.,2006). For example, a travel-planning agent might need to

access data from airlines, hostels, rental car companies, weather sites and tourist sites, each of which may have different ways of representing the relevant semantic information. Such an agent would face problems of translating different vocabularies and representations for this data into a common format. Ontologies, therefore, provides means for overcoming heterogeneities in vocabularies of these independent domains and can facilitate the interoperability.

Though the Grid and the Semantic Web are considered to be disparate domains, at points both of these technologies have some shared interests. The purpose of these technologies is to operate at a global level, with multiple players and heterogeneous technologies and thus

### Semantic Grid



The Semantic Grid is an extension of the current Grid in which information and services are given well-defined meaning, better enabling computers and people to work in cooperation.

they both need to have readily accessible resources and shared knowledge for the service provisioning. The Semantic Web can support the Grid services for describing meta-data for its base and high level services and for the representation of knowledge that can be used by its applications.

The aim of the Autonomous Semantic Grid endeavor is to take an initiative in order to develop effective methods for enabling such complex resource sharing. The key to this is an infrastructure where all resources, including services, are adequately described in a form that is machine-processable, i.e. knowledge is explicit - in other words, the goal is to provide semantic interoperability, based on the technologies of the Semantic Web. The next section highlights the importance of semantics in a system like the Grid.

## 2.2 Semantics for Grid Computing

### 2.2.1 Enabling semantics in Grid systems

It is of great importance that a service should be properly described by using the proper meta-data so that it is accessible to requesting entities and can be invoked without much effort and is interoperable if required. UDDI is a very well known solution in the current systems for searching the appropriate services by users, and the services are classified on the basis of the functionality that they provide. Reasoning capabilities play a vital role when services are classified into hierarchies or a particular Web service is retrieved.

Currently the semantics can be associated with the grid services by adding naming conventions to a WSDL document for the port, service type field respectively and then linking it to a specification document afterwards. Bridging the gap between the Semantic Web and Web services have already attracted attention, for instance DAML+OIL, which is a predecessor of Web Ontology Language has been explored in myGrid (myGrid,2007). The Grid services contain the provision for the service instances to be formed and destroyed and maintain their configurations over a period of time. However, this is currently a challenge for the Web services as they do not have this provision. Ontologies can provide a great support in addressing these issues.

### **2.2.2 Amalgamation of Information**

During the scientific experiments, it is sometimes a necessary issue to integrate information from distributed, autonomous resources in order to respond to a particular query posed by some scientist. In the field of genomics, for example there are thousands of data stores containing the details and structures of genes which might need to be checked for a particular gene based on the information that has been retrieved from these diverse data stores. The access and retrieval of data is currently supported by the Web and the services for data retrieval in Grid technology but the next milestone to overcome is the semantics of the data that is being processed and not just the availability of data. The semantics annotations will resolve the differences in semantics between diverse technologies and can enable the automation of information integration.

Integration of information is not just restricted to the experiments that are being conducted by the research community. Due to the current technological advancements in distributed systems like the Web and grids, the resources that are available on the Web can be retrieved and accessed by users transparently without knowing that a number of services were composed in order to obtain those results. Resource Description Framework (RDF) provides the support to store data from disparate sources and combine them in a uniform way so that it can later be used by some applications. Various research communities have been focusing on these issues as individual problems and solutions for them.

Sometimes shared knowledge for a particular domain through the use of ontologies can be used to overcome the interoperability issues that are present in the form of structural differences in databases or conflicts in the input and output parameters of services. Services utilize reasoning capabilities in order to ensure that the composition process is valid or not. Therefore, the semantically enabled services can be used for the following reasons:


- categorization of resources which can be computation intensive or storage resources etc.



- mapping set of parameters i.e. input or output associated with a service.
- choosing appropriate problem solving techniques.
- providing access to resources and monitoring their usage.

The aim of this research is to provide a mechanism for providing the interoperability in distributed systems through the use of semantics and keeping into account the fact that the existing standards are not violated during this transformation. Our earlier work in this context is the AgentWeb Gateway (Suguri et al.,2004) (Shafiq et al.,2005) that addresses the interoperability issues between these technologies at structural and syntactical levels. The limitation of our earlier work was the focus on structural and syntactical interoperability. To do so, in this thesis I try to address the issues raised in developing system architecture for semantic interoperability between agents and Web services for the translation, registration and querying of translated ontology.

Resource Description Framework (RDF)



Resource Description Framework (RDF) is a standard, developed by (W3C), for representing information about resources. In RDF, a pair of resources (nodes) connected by a property (edge) forms a statement: (resource, property, value), often called an RDF triple.

## 2.3 Research Questions

The research questions that will be addressed in this thesis to address the hypothesis are as follows:

Is it possible to integrate agents and Web services without modifications to their standards? What will the role of ontologies be in integration of these technologies? These questions have been further divided in to sub-questions that need to be answered as well:

- How will the agents communicate with the Web services?
- Can the agents and Web services communicate without violating standards?
- Who will be responsible for registering the services in agents and Web services?
- How will FIPA ACL and OWL translations take place?
- How will the delegation of work take place when a query comes in?
- What are the steps involved from user request to service retrieval?

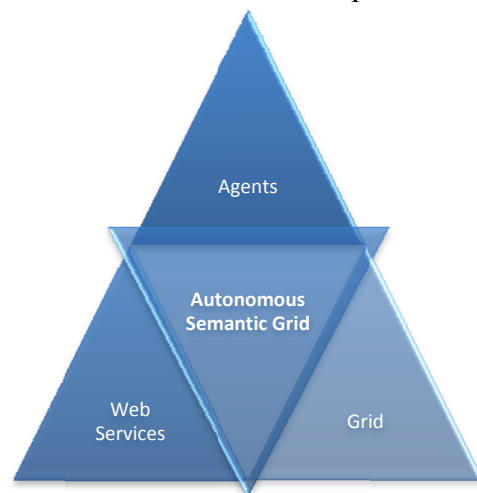
- How will the negotiations take place between the agents and Web services once a particular service has been invoked?
- How will the use of ontologies provide a solution so as to have a semantically enabled communication?
- Why use a formal language for system specification?

In order to answer these questions it is important to have an idea of the technologies that are involved and are, thus, briefly described in the next section.



## 2.4 Synergy of Technologies

The Web is evolving toward machine-readable infrastructure for sharing knowledge among humans, as well as machines, as previously discussed. Application designers building systems that harness Web services are facing many of the same issues that designers of agent systems have been tackling for more than 30 years. We believe that agents have much to offer in overcoming some of the inherent difficulties when dealing with complex and dynamic environments. As pointed out, such an environment requires autonomous and conversational components and agents can offer these capabilities. Agents are able to decide at run-time which resources and/or service to use for a particular task.



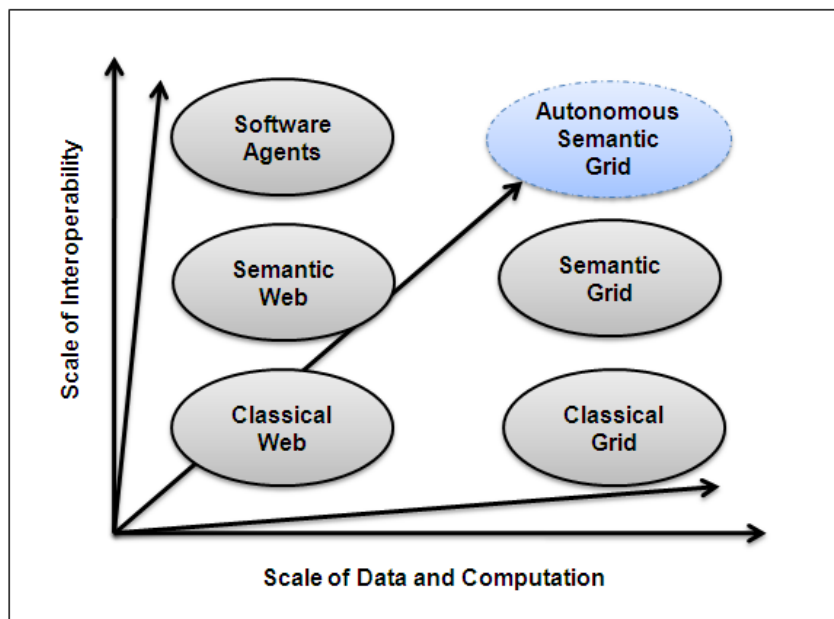
**Figure 2.1:** Autonomous Semantic Grid

In open systems, the structure of the system itself can dynamically change. The characteristics of such a system are that its components and resources are not known in advance, can change over time, and may be highly heterogeneous. The best-known example of a highly open software environment is the Internet. The functionality is almost certain to require techniques based on negotiation and cooperation, which lie very firmly in the domain of agent systems. The machine should not just act as a dumb receptor of task descriptions, but should cooperate with the users to achieve their goal.

These considerations give rise to the idea of an agent acting as an expert assistant or delegate with respect to some applications, knowledgeable about both the application itself and the user, and capable of acting with the user in order to achieve the user's goals. The Grid community has previously focused on interoperable infrastructure and tools for secure and reliable resource sharing within dynamic and geographically distributed organizations.

In contrast, those working on agents have focused on the development of concepts, methodologies, and algorithms for autonomous problem solvers that can act flexibly in uncertain and dynamic environments in order to achieve their goals (Ferber,1999). In (Foster et al,2004), the authors explicitly identify the requirements for the integration of software agents with the Grid environment. Pioneer work has been carried out at our research group in this area; we have proposed the same requirements in our work as early as 2003 (Ahmad et al., 2003).

Our vision of the integration of agents with Semantic Web and Grid computing is to lay foundation for self-regulating system, namely Autonomous Semantic Grid as shown in Fig. 2.1. In the proposed system higher level interaction is pivotal for the creation of virtual organization that exhibits characteristics of Autonomous Decentralized System (Mori, 1993). Theoretical foundations of Autonomous Decentralized systems rely on the principles of autonomous controllability and autonomous coordinate-ability. These two properties assure online expansion, fault tolerance and online maintenance of the systems. Autonomous Decentralized system can be realized through software agents, which are autonomous, proactive, and goal-oriented problem solving entities with social ability for high level interaction.



**Figure 2.2:** Synergy of software agents with Web services and Grid computing  
Revised from: Norman Paton (SG, 2005)

The European Union project “From Grids to Service-Oriented Knowledge Utilities: A critical infrastructure for business and the citizen in the knowledge society” (SOKU) has very similar requirements and goals as envisioned in Autonomous Semantic Grid. The Service Oriented Knowledge Utilities vision highlights three key concepts, i.e. Services, Knowledge and Utility. Services orientation is about instantiating and composing services dynamically. Knowledge will be assisted by semantics while utility is directly and immediately useable service with established functionality. IBM’s Autonomic Computing is another example of similar project inspired by biological systems (Baresi et al, 2006). Ian Foster et al use Virtual Managed system terminology close to the Autonomous Semantic Grid vision (Keahey, 2006).

Figure 2.2 illustrates the role of the Autonomous Semantic Grid with the other technologies; the technologies vary in terms of interoperability and diversity along the Y-axis and in terms of data along the X-axis and the Autonomous Semantic Grid lies at the top of them sharing almost all the capabilities of these technologies. In order to realize Autonomous Semantic Grid vision, it is important to understand these technologies that will play a vital role and can benefit from this vision in the long run.

## 2.5 Summary

This chapter specifies that the semantic interoperability between agents and Web services is the key to the vision of Autonomous Semantic Grid. In order to have a clear understanding

of the problem domain the first step is to understand the context of semantic interoperability and how different researchers have perceived it. Although a number of interoperability issues have been defined in the information systems from structural, syntactic and semantic aspects but the nature of the problem addressed in this thesis is closely related to the semantic interoperability issues. Since the ongoing discussion is about different technologies like Grid computing, agents and Web services therefore the importance of semantics cannot be overlooked in all of these domains. All of these technologies consider the semantic aspects in their own way, which has been discussed in this chapter. Based on the hypothesis presented in chapter 1 an overall discussion of the problem domain and list of research questions has been explicitly stated in this chapter. By addressing the research questions step by step the proposed solution has been devised and is presented in the later chapters of this thesis. Although the main concerns addressed in this thesis are related to achieving successful interactions and communications among agents and Web services. The next chapter will present the overview and ongoing research in the relevant domains.

# Chapter 3

## Literature Review

---

This chapter will examine the core technologies and the research undertaken in the thesis and will illustrate how it is related to the research performed by others. This will be achieved by understanding the background knowledge. The aim is to focus on the technologies that work together in a synergic way for dealing with the issues that are unveiled in the previous chapter. This chapter presents the concept of agents, which are the key entities in the proposed solution and how these agents communicate when working in a Multi-agent system. FIPA compliant Multi-agent systems will be explained in more detail with their architectural details as they are integrated in our solution. As the study is to achieve the vision of Autonomous Semantic Grid, research progress in technologies like Grid computing, Semantic Web and Web Services is also presented in general. However, the focus has been the convergence of interest in these technologies where each of them can benefit from one another. An overview of the state of the art solution has also been discussed that focuses on the most relevant advances to address the communication concerns between agents and Web services with a focus on limitations of technologies to cope with the problems identified in this thesis.

Information and knowledge are becoming the focus of the economic activity of the entire world. Global information systems, such as the Internet and the World Wide Web, are no longer just pathways for digital data rather they generate new information and knowledge for commercial activity, education, business and research. For the first time, in the history of human kind, non-material resources such as software and information services have become the real wealth of a knowledge-based society. It poses a great challenge to process this extremely large set of data and extract new knowledge autonomously. A number of new technologies such as Grid computing, Semantic Web and software agents are emerging that can help to cope with this challenge through the synergies of their integration (Ahmad et al.,2003) (Foster et al.,2004) .

### Software Agents



Agents can be defined as a piece of software that is an autonomous problem-solver and is capable of effective operations in dynamic and open environments.

### **3.1 Software Agents**

Agents can be defined as a piece of software that is autonomous problem-solver and is capable of effective operation in dynamic and open environments (Wooldrige.,2002) (Singh et al.,2005). Moreover an agent can be considered as a smart software system that has the capability of performing autonomous actions on behalf of its user or owner. Agents in a Multi-agent System will be coordinating or acting on behalf of users or owners with very difficult goals or motivations. Agents embody a stronger notion of autonomy than objects, and in particular, they decide for themselves whether or not to perform an action upon the request of other agents.

The agent is the fundamental cell of the distributed intelligent. Hence, its artificial intelligence (AI) can be built by using all AI implementation technology currently known and it can compute regardless of its current location. Another property of the agent is to switch their locations by means of migration. In this manner, they are able to do their computing in an environment, which provides more computing power.

However, the primary feature of agent technology is the agent's ability to communicate with each other. This enables the agents to unite their efforts to become a collective of working individuals, who are aware of each other's goals and intentions (Ferber,1999). Moreover for communication a mechanism, is required that can serve as shared vocabulary between communicating agents and the ontologies. Software agents today have a vast number of applications varying on the type of functionality and services they offer ranging from e-mail, auctions, to distributed systems etc. As the main focus in this thesis is to deal with technologies like Web services and Grid computing as principal candidates for agent application, the importance of agents in these technologies is mentioned next.

#### **3.1.1 Personalizing Web Services through Agents**

In (Baader et al.,2003), the authors answer the commonly raised issue of how agent technology can be used to personalize Web services. They have highlighted a number of critical issues that web service and agent platforms must address in order for two paradigms to work together, and propose an automated component that can be integrated with existing web service infrastructures. Particularly, they are trying to address the challenge of how a consumer can assign a particular job that will be delegated programmatically and will autonomously be interacting with the Web services according to context when acting on the behalf of a consumer.



At present, the Web services are used to define the behavior of the service. The arguments to utilize the service must be known in advance, and should be explicitly registered before enactment. Furthermore, the protocol must be defined to precisely match the Web services Description Language definition of the Web service. It will, thus, reduce the restrictions, and allow a more extensible coordination, of web service discovery and invocation. They intend to semantically annotate web services, on which one can reason about the behavior of the services.

### Multi-agent Systems (MASs)



A Multi-agent system is a collection of several interacting agents in which each agent has incomplete information or capabilities for solving the problem.

Whereas (Walton et al.,2004) explores the option of utilizing software agents for the construction and enactment of e-Science applications. The agent capabilities are explored for the immediate construction and verification of experimental data. Moreover, they try to address the issues of service discovery.

## 3.2 Multi-agent Systems (MAS)

Software agents can work independently as well as in the form of groups forming communities of multiple agents for achieving particular goals. These agents can then cooperate with one another, normally by exchanging messages through some computer network infrastructure and are known to be Multi-agent systems. Multi-agent Systems are one of the appealing technologies in software-based framework that offer a friendly environment in support of agents for providing constant services. Multi-agent Systems provide proper execution environment to agents so that they can assure the provision of services to other agents by cooperating, coordinating, and negotiating.

Multi-agent Systems are a relatively new sub-field of distributed systems. Since the beginning of study on Multi-agent Systems in 1980, they have gained a lot of popularity and recognition. The idea of Multi-agent Systems is not tied to a single application domain rather they can be adopted to host different application domains. One of the reasons behind the popularity of Multi-agent Systems is the fact that agents can exploit the opportunities

presented by massive distributed systems such as the internet. Multi-agent Systems represent virtual societies where software entities (agents) acting on behalf of their owners or controllers (people or organizations) can meet and interact for various reasons (e.g., exchanging goods, combining services, etc.) and in various ways (e.g., creating virtual organizations, participating into auctions, etc.)

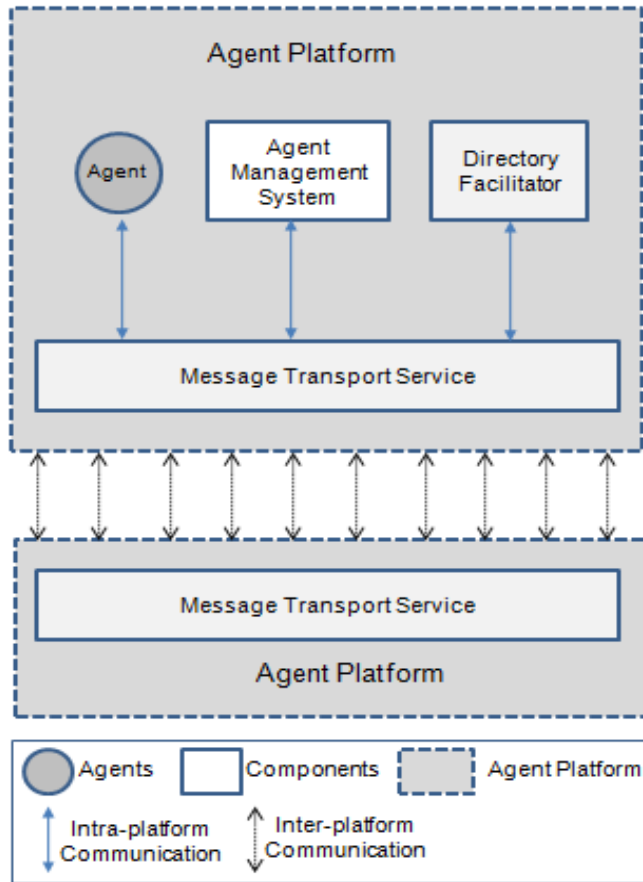
#### FIPA Architecture Reference Model



The FIPA Architecture Reference Model is a standardized view of an Agent Platform specified by FIPA along with the required components for its execution.

### 3.3 Agent Platform

Software agents provide multiple services to its users. For the provision of these services, software agents require a proper execution environment in which they can execute themselves and keep themselves ready for service provision. Such an execution environment in which software agents can be created and can behave according to their specification is called “Agent Platform”. Many Agent Platforms provide an environment for the community of agents for the provision of dynamic services. Figure 3.1 depicts the FIPA reference architecture model of a typical agent platform and the required components.



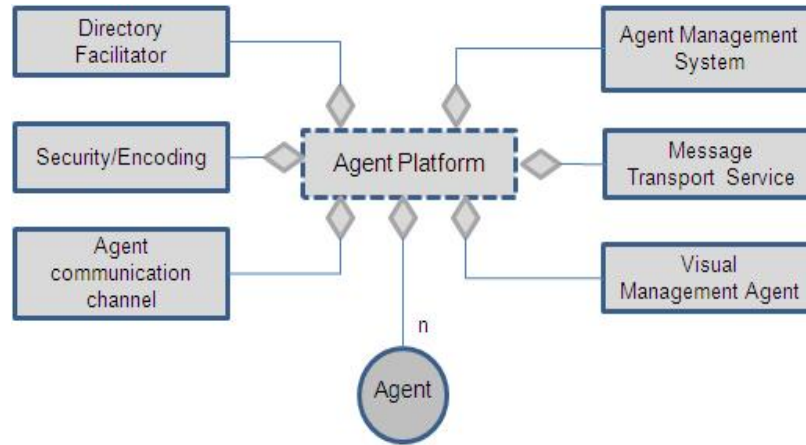
**Figure 3.1:** FIPA Architecture Model (FIPA, 2002)

### 3.4 FIPA (Foundation for Intelligent Physical Agents)

This research mainly focuses on the involvement of standards; therefore, we keep FIPA as a standard governing body and reference model, which will provide rules of an abstract architecture for Multi-agent Systems developers to follow in order to be coherent with the solution proposed in this thesis. The main objective of FIPA is to develop software standards for heterogeneous and interacting agents and agent-based systems. FIPA focuses on the interpretability issues of Multi-agent Systems. FIPA has standardized a few components within Multi-agent Systems. In order to be called FIPA compliant, implementation of some of those components is mandatory. Figure 3.2 depicts the architecture of a platform according to FIPA specifications. A typical agent platform requires several other components along with the basic building blocks previously mentioned in the FIPA reference model and are as follows:.

- Agent Management System (AMS)
- Directory Facilitator (DF)

- Message Transport Service (MTS)
- Agent Communication Channel (ACC)
- Agent Communication Language (ACL)
- Encoding Services
- Visual Management Agent (VMA)



**Figure 3.2:** Components of FIPA compliant Agent Framework

### 3.4.1. Agent Management System (AMS)

Expected growth of Multi-agent Systems (MAS) with a community of social agents in heterogeneous applications has made it focal point for research. The Agent Management System (AMS) (Ghafoor et al, 2004) (Khan et al, 2005) is responsible for managing all the agents within a Multi-agent Systems and is considered to be the governing authority for any agent system. A single agent platform can be distributed over several machines which provide scalability and load balancing etc.

One of the major reasons for not deploying Multi-agent systems frequently is the absence of fault tolerance. If the Agent Management System fails then the whole distributed system will exhibit abnormal behavior. Failure of Agent Management System leads towards abnormal behavior in the distributed platform.

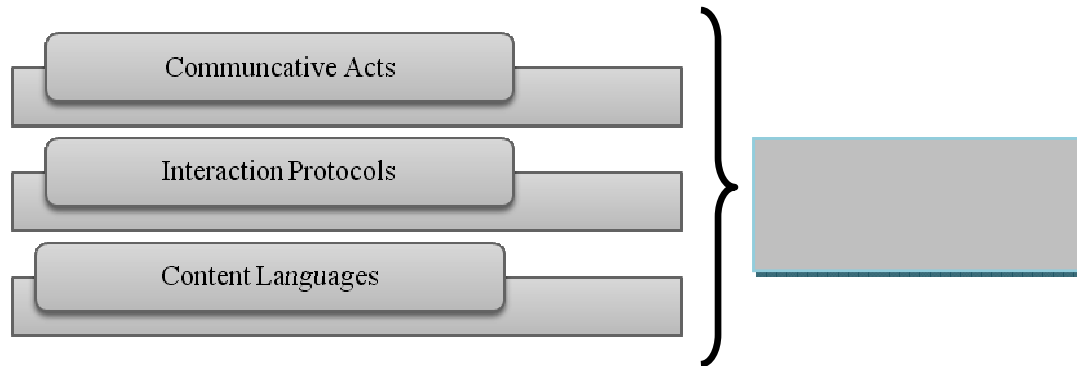
### 3.4.2 Message Transport Service (MTS)

Message Transport Service is the backbone of any Multi-agent System and is more related to the research in this thesis. MTS embodies the communication at the three levels that is communicative acts, interaction protocols and content languages. It supports the sending and receiving of Agent Communication Language messages between agents. The agents involved

may be local to a single Agent Platform or on different Agent Platforms. Figure 3.3 represents the various layers of Agent communication language.

Two modes of communication are involved for message transportation.

- Intra-platform Communication
- Inter-platform Communication



**Figure 3.3:** Agent Communication Language

### 3.4.3. Agent Communication Language (ACL)

The Agent Communication Language package is responsible for the creation of a message that is understandable by all entities involved in the Multi-agent system. Through this package all agents will create a message through some pre defined rules. And the message will be sent to the required destination. At the reception end, the agent will take its own decision based on the Agent Communication Language message.

Agent Communication Languages provide agents with a means of exchanging information and knowledge, which is really the essence of all forms of interaction in Multi-agent systems. The resulting language is FIPA ACL. FIPA ACL is a protocol for specifying message format and defining descriptions of their pragmatics that is the communicative acts or intentions of the agents. Furthermore, FIPA also define semantic languages to successfully communicate with each other. FIPA published Semantic Language which provides rich semantics. Every agent has common semantics to talk to each other based on shared ontology.

### 3.4.4 FIPA Semantic Language (FIPA SL)

The FIPA SL language is a human-readable string encoded (i.e. a content expression in FIPA SL is a string) content language and is probably the mostly diffused content language in the agent research community. FIPA SL is principally designated in open systems where agents developed by different communities executing in heterogeneous environments want to

collaborate with one another. Therefore it is important that FIPA SL should support the following requirements:

1. It should be able to represent propositions;
2. It should be able to represent actions;
3. It should be able to represent objects, including identifying expressions to describe objects.

To allow agents, or a group of agents to perform some tasks, with minimal computational burden, semantic and syntactic subsets of the full FIPA SL content language have been proposed. These subsets are defined by the use of profiles, that is, statements of restriction over the full expressive power of FIPA SL. These profiles are FIPA SL0, FIPA SL1 and FIPA SL2 in increasing order of expressivities.

### **3.4.5 Directory Facilitator**

Directory Facilitator (DF) is another component of a Multi-agent system. It is accountable for providing a yellow-pages directory service to other agents. Agents may register their services to the DF or query the DF to find out what services are offered by other agents. An agent is responsible for providing information related to the service, i.e. `service_type`, `service_name` etc. FIPA imposes that each Agent Platform has its own DF that is known as default DF. Other DFs may also register with default DF to create a federation.

### **3.4.6 Visual Management Agent (VMA)**

VMA is not a component specified by FIPA however a typical agent platform should be able to support the handling of agents by defining them. VMA is an agent that offers a graphical interface to platform administration and platform monitoring. The agent offers many services that show the state of the Agent Platform and it also offers various tools that are used to perform administrative interaction with the AMS agent, the DF agent and are also used to debug and test applications.


The state of the agent platform also shows the details of the agents that reside inside the platform. The VMA itself offers some internal agents for platform management and monitoring that can be used to perform different tasks such as:

- Examination of the message exchanges among different agents;
- Create or compose ACL messages and send them to other agents;

- Display the list of all the ACL messages sent or received by the agent;
- Read and save ACL messages from/to file;
- Create ontologies graphically.

VMA also provides graphical interface for the administration of the Directory Facilitator and Agent Management System. Because VMA is an agent, therefore it would communicate with an AMS agent and DF agent through passing ACL messages. For the creation of ACL messages a VMA package will use an ACL package and will compose an ACL message. After that the ACL message will be sent to the Message Transport Service that will forward that message to the respective agent.

Grid Computing



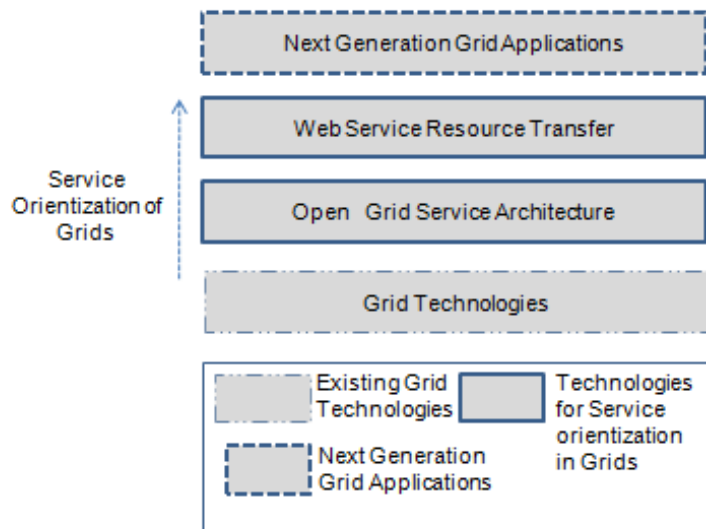
Grid Computing is a technology that enables resource sharing and coordinated problem solving in dynamic multi-institutional virtual organizations.

### 3.5 Grid Computing

The Grid is an infrastructure that permits flexible, safe, and harmonized sharing of resources between dynamic groups of entities and organizations which are referred to as Virtual Organizations (VO) (Foster,2001). VO can be defined as a group of individuals or institutions who share the computing resources of a "grid" for a common goal. Grid middleware provides the basic guidelines for the mediation among various layers of Grid in order to make use of service registration, retrieval and invocation and utilize the shared resources. It is analogous to the electric power grid in that it can potentially provide a universal source of IT resources. Grid is envisioned to provide a universal source of computing power as utility and this can potentially have a huge impact on human capabilities and on the entire society.

Figure 3.4 depicts the next generation Grid applications from their inception and how they are evolving to service orientation. Open Grid Forum has been continuously doing research in defining standards and specifications to amalgamate Grid systems and Web services so that they can be deployed at a broader level and are trying to utilize the W3C standards like WSDL (WSDL,2001), SOAP (SOAP,2007) and UDDI (UDDI,2005) for interoperability. Open Grid Service Architecture (OGSA) specifications are the result of their continuous

efforts. OGSA supports creation, termination, management, and invocation of transient services as named managed entities with dynamic, managed lifetime via standard interfaces and conventions. Web Services Resource Framework (WSRF) has been proposed to implement OGSA (OGSA,2003). WSRF provides a set of operations that Web services implement in order to become stateful (Tuecke et al.,2003). Web Service Resource Transfer (WSRT) is further enhancement in WSRF to cope with WS-Management (IBM,2007). WSRT not only entails the definition of the resources but also provides access mechanism to them along with the functionality that was previously provided by WSRF.



**Figure 3.4:** Stack view for service oriented Grids

Moore’s law of integrated circuits (Moore,2005) and Edholm’s law of bandwidth (IEEE, 2004) are well known. Moore’s law gave rise to extremely high capacity circuits, such as large storage and powerful processors. Regardless of all these efforts, making supercomputers to meet increasing demand of data processing have been infeasible mainly due to cost factor. According to Edholm’s law increase in communication bandwidth facilitates efficient transmission of bulk data. The major focus of all research in Grid computing is to exploit these two factors for pooling resources and realization of alternate to costly supercomputers for handling huge amount of data processing demand globally.

Obviously, great disparity exists between the growth rate of information generation and its processing and effective management. Grid systems are complex in nature and require continuous involvement of human input in order to perform the specified tasks. It cannot handle these emerging data-intensive applications effectively. Therefore, a fundamental requirement to process such a scale of data is to redesign Grid so that it can process these kinds of information autonomously without human intervention. This autonomous information processing should lead to new information and knowledge generation. In the



next section, we summarize the Semantic Web, which is a foundational effort for such autonomous processing of data and information.

Grid services should be able to provide novel means composed of distributed services in a transparent way. The aim of the future services in Grid systems is to reuse the on hand components and services in a way that they can coordinate with one another in a flexible way. Keeping this trend in view, the Grid community is gradually moving toward the service centric approach in the form of specifications by Open Grid Service Architecture (OGSA,2003), this will integrate the service based systems with the requirements and methodologies of Grid systems, but this integration will require a significant extension of Web services with the grid services as the Grid services have the following properties:

- distributed and changing;
- there is no central governing body;
- services can be composed of other services and might spawn hundreds of other services;
- an experiment or simulation can take a huge amount of time for processing.

These issues greatly require addressing factors like availability, correctness and security concerns. Till now Web services are most of the time available and do not maintain their state whereas grid services are stateful and are short lived.

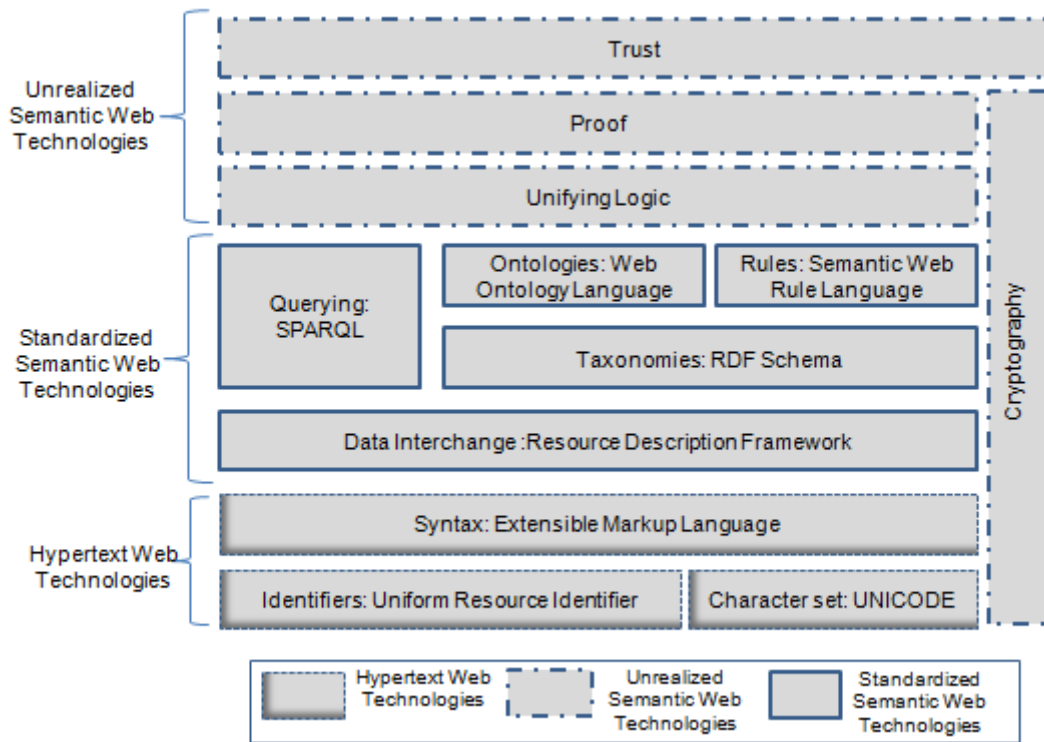
### **3.6 Semantic Web**

Web services have emerged as a standard for Web-based technology for exchanging information on the Internet (Nassuni,2003). These are modular, self-describing, self-contained applications that are accessible over the Internet. By definition, a Web Service is an executable program function that can discover and invoke any service on the Web (Martin,2007). Web services utilize Internet protocols for the registration, discovery and invocation and are language neutral and support heterogeneous platforms by providing transparent access to users.

Web services architecture defines three major roles consisting of service provider, service requester, and service broker (Papazoglou et al.,2006). The Semantic Web is the extension of the current Web (Lee,2001) that aims at marking web contents with richer metadata so that they can be processed by machines without human intervention. Resource Description Framework (RDF) and Web Ontology Language (OWL) are some of the languages to represent the contents with semantics (Nassuni et al.,2003). The technology stack of Semantic Web is shown in Figure 3.5. The development of ontologies could be the key to

overcoming the semantic heterogeneities by explicitly defining machine processable semantics.

Semantic Web Services (SWS) can be defined as Web service with the semantic language representation of services for efficient discovery, composition and execution in an open and distributed environment (Paolucci et al.,2003). Many research groups have put forward efforts to automate and standardize Semantic Web Services. DAML-S (DAML-S, 2002) and OWL-S(OWL-S, 2003) are well known initial efforts for the creation of Semantic Web Services. Current notable ongoing projects for Semantic Web Services (Martin,2007) realization are Semantic annotations for WSDL (SAWSDL, 2007), Semantic Web Services Framework (SWSF, 2005), and Web Service Modelling Ontology (WSMO, 2006). A great deal of research has been carried out for semantic web services and many challenges are ahead (Papazoglou et al.,2006). It is important to highlight that semantic technologies provide partial solutions to the information explosion problem. It gives well defined meaning but to process such scale of data we need to have appropriate Grid system architecture. SWS are the basic building blocks for realizing Semantic Grid vision.



**Figure 3.5:** The Semantic Web Stack (Lassila, 2005)

The Semantic Web initiative (Lee,2001), (W3C,2004) that addresses the problem of XML, lack of semantics by creating a set of XML based languages, also relies on ontologies that explicitly specify the content of the tags. The Web Ontology Language (OWL 2.0) is a W3C recommendation for such a language that supersedes the earlier DARPA Agent Markup language (DAML+OIL) (DAML,2001). OWL is an extension to XML and the Resource Description Framework (RDF) enabling the creation of ontologies for any domain and the instantiation of these ontologies in the description of resources. The OWL-Services language (OWL-S,2003) is a set of language features arranged in these ontologies to establish a framework within which the Web services may be described in this semantic web context.

The OWL-S ontology is conceptually divided into three sub-ontologies for specifying what a service does, how the service works and how the service is implemented. The *Profile* describes what the service does so that it can be discovered at matchmaking time. It contains the contact information of providers, an extensible set of features that specify characteristics of the service and a functionality description by specifying the inputs, outputs, preconditions and effects (IOPE s) of the service. The *Process* presents the internal working of the service in terms of the internal processes, their process model and the internal dataflow; whether each service is either an atomic process that is executed directly or a composite process that is a combination of other sub-processes. The *Grounding* specifies the operational level details of the service by linking the conceptual level descriptions to the WSDL description of the service.

### **3.7 Semantic Grid**

The Semantic Grid community is trying to resolve the issues regarding automation and ability of dealing with dynamic requirements in Grid services. Autonomy can be incorporated into Grid systems somehow by associating semantics, which will facilitate machines to comprehend the information and produce more useful knowledge while processing enormously huge data. The Semantic Grid is considered to be built on existing Grid systems where the services are defined in a well understandable way, better machines and people can collaborate with one another through sharing resources, and the available services (SG,2007). For this kind of environment, it is a key aspect to provide necessary information for the user requirements and applications. Moreover, there should be a proper mechanism for representing the providers and the resources they offer which can be software or hardware. These resources can then be used by the advanced services to efficiently utilize the Grid systems to their full potential.

It is very important to describe a service in a way that can facilitate automated discovery, selection, matching, composition and interoperation, invocation and execution. To find a service according to the requirements of the user it is an important consideration that the services should be characterized by the type of functionality they provide and this approach

is widely accepted till now i.e. UDDI. It is also important that an associated semantic description with a service should be consulted when making service matches between the two services. Condor (Condor,2007) provides a search facility in order to find the computation specific resources, which is used to choose computational resources where the environment is dynamic and the services are continuously changing.

Moreover, the creation of virtual organizations requires higher level abstraction with conversation and interaction characteristics provided by software agents.

### **3.8 Web Services**

A Web service is a self-describing, self-contained application logic that provides some kinds of business functionality to other applications through an Internet connection. Other applications can access Web services through standardized web protocols and data formats, such as HTTP, SOAP and XML. A Web service is language independent and its implementation is not visible to the user or application that calls the service, since all the interaction happens through standard web protocols. Any program that can process XML and communicate via HTTP is capable of calling a Web service.

The aim of the Web services endeavor is to obtain an environment where service customers and service providers can locate one another, connect with each other dynamically, set the terms and conditions of service invocation automatically and then execute the necessary actions according to the prevailing contract. To date, Web services architecture has been developed that consists of five layers for supporting service descriptions, publishing service descriptions and discovering services. The key advantages of these and related standards include interoperability between distributed applications regardless of the underlying platform, implementation language and operating system.

Given these advantages, it is clear that Web services have much to offer. Web services have promised to change the Web from a database of static documents to an e-business marketplace. The Semantic Web promises to make information understandable to a computer and Web services promise to provide smooth and painless integration of disparate applications. Web services offer a new potential of automation in e-Work and e-Commerce, where fully open and flexible cooperation can be achieved on-the-fly, with low programming costs. However, there are a number of shortcomings which prevent realizing the goal of smooth cooperation among heterogeneous service oriented systems.

### 3.9 Role of Ontologies

To have a meaningful conversation between two entities it is very important to have a joint and clearly defined set of vocabulary. An ontology can simply be considered as a taxonomy for defining terms, their relationships, the set of actions to be performed on them or maybe a combination of all these. Each Ontological schema is itself a complete structure of any concept, action, predicate or all of them and other schemas can be added into it. There could be schemas of primitive data types as well.

Different people have defined ontologies in different ways. One of them is considered to be very well known and was anticipated by Gruber, (Gruber,1993): though it is a general definition and according to him ontologies can be defined in specific contexts. As an example take agent technology into consideration. (Russell et al.,1995) ascertain that ontology is a formal depiction of the terms and relationships that can exist between the groups of agents. The significance of the ontologies is apparent from the subsequent definition: ontology is a hierarchically structured set of terms to describe a domain that can be used as a skeletal foundation for a knowledge base (Swartout et al.,1996).

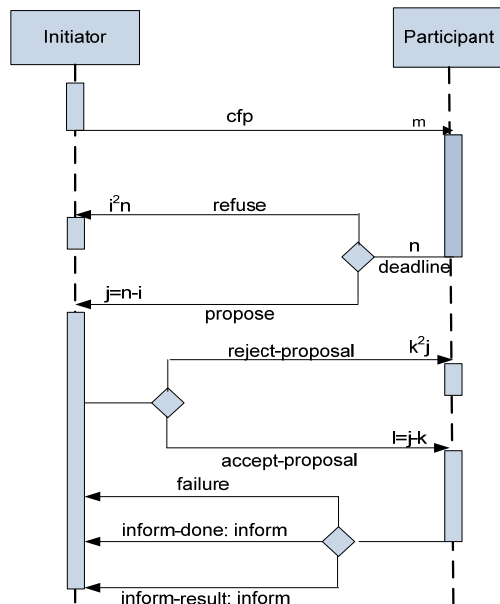
According to (Fensel,2003) Ontology is a formal, explicit specification of a shared conceptualization. From this definition four main concepts are identified: an abstract model of a phenomenon termed "conceptualization", a precise mathematical description provides the word "formal", the precision of concepts and their relationships clearly defined are expressed by the term "explicit" and the existence of an agreement between ontology users is hinted at by the term "shared". Keep into account the above mentioned descriptions of Ontologies we can characterize some of the important facts about them:

- A particular domain can be specified utilizing the capabilities of Ontologies;
- Through the use of ontologies we can define the concepts and their relationship with one another;
- A taxonomy based ideology is used to classify the concepts.

### 3.10 Negotiations between agents and Web services

Keeping in mind our long-term objective of interoperability between agents and web services the overview of OWL-S as a potential language for semantic Web services has been observed (Mcilraith et al.,2001), (Martin et al.,2004). With OWL-S markup of services, the information necessary for Web service discovery can be specified as computer-interpretable semantic markup at the service Web sites, and a service registry or ontology-enhanced search engine could be used to locate the services automatically (Sycara et al.,2004). Execution of a

Web service can be thought of as a collection of remote procedure calls. OWL-S markup of Web Service provides a declarative, computer-interpretable API that enables automated Web service execution. Figure 3.6 depicts the steps laid out by FIPA for negotiation entities when volunteering to engage in negotiations.



**Figure 3.6:** Negotiation Process specified by FIPA

Given a high-level description of the task by the user, automated composition and interoperation of Web service to perform the task is of particular interest to us. With OWL-S, the information necessary to select and compose services would be encoded at the service Web sites (Laukkanen et al.,2003). Software agents can be written to manipulate and interpret this markup, together with a specification of the task and thus can be bestowed with the ability to perform the task automatically (Sirin et al.,2003). Web Service Conversation Language (WSCL,2002) can be used to implement Contract Net Protocol (CNP,2002) for negotiation among Web Services. The flexibility of negotiation is far-off from that prevalent in the agent infrastructure. FIPA (FIPA,2003) provides detail specifications of Request protocol, Request/ Response protocol, CNP, English Auction, Dutch Auction, Brokering protocol, etc.

### 3.11 $\pi$ -ADL (Architecture Description Language)

$\pi$ -ADL (Architecture Description Language) (Oquendo,2004) (Oquendo,2005) is a formal language, with its primary focus on the software architecture description at the mathematical sense and the ability to reason about its behaviour.  $\pi$ -ADL formal foundations are based on higher-order typed  $\pi$ -calculus (Sangiorgi,1992) (Milner,1993). $\pi$ -ADL encompasses the architecture centric constructs for representing the structure, as well as behaviour details.

Behaviours and abstractions are considered to be top level constructs in a  $\pi$ -ADL program. Each behaviour defines a concurrent thread in execution, which means if multiple behaviours have been defined then there will be multiple threads in execution for each behaviour. However, an abstraction is a reusable behaviour template that can be invoked by other behaviours and abstractions. Variable and connection declaration are present in the body of behaviour or an abstraction. Connections are the bridging entities for connecting behaviours with abstractions and also to connect an abstraction with another one. Connections have the same functionality as that of channels in  $\pi$ -calculus. Connections are typed, and can send and receive any of the existing variable types, as well as connections themselves. Constructs like compose, choose and replicate are also existent in the language to support parallel processing.

In order to execute these  $\pi$ -ADL specifications a compiler is required that can support this execution.  $\pi$ -ADL.NET compiler (Qayyum et al.,2006)(Qayyum,2008) is a unique effort for the implementation of  $\pi$ -ADL on a .NET platform.  $\pi$ -ADL.NET is a domain independent compiler that provides the complete benefit of modelling and refinement of the software architectures.

### **3.12 State of the Art Solutions**

The previous sections of this chapter highlight the technologies, the ongoing research in respective technologies that will aid in the realization of Autonomous Semantic Grid in one way or the other. Taking into account the degree of flexibility and autonomy that can be achieved by integration of these technologies and that the long-term goal of this research is to propose architecture to establish communication among software agents and Web services for problem solving on a large scale. If an agent can read and interpret the semantic content, apply the logic or understand the mechanics of an entity on the Web through its attached ontology, then it may reason about, interact or modify its behaviour. Different research communities are trying to solve this problem from different perspectives. Based on this challenge a literature survey is presented to discuss the state of the art solutions that are the relevant for the problems identified in the first chapter.

#### **3.12.1 Web Services Integration Gateway (WSIG)**

An integration gateway known as Web Services Integration Gateway (Greenwood et al., 2004) has been proposed that provides the basic constructs for providing two way communications between the agents and Web services. It is an add-on that has been developed for the JADE (JADE,2006), (Spanoudakis et al.,2007) platform specifically.

WSIGS for Jade is an example of the integration where very fewer changes take place. JADE is a framework that is used to develop FIPA compliant Multi-agents system and is completely implemented using Java.

It is quite obvious that agents and Web services are different in terms of their technological structures. Therefore, WSIG proposes to add a specialized service entity that will resolve the issues of difference between the two and will then help to connect both domains. As the add-on is specifically developed for the agent system it is compliant to the FIPA specification and provides support for the communication constructs provided within the agent systems. However the vocabulary sets that are the ontologies and their role in the communication of agents with the Web services has not been considered. For the negotiation process the assumption has been taken that it is possible. However there is not much work done to address the negotiation process. WSIG is specifically built to support the JADE system and the agent can invoke a particular Web service by sending messages through the intermediary service. The main objective of this research was to target JADE as a software agent platform that can facilitate its agents to invoke Web service and vice versa in a transparent way and without much human intervention.

### **3.12.2 WS2JADE**

WS2JADE is a toolkit (Nguyen,2005) that has been developed for JADE agents to find services and utilize them. The key focus of the WS2JADE is to deal with the concerns more related to service description and interaction etc. In this approach the proxy agents that are residing in WS2JADE system are responsible for making the services visible to the agents and have many to many relationships with agents. The JADE agents are single threaded most of the times and can therefore allow the same web service to be accessed on different proxy agents, achieving a parallel access leads to the composition of relevant services. Searching in WS2JADE is performed by the proxy agents of the UDDI. This methodology uses the agents to capture the behaviour of the services and then these services are deployed and exposed as agents and can thus be discovered by other agents by using the conventional message passing techniques. This mechanism requires that all the existing services should be captured and deployed again in the form of agents from scratch.

This work, however, captures the details of the service descriptions and their interaction with one another using the proxy agents. How these services will be utilized by the agents in a more intelligent way rather than just processing the service descriptions is still an open issue. Though WS2JADE has demonstrated some good work while interacting with various Web services, still it requires a lot of improvement as it provides the integration in single mode (fulfilling one requirement at a time) and the issues for resolving the gap between the



stateful communications of agents with stateless communication of Web services is still unaddressed. WS2JADE look forward to resolving it using ontologies.

### **3.12.3 OWL -P**

OWL-P (Smith et al.,2002) is a proposed technology for providing the communication between the agents and the Web service standards. OWL-P is responsible for the specification and integration of commitment based protocols and has various components including the ontologies for the protocol specification, rule based representation and an algorithm for integration of the protocols and last but not the least a runtime environment. The aim of a runtime environment is to facilitate the registration and searching of the services and the execution of the protocol by sending and retrieving messages.

OWL-P can be considered not only as language but as a runtime environment that can support the agent and the Web services interaction keeping into account the agent preferences and can integrate various protocols. But the work has been done considering the standards of the Web services and follows the protocol combination scheme for the inter-communications of processes. Semantic Web Rule Language has been used for defining the rules and the software designer is responsible for defining the axioms, but how different protocol combinations and the heterogeneity issues raised have not been considered in their work.

### **3.12.4 Middleware Ontology Service (MOS)**

MOS (Khalid et al.,2007) is a solution provided to cater for the continuous support for ontologies in Multi-agent Systems. A dedicated agent is responsible for handling the incoming request for some information from a particular ontology within the agent platform. The work highlights the description of ontological representation, storage and querying and supports these operations to some extent. The system provides the support for agent communication systems using ontologies in an agent platforms but the work does not highlight the potential need of combining agents and Web services and how the translations will take place when the agents want to communicate with the Web services. Similarly the fact that agents can sometime engage themselves in a negotiation process in order to achieve a particular goal has also been overlooked. No proper mechanism has been devised as how to communicate with the outside world that is the Web services. Due to this limitation the agents cannot engage themselves in any sort of negotiations.

### 3.13 Comparison of State of Art solutions

Table 3.1 summarizes a comparison of related work across a number of criteria including that the extent to which these solutions adheres to the standards of FIPA and W3C. Similarly the protocol translation between agents and Web services that is: communication protocol translation is considered by only a few. Similarly the factors like ontological translations, negotiation support and decision support is overlooked among almost all of these solutions.

<b>Solutions</b>	<b>FIPA Compliant</b>	<b>W3C Compliant</b>	<b>Protocol Translation</b>	<b>Ontological Translation</b>	<b>Negotiation Support</b>
<b>WSIGS</b>	Yes	Partial support	Partial Support	No	Partial Support
<b>WS2JADE</b>	Yes	No	Yes	No	No
<b>OWL-P</b>	No	Yes	No	No	Yes
<b>MOS</b>	Yes	Yes	No	Yes	No

**Table 3.1** Comparison of state of the art solutions

It is very clear that none of the solutions that have been provided up to now consider all of these aspects as a single solution, therefore this thesis considers a number of these aspects as a single solution. The solution focuses on a standardized solution that considers not only protocol translation but also resolves the semantic aspect by providing ontological translation. Similarly the negotiation process is another open issue, though FIPA has provided the basic guidelines for the negotiation process in the agent technologies but most of the solution providers are unable to incorporate them to engage in Web service negotiation.

### 3.14 Summary

Through this chapter it has been made clear that the component technologies exist but they are not well integrated with one other to address the interoperability issues between agents and Web services. It is obvious from the state of the art solutions that efforts were made to personalize the Web services through the use of agents but none of them provide a complete detailed solution to cope with the interoperability issues between the domains of agents and

Web services. The perspective of various research communities has been explored which are the most relevant to the problems identified in the earlier chapters. The technologies like agents have been highlighted as to benefit the other related technologies. A survey of ongoing research in technologies like Grid computing, Semantic Web and Web services has been presented that depicts that these technologies have limitations, but if integrated with one another can enable these technologies to perform more meaningful tasks and can lead to revolutionary future applications for the Web. However the development of such a large application is a challenging task, therefore an introduction to  $\pi$ -ADL language is presented as it has been used in the subsequent chapters to specify the system and execute those specifications using the  $\pi$ -ADL.NET compiler.

# Chapter 4

## Proposed Solution

---

Chapter 3 provides the overview of the ongoing research and the limitations that have lead to the development of solution for the indentified problems in this thesis. To address these problems there is a strong need to develop system architecture for the interoperability between agents with the Web services without changing existing specifications of both FIPA and W3C. This transformation will facilitate the autonomous agents to dynamically discover, translate, invoke and negotiate with the Web services. In order to have a comprehensive solution it is important to address certain questions like how agents and Web services communicate with one another and how the use of ontologies can resolve the semantic issues in the communication between agents and Web services. How the task delegation will be performed once a certain service is requested? What are the components required and how the agents can negotiate with a Web service once it is invoked? How the information is exchanged between agents and web services? Who will be responsible for task delegation? How the system components will be integrated with one another to have a successful interaction between agents and Web services? The complexity of the proposed architecture is a merger of multiple technologies and, does not permit the complete implementation of the complete architecture; therefore component level details have been specified and validated by executing them in  $\pi$ -ADL.Net compiler to provide the early implementation of the overall system.

### 4.1 Ontology Gateway

Ontology gateway is a middleware-based integration that facilitates the communication between agents and Web services with a special focus on the interoperability of W3C and FIPA ontological representations by defining translations between FIPA Semantic Language and OWL (Pasha et al, 2006) (Sabih et al, 2007). The devised architecture aims to address the issues for providing semantically enabled communication in distributed environments where capabilities of agents and grid systems can be united and reused to accomplish the realization of autonomous semantic grid and utilizing a service based framework. To obtain this integration, we use the properties of Semantic Web that allows the conceptual representation for the distributed knowledge through the use of ontologies. The terms are defined by the use of open ontologies which can be made available by using the standard protocols.

FIPA has defined a criterion to be followed by every language that has to be considered as a FIPA compliant language. The communication framework developed by FIPA allows the agents to communicate with one another using a language that fulfils the minimal criteria to be considered as a FIPA compliant language. The aim is to give a general idea of how the agents can communicate with the OWL-based Web Services by bringing the semantic interoperability. Another key characteristic of the proposed architecture is that it is not restructuring the standards of FIPA and W3C. Our focus is the interoperability of both technologies with a specific focus on the content language translation so that the agents are able to communicate with the Web Services. The Ontology Gateway acts as a translator that will facilitate the protocol transformation along with the language interoperability.

### Ontology Gateway



Ontology Gateway is a middleware based approach for the communication of software agents and Web Services without violating existing standards.

### Web Ontology Language (OWL)



OWL is a W3C specified Web Ontology Language and supports greater interpretability by providing additional vocabulary and formal semantics.


## 4.2 Issues in Semantic Interoperability

In order to specify the semantics of the distributed services explicitly, W3C recommends the OWL (Web Ontology Language). This will enable the applications to utilize the semantics associated with distributed services and to invoke the service that has been specified in OWL. The foundation of the Web Ontology language is based on description logic and, therefore, makes it the most appropriate language for describing terms, concepts, relationships and their properties in the form of hierarchies. OWL is considered to be built on Resource description Framework (RDF), which stores the data in the form of triples and provides supplementary vocabulary and semantics. OWL as content language guarantees that the semantics of the messages exchanged is explicit and unambiguous between the sender

and receiver. OWL has all the features needed to describe rich knowledge structures by agreeing on how meaning is conveyed. It is simpler for applications to share meaningful content. OWL provides the provision of describing classes with their sibling as sub classes and attributes, properties to be defined in a simple way. FIPA SL fulfills the minimal criteria specified by FIPA to be considered as content language that is most suitable for agent communication and is human readable. Moreover various flavors of FIPA SL not only enable it to describe predicates and propositions, but it can also express actions and objects. Table 4.1 represents the comparison of OWL and FIPA SL as content languages.

Table 4.2 and 4.3 present the direct and extended mappings for subsets of FIPA SL2 and OWL Full. These mapping can enable the two languages to specify the logical behaviour and the contextual details in which the data/information is being transferred between the sender and receiver in an independent way. These devised mappings will allow the applications to use any language without adapting to a particular data standard. Various applications for the Grid system will enable these translations to solve the semantic constraints between the ontological description of a term and the user query. Comparison of the two languages in terms of expressive power and interoperability is performed and pros, cons for using FIPA SL or OWL as a content language are also discussed.

**FIPA Semantic Language (FIPA SL)**



FIPA SL is a content language. It provides the constructs for defining Propositions, objects, actions.

### 4.3 Expressiveness of OWL and FIPA SL

FIPA Content Languages, including SL, should be expressive enough to satisfy the following requirements:

- They are capable of representing objects, including identifying expressions to describe objects.
- Proposition should be completely represented.
- They are capable of representing actions.

In FIPA SL, an action expression defines an action that can be performed. An action may be a single action or a composite action built using the sequencing and alternative operators.

An action is used as a content expression when the act is requested and other communicative acts can be derived from it. The propositions may involve explicit or implicit quantification, logical connectives like NOT, AND, OR and modal operators like BELIEF, UNCERTAINTY and INTENTION.

OWL as a content language guarantees that the semantics of the messages exchanged is explicit and unambiguous between the sender and receiver. OWL provides almost all the capabilities that are required to express the knowledge that is to be exchanged, thus making it simpler for applications to exchange meaningful data. But as OWL comes in three sub languages (OWL-Lite, OWL-DL and OWL-Full) the layering structure of these three subsets causes the interoperability issues between them. OWL’s expressive power as a knowledge representation language seems to be adequate for most needs of current agent based systems. OWL is also able to represent proposition, actions and objects. OWL has rich class, property, and axiom to model the world. It can also express the logical connective like AND, OR, NOT and IntersectionOf, UnionOf and ComplementOf.

*\* Ref. Please see APPENDIX C and D for detailed constructs that have been used from FIPA SL and OWL respectively.*

FIPA-SL	OWL
SL is Based on First Order Predicate and Modal Logic	OWL is Based on Description Logic
SL comes in three profiles SL-0, SL-1, SL-2	OWL also has three subsets OWL-Lite, OWL-DL and OWL-Full.
Agent Belief, Desires and Intentions can be represented	Expression of Beliefs, Desires and Intentions is not possible
SL is human readable and has efficient encoding and parsing techniques	OWL is not easily readable
FIPA-SL is widely used and Strongly recommended in agent systems as a content language	OWL is widely recommended for specifying ontologies in web services framework

**Table4.1.** Comparison of OWL and FIPA-SL as content language

#### 4.4 Representing Agent Attitudes

The internal state of an agent is an intentional description making reference to beliefs, desires, intentions and other modalities that agents may have. In the agent domain, it is



necessary to describe the internal state of an agent before sending a message and after receiving it. Therefore, it results in useful assignment of meaning to communication primitives. The main challenge in this context is to provide a clear formalism for expressing the semantics in an unambiguous way for human developers. It should also ensure that the agent's process of interpreting the meaning of the message is compliant to the given semantics. Pre-conditions and post-conditions are expressed in modal logic. The semantics of a communicative act is specified as a set of FIPA SL formula. FIPA SL is a quantified, multi-modal logic language, with modal operators represented as follows:

- ❖ **Belief: (B <agent> <expression>)**  
Agent believes that expression is true
- ❖ **Uncertainty: (U <agent> <expression>)**  
Agent believes that expression is more likely true than false
- ❖ **Intention: (I <agent> <expression>)**  
Agent intends that expression will become true and will plan to bring it about
- ❖ **Persistent Goal: (PG <agent> <expression>)**  
Agent holds a persistent goal that expression becomes true, but will not necessarily plan for it.

But the problem is that we cannot express modalities like Belief, Uncertainty and Intention in OWL as OWL is not based on modal logic. The mental attitudes about the state of the world cannot be expressed when OWL is used as the content language. Only FIPA SL is able to express these modalities. This is where SL distinguishes its expressive power from OWL. The solution proposed in this thesis explicitly considers two types of mapping that are direct and extended mappings. In direct mappings, one language constructs are mapped with the other language in a one to one relationship. In extended mappings the constructs provided by one language have been extended to map the constructs of other language which cannot be directly mapped. Since we are deploying agents and Web services as they have been standardized by their governing bodies, therefore in this solution we do not consider the modalities of agents unless, the entities at Web services have used agents as a solution also.

## 4.5 Comparison based on Logic

SL is based on modal extension of first-order predicate logic. Therefore, the expressive power of SL includes a rich set of operators and constructs, but as the expressive power increases the complexity of reasoning also increases. Description logics represent a subset of first-order predicate logic aimed at being tractable and decidable while maintaining a richness of semantic expressiveness.

	OWL	FIPA
<b>Basic Class</b>	owl:Thing	:THING
	owl:Class	Frame :Class
<b>Class Axioms</b>	rdfs:subClassOf	subclass-of
	owl:equivalentClass	Equivalent :SAME-VALUES
<b>Class Description</b>	Individual	Individual
	rdfs:domain	:DOMAINS
	rdfs:range	:SLOT-VALUETYPE
	owl:restriction	Content Description
	owl:minCardinality	:MINIMUMCARDINALITY
	owl:maxCardinality	:MAXIMUMCARDINALITY
	owl:DatatypeProperty	:VALUE-TYPE
	owl:intersectionOf	BinaryLogicalOp = "or"
	owl:unionOf	BinaryLogicalOp = "and"
	owl:complementOf	UnaryLogicalOp = "not"
	owl:oneOf	set-of
	owl:disjointWith	:NOT-SAMEVALUES

**Table 4.2.** OWL Class Mappings

It has declarative formalism for the representation and expression of knowledge and sound, tractable reasoning methods. OWL is based on description logic and is distinguished by its decidable characteristic and formally defined semantics, which enables the subsumption relationship to be computed by suitable algorithms.

Compared with many of its predecessors, description logics provide a formal, logic-based semantics to make it a useful knowledge representation framework for different application domains. A comparative analysis of both the languages was performed in depth and certain

mappings were devised in order for both the entities to communicate in an efficient way. Table 4.2 and Table 4.3 represent these mappings.

	<b>OWL</b>	<b>FIPA</b>
<b>Property</b>	rdfs:domain	:DOMAINS
<b>Associations</b>	rdfs:range	:SLOT-VALUE-TYPE
<b>Property Relationships</b>	owl:equivalentProperty	:SLOT-SAMEVALUES
	owl:inverseOf	:SLOT-INVERSE
	rdfs:subPropertyOf	:SLOT-SUBSETOF-VALUES
	rdfs:Datatype	:SLOT-VALUE-TYPE
<b>Property Restrictions</b>	owl:cardinality	:SLOTCARDINALITY
	owl:maxCardinality	:SLOTMAXIMUMCARDINALITY
	owl:minCardinality	SLOT-MINIMUMCARDINALITY
	owl:allValuesFrom	:SLOT-SAME-VALUES
	owl:someValuesFrom	:SLOT-SOMEVALUES
	owl:hasValue	:SLOT-SUBSETOF-VALUES

**Table 4.3.** OWL property mappings

## 4.6 Proposed System Architecture of Ontology Gateway

The idea behind the proposed architecture of Ontology Gateway is how the software agents will communicate with OWL based Web Services and vice versa. The objective of Ontology Gateway is to translate the OWL to FIPA ontologies with minimum semantic loss because while translation from OWL to FIPA SL we can directly map the constructs provided, whereas when translating from FIPA SL to OWL we have to make a trade off on concepts like belief, desire, intention, as they cannot be directly mapped. The proposed architecture

does not require any change in the standard specifications and implementation of the existing technologies. Open systems like Grid are still working on hiding resource heterogeneity and making a scalable and robust infrastructure, while software agents act as key entities of these systems to semantically interoperate and negotiate with each other. Figure 4.1. depicts the abstract architecture of the proposed system.

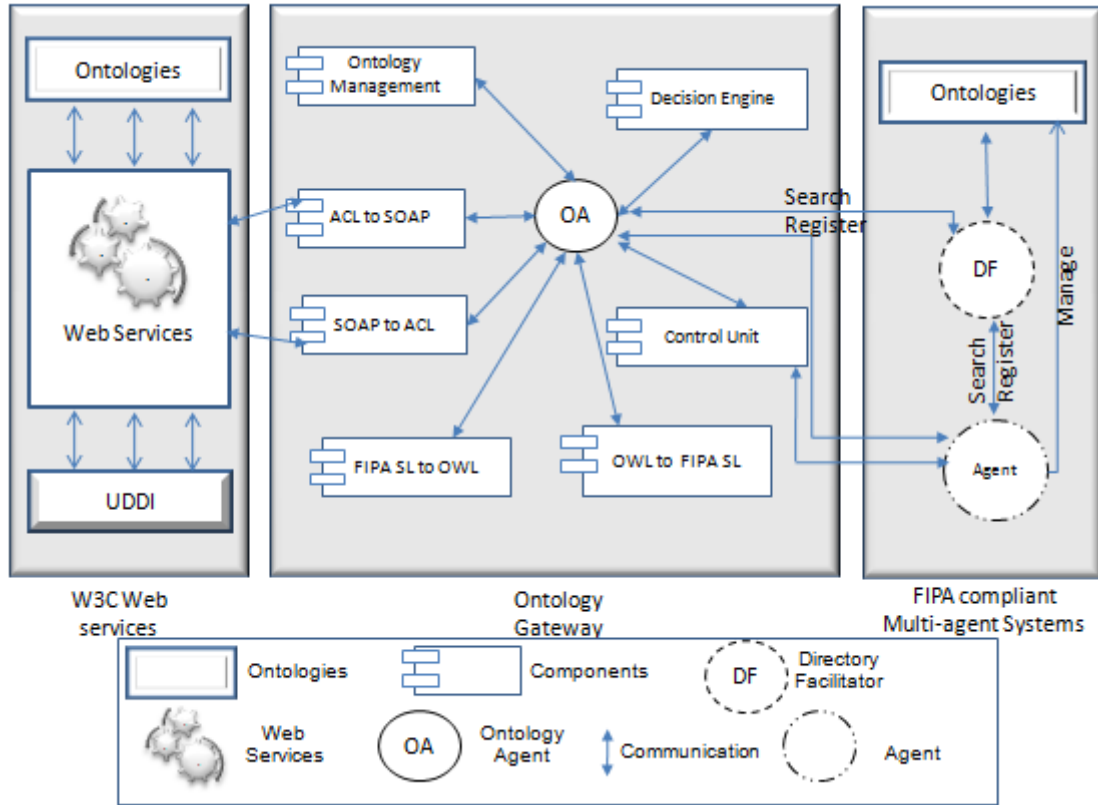


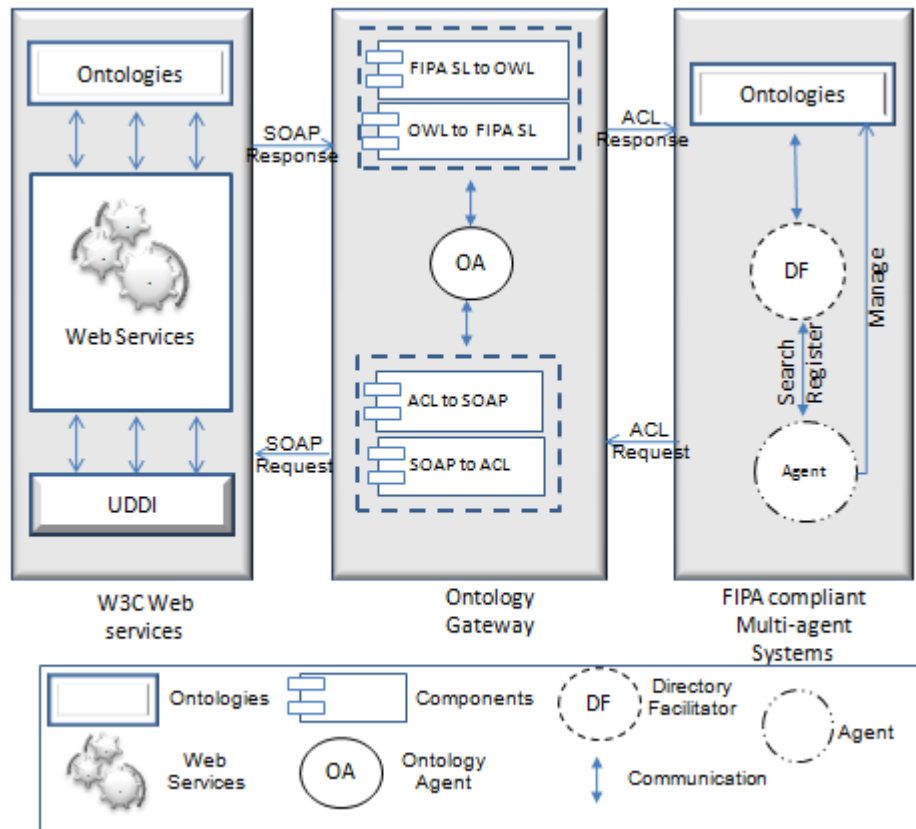
Figure 4.1: **Initial architecture of Ontology Gateway**

The main issues in designing the proposed system are how agents will communicate with OWL based Web Services, how an agent can understand an OWL based Web Service, and how OWL Web Services can communicate with agents to obtain services required to accomplish a task. Since both OWL and FIPA SL have different principles in terms of syntax, semantics and implementation, devising transformations for such a system is quite demanding and challenging. It is very important to highlight the aspect that the ontologies that need to be transformed need to share the same vocabulary set and a uniform structure. In the future, however, dynamic support for ontologies can be provided. Second key concern is to design a system that need not change the underlying specification and implementations of both technologies, which is not a trivial task.

The proposed architecture reuses components in our earlier work i.e. AgentWeb Gateway (Shafiq et al., 2005). Major reused components are ACL to SOAP and SOAP to ACL converters. The proposed architecture shown in Figure 4.1 outlines main subsystems and their interaction. The brief description of the subsystems and the interaction is highlighted in the next sections.

#### 4.6.1 Agents Communicating with OWL Based Services

Consider a case of an agent that needs to establish communication with OWL based Web Service for service utilization. Figure 4.2 shows the scenario where an agent initiates a search request for a particular service. This search requires a number of steps starting from the transformation of the ACL search query into SOAP search query using ACL to SOAP converter and forwarding it to UDDI where the required service is registered. Once the software agent obtains a reference to the required service, it retrieves the service, and extracts associated parameters necessary for service consumption.



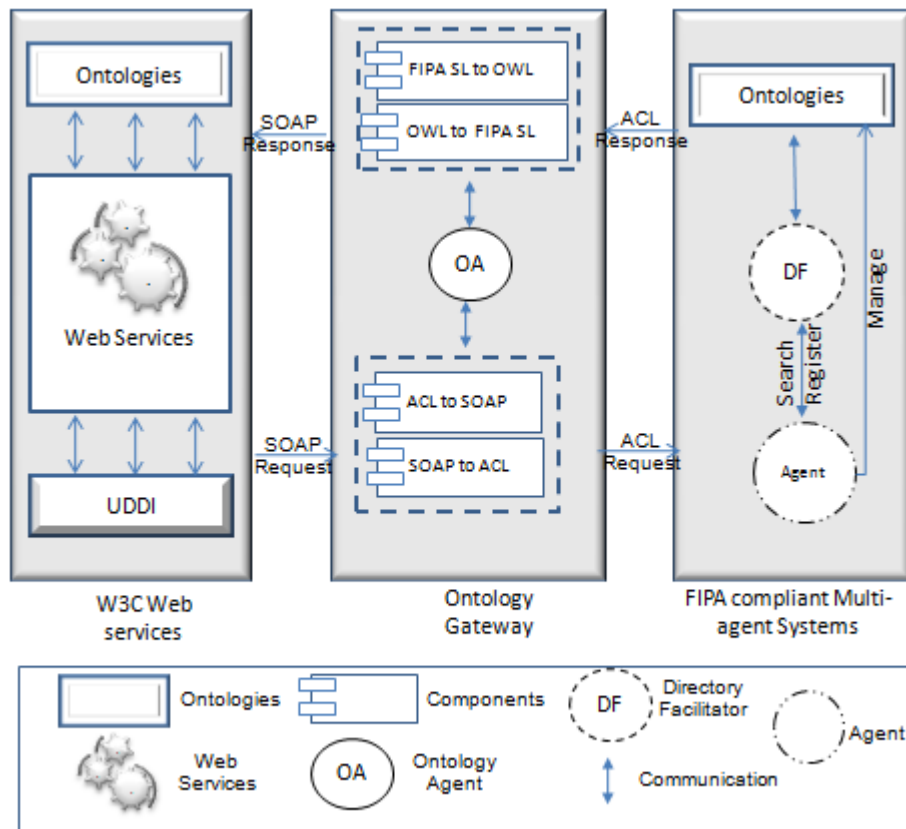
**Figure 4.2:** Agents Communicating with OWL based services

It is important for the agent to be aware of the ontology being used, Agent-Action Schema, Predicate Schema and Concept Schema. For this reason the web service is processed through the Ontology Gateway that generates the FIPA compliant equivalent

message of the content that was previously described in Web Ontology Language. The Agent-Action schema contains the list of actions that the Agent can perform, whereas the Predicate schema in the Agent’s ontology contains a list of outcomes/responses of the Agent, and finally the Concept schema defines the entities and their properties.

#### 4.6.2 OWL Based Service Communicating with Agent

Figure 4.3 depicts a scenario where a web client requests information about a service that was published in a Multi-agent Systems environment. As the query was transferred through SOAP protocol and was carrying XML based data therefore was routed to SOAP to ACL module.



**Figure 4.3:** OWL based services interacting with Agents

This module retrieves the service name and additional parameters, whereas initiating a valid ACL-based Directory Facilitator search query to the agent platform. Directory Facilitator of the agent platform then carries out a search and if requested service is retrieved, then the service name is passed to the agent who initiated the request through Ontology Gateway.

In this process, Ontology Gateway converts FIPA SL content into an equivalent OWL content, and service parameters are embedded into the SOAP message, which is forwarded to the web client that initiated the search requested for the service. This enables the web clients to utilize the Ontology Gateway for retrieving information from a Multi-gent System.

The Ontology Gateway enables bidirectional semantic communication between OWL based services and FIPA compliant agent system. The process of translation from OWL to FIPA ontology is complete but numerous tasks are required. For instance, in the Ontology Gateway, parsing of each and every concept in a given ontology is performed, which requires recompilation of the ontology upon the addition of new concept.

#### $\Pi$ -ADL



$\pi$ -ADL (Architecture Description Language) is a formal language and its primary focus is on software architecture description in a mathematical way and the ability to reason about it. Formal foundations of  $\pi$ -ADL are based on higher-order typed  $\pi$ -calculus.

### 4.7 Details of Architecture Components

This section provides a detailed view of each component of Ontology Gateway along with its  $\pi$ -ADL specifications each module in detail. The objective of using  $\pi$ -ADL (Architecture Description Language) is its primary focus on the software architecture description at the mathematical sense and the ability to reason about it.  $\pi$ -ADL encompasses the architecture centric constructs for representing the structure, as well as behavioural details of any system. The constructs provided by the  $\pi$ -ADL provide the complete freedom of defining the dynamic behaviour, interactions among various entities, and constraints for any architectural model. Also tool support is provided in the form of  $\pi$ -ADL.NET compiler to formally validate these specifications.

The Ontology Gateway proposed in this thesis addresses the issues of communication between agents and Web service. However, it is difficult to develop a complete executable system due to the broad scope of the project and the technologies involved. Therefore, the formalization of the architectural components is provided through the use of  $\pi$ -ADL that provides all the necessary constructs for describing dynamic architectures. The benefit of these  $\pi$ -ADL specifications is the early executable model of the system that is technology

neutral. Another important factor for the formal specifications using  $\pi$ -ADL is to provide clear and unambiguous specifications of the system.

Different components have been introduced as a part of Ontology Gateway that will be required to overcome the communication gap between agents and Web services communication. Based on the functionality of these components, their interaction with other components, inputs, outputs have been clearly specified and executed using  $\pi$ -ADL.NET. Figure 4.4 depicts the  $\pi$ -ADL model of the overall system where the communication request is made by the actor represented as a behaviour and the other components participate by playing their roles in the communication process described as an abstraction and communicate through shared connections for communication.

**Figure 4.4:** The  $\pi$ -ADL model of the overall architecture



### 4.7.1 Control Unit

First of all, the requesting agent initiates the communication process with the services exposed by Ontology Gateway and sends its reference and query. This query is received by the Control Unit. The query received contains a number of parameters i.e. if it is a request or a response, who is the sender, reference of the receiver, content, language used and reference ontology, from which the Control Unit identifies the necessary actions to be taken. Figure 4.5 presents the  $\pi$ -ADL specifications of the control unit where first the agent is validated to check if the agent is registered. If the agent, is registered only then the communication initiates, otherwise the request is discarded. Agent authenticity is described in terms of true and false hence the connection receives either of these values if the agent is recognized. Once the authenticity of the agent is checked then the requested service parameters are sent to the ontology agent for further processing. During the communication or negotiation process it sometimes becomes necessary to take additional information from the user agent therefore the Control Unit is also responsible for sending out the additional parameters that are required in order to make a service binding.

The bold words in the code show the keywords provided by  $\pi$ -ADL. Lines 1 to 4 in figure 4.5 presents the *CU* as an abstraction with *cuQuery* as parameter which is composed of certain parameters used in communication. Line 3 presents the *content* field which is a *view* of multiple values. Line 18 declares a connection *vConn* between the Control Unit and the Ontology agent to check the agents authenticity, *isAgentValidated* is the field of the Boolean type which means it is true if the agent is authentic and false if it is not. Line 23 shows that response is received from the ontology agent specifying if the agent is registered or not. If the agent is registered then the query is sent to the negotiation module for further processing as shown in line 35, if it is not a registered agent then a response is sent with a message *unrecognized request* as highlighted in line 28.

```
1.  value CU is abstraction (cuQuery : view[request : String, sender : String, receiver : String,
2.      content : view[service_id : String, service_name : String,
3.      params : sequence[view[name : String, val : String]], numOfParams : Integer],
4.      language : String, ontology : String, reply_with : String)
5.  {
6.      aConn : Connection[view[request : String, sender : String, receiver : String,
7.      content : view[service_id : String, service_name : String,
8.      params : sequence[view[name : String, val : String]], numOfParams : Integer],
9.      language : String, ontology : String, reply_with : String]];
10.     qConn : Connection[view[request : String, sender : String, receiver : String,
11.     content : view[service_id : String, service_name : String,
12.     params : sequence[view[name : String, val : String]], numOfParams : Integer],
13.     language : String, ontology : String, reply_with : String]];
14.
15.     via qConn receive query;
16.     unobservable;
17.     isAgentValidated : Boolean;
18.     vConn : Connection[Boolean];
19.
```

```

20. //this is where we invoke ontology agent
21. via out send "*** CU ***\n\n--Validating Agent Authenticity--\n";
22. via OA send cuQuery where {vConn renames vConn};
23. via vConn receive isAgentValidated;
24. via out send "*** CU ***\n\n";
25. if (!isAgentValidated) do
26. {
27.     via out send "--Unrecognized Agent--\n";
28.     cuQuery::reply_with = "Unrecognized request";
29.     via qConn send cuQuery where {qConn renames qConn};
30.     done;
31. }
32. else do
33. {
34.     via out send "--Forwarding service profile parameters to NM--\n";
35.     via NM send cuQuery where {aConn renames cConn};
36. }
37. }

```

**Figure 4.5:** Control Unit specifications in  $\pi$ -ADL

#### 4.7.2 Ontology Agent

As soon as the control unit delegates the information to the ontology agent it is then the responsibility of this agent to propagate the different messages that are being received or sent from either the Web services or the web client to route them to the appropriate module for further processing. The Ontology Agent will be receiving a query from agents in the form of ACL messages. It will then reply to the agents by executing those queries on the registered ontologies and sending an ACL message as a reply. Whereas the web clients will be sending the queries using SOAP as a protocol therefore it will route that message to the SOAP to ACL converter which will extract the contents in message and will then send an ACL message to the Ontology agent with the parameters that were received by the web client.

In Figure 4.6 Ontology agent specifications are written in  $\pi$ -ADL, where *OA* is viewed as an abstraction and declared in lines 1 to 4 that receives an input and delegates the task to appropriate module based on the type of request/response that is to be performed. Line 6 defines the connection *vConn* established between the Control Unit and the Ontology Agent with the *result* as a Boolean value declared in line 7 in order to determine that the agent is registered or not as this value has to be utilized by the CU. Once the agent is validated to be registered with the agent platform then the message is forwarded to the transformation module for the conversion which through a connection between the transformation module that is *oaQueryConn* and is declared from lines 8 to 11. The transformation processes the message based on its type which can be a SOAP message, an ACL message or it could be content in the form of OWL or FIPA SL. It is clear from the  $\pi$ -ADL specification of the ontology agent that the structure of the query that is forwarded from the Control Unit abstraction is compatible with the Ontology agent and is composed of the following fields.

A typical message contains the following fields:

- request:** a performative which has data type string and used for requesting or sending information.
- sender:** address of the entity sending the query.
- receiver:** receiver parameter defines the address of the entity going to receive the message.
- content:** contains the services with their name and number of parameters associated with that service.
- language:** the type of language used which could be SL or OWL.
- ontology:** this field is the name of the ontology used
- reply-with:** this field highlights the expression used by the entities to identify the message.

```
1. value OA is abstraction (oaQuery : view[request : String, sender : String, receiver : String,
2.     content : view[service_id : String, service_name : String,
3.     params : sequence[view[name : String, val : String]], numOfParams : Integer],
4.     language : String, ontology : String, reply_with : String])
5. {
6.     vConn : Connection[Boolean];
7.     result : Boolean;
8.     oaQueryConn : Connection[view[request : String, sender : String, receiver : String,
9.     content : view[service_id : String, service_name : String,
10.    params : sequence[view[name : String, val : String]], numOfParams : Integer],
11.    language : String, ontology : String, reply_with : String]];
12.
13.     result = true; //default initialization
14.     via out send "**** OA ****\n\n";
15.     //lookup service availability from different transformation components, and update the value of result
16.     via OwlToSl send oaQuery where {oaQueryConn renames convQueryConn};
17.
18.     unobservable;
19.
20.     via SIToOwl send oaQuery where {oaQueryConn renames convQueryConn};
21.
22.     unobservable;
23.
24.     via AclToSoap send oaQuery where {oaQueryConn renames convQueryConn};
25.
26.     unobservable;
27.
28.     via SoapToAcl send oaQuery where {oaQueryConn renames convQueryConn};
29.
30.     unobservable;
31.
32.
33.     via out send "--Communicating service availability to CU--\n";
34.     compose
35.     {
36.         via vConn send result;
37.     and
38.         done;
39.     }
```

```
40. }
41. }
```

**Figure 4.6:** Ontology Agent specifications in  $\pi$ -ADL

#### 4.7.3 OWL to FIPA SL Module

When the Ontology agent receives the information in the form of OWL from the web client it requires translating it into equivalent FIPA content language that is FIPA SL. The transformation takes place based on the mappings provided in Table 4.2 and Table 4.3.

#### 4.7.4 FIPA SL to OWL Module

When the information is propagated from the agent platforms to the web clients the content languages that is FIPA SL has to be translated into equivalent web understandable content that is OWL. In this case, therefore, this module will perform the translation based on Table 4.2 and Table 4.3.

#### 4.7.5 SOAP to ACL Module

For an agent to understand the content that is sent by a web client it is necessary that the protocol that is carrying information from web client should be in compliance with the agent's protocols. Therefore, SOAP to ACL will generate an equivalent ACL message in correspondence to every SOAP message that was sent.

#### 4.7.6 ACL to SOAP Module

ACL to SOAP module is utilized when an agent is sending the information through the agent platform to the web client and as the web recommends only the use of SOAP therefore the ACL component will transform the sent information into the form that is compatible with the SOAP.

```
1. value OwlToSl is abstraction (convQuery : view[request : String, sender : String, receiver : String,
2.     content : view[service_id : String, service_name : String,
3.     params : sequence[view[name : String, val : String]], numOfParams : Integer],
4.     language : String, ontology : String, reply_with : String])
5. {
6.     convQueryConn : Connection[view[request : String, sender : String, receiver : String,
7.     content : view[service_id : String, service_name : String,
8.     params : sequence[view[name : String, val : String]], numOfParams : Integer],
9.     language : String, ontology : String, reply_with : String]];
10.     unobservable;
11. }
```

```

12.
13. value SToOwl is abstraction (convQuery : view[request : String, sender : String, receiver : String,
14.     content : view[service_id : String, service_name : String,
15.     params : sequence[view[name : String, val : String]], numOfParams : Integer],
16.     language : String, ontology : String, reply_with : String)
17.
18. {
19.     convQueryConn : Connection[view[request : String, sender : String, receiver : String,
20.     content : view[service_id : String, service_name : String,
21.     params : sequence[view[name : String, val : String]], numOfParams : Integer],
22.     language : String, ontology : String, reply_with : String]];
23.     unobservable;
24. }
25.
26. value AclToSoap is abstraction (convQuery : view[request : String, sender : String, receiver : String,
27.     content : view[service_id : String, service_name : String,
28.     params : sequence[view[name : String, val : String]], numOfParams : Integer],
29.     language : String, ontology : String, reply_with : String)
30. {
31.     convQueryConn : Connection[view[request : String, sender : String, receiver : String,
32.     content : view[service_id : String, service_name : String,
33.     params : sequence[view[name : String, val : String]], numOfParams : Integer],
34.     language : String, ontology : String, reply_with : String]];
35.     unobservable;
36. }
37.
38. value SoapToAcl is abstraction (convQuery : view[request : String, from : String, to : String,
39.     body : view[class : String, fields : sequence[view[name : String, val : String]],
40.     properties : sequence[any], subclasses : sequence[any], restrictions : sequence[any]],
41.     fault : String, reply_with : String)
42. {
43.     convQueryConn : Connection[view[request : String, sender : String, receiver : String,
44.     content : view[service_id : String, service_name : String,
45.     params : sequence[view[name : String, val : String]], numOfParams : Integer],
46.     language : String, ontology : String, reply_with : String]];
47.     soapView : view [encoding_style : String, from_http : String, to_http : String, body : String,
48.     fault : String];
49.     aclView : view [encoding : String, sender : String, receiver : String, content : String,
50.     performative : String,
51.     reply_to : String, ontology : String, protocol : String,
52.     conversation_identifier : String,
53.     reply_with : String, in_reply_to : String, reply_by : String];
54.     unobservable;
55. }

```

**Figure 4.7:** Transformation Component

#### 4.7.7 Transformation Component:

Since this ontology is written in OWL for a Semantic Web service, although it is considered as a valid content language by FIPA, but it is not as expressive as SL, so there is a need to translate this ontology from OWL to SL. The Control Unit feeds this ontology to the OWL to FIPA ontology translator through Ontology agent, which returns the FIPA Ontology equivalent of the OWL ontology fed as an input. This is done with the help of a matchmaking service that returns a reference or handle of that service to the Ontology agent and then to

Control Unit. This handle enables the Control Unit to fetch the service profile of the service and its ontology, without which semantic understanding is unattainable.

Figure 4.7 presents the  $\pi$ -ADL specifications for the transformation modules i.e. OWL to SL, SL to OWL, ACL to SOAP, SOAP to ACL based on the type. If it is content transformation it will be forwarded to SL to OWL/OWL to SL modules or if it is the protocol transformation it will be forwarded to ACL to SOAP/SOAP to ACL respectively. These four modules accommodate one request at a time as requested by the Ontology Agent. The transformation module takes an input query from the Ontology agent and then converts the query into the format required by the ontology agent. Lines 1 to 4 present the *OWLtoSL* abstraction. The message is sent by the ontology agent to the transformation module by the connection *convQueryConn* so the message format is consistent to the one used by the agents. However, within the content field the *view* keyword is used which depicts a service name identified by the identifier and the conversion is done based on mappings provided in the earlier section of this chapter and requires parsing of the query, therefore these low level details were simply replaced by the *unobservable* keyword.

An important aspect to be noted in the  $\pi$ -ADL specifications of Figure 4.7 is the abstraction *SOAPTocl* presented in lines 38 to 41. The parameters of the *convQuery* are different from that of the previous abstractions of the transformation modules. The *from* field is presented by the type string and contains the address of the web client and *to* field represents the entity to which the web client will be communicating in the agent platform. The body of the message, however, contains *classes*, *fields*, *properties*, *subclasses*, *restrictions* etc. The *body* field in line 39 contains the various classes with the fields associated. The *properties*, *subclasses* and *restrictions* are defined as *sequence[any]*. Line 47 and 49 depict the *soapView* and *aclView* containing how the parameters are represented from SOAP to ACL when the translation takes place.

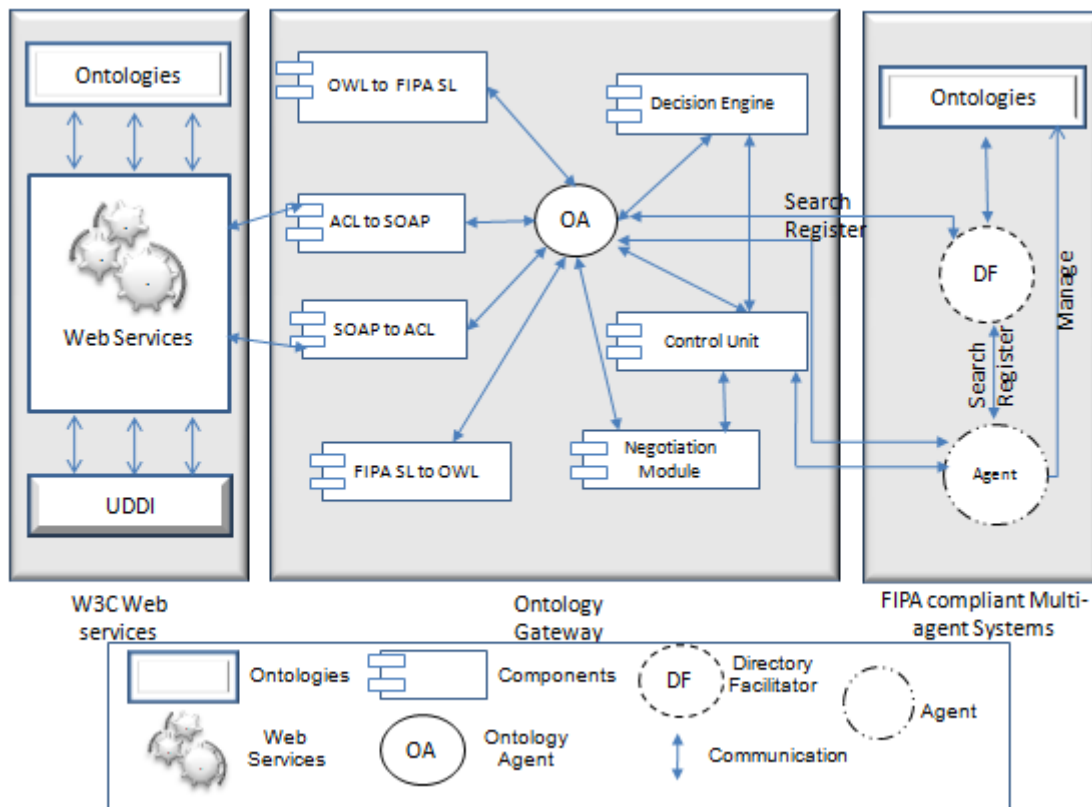
### Directory Facilitator (DF)



Directory Facilitator is a registry that exists in the agent platforms where the agents can register themselves along with their service descriptions and the other agents can discover them.

#### 4.7.8 Negotiation Module:

Negotiation can be referred as an iterative process for communication and taking decisions between the participating parties (Bichler, 2003) who: (i) are unable to achieve their goals through simple communication (ii) share information between one another based on some arguments and relevant responses (iii) cooperate with one another on interdependent tasks; and (iv) finally look for a decision which leads to consensus of the participating bodies. The result of a negotiation process is not guaranteed to be an agreement because sometimes the participating bodies might not come to an agreement thus resulting in a disagreement. A rule set or a decision engine is required in order to opt from a set of alternatives that what actions should be performed.



**Figure 4.8:** Negotiation among agents and Web Services

In order to conduct meaningful negotiation between agent and Semantic Web Service, a negotiation module also known as a Mediator, needs the reference of the requesting agent, its FIPA ontology and service profile of the Semantic web service. Handles to all of these resources are passed to this module at the time of transfer of control to it by Control Unit. This module shall create an agent at runtime referred to as Runtime Agent (RA), in which shall be used to query the web service ontology. The Runtime agent is responsible for

requesting parameters from the service provider and the requesting agent once the binding has been done between agent and service. Moreover, the Runtime agent extracts all negotiable parameters from the profile of the Semantic web service with the help of understanding its ontology.

```

1.  value NM is abstraction(nmQuery : view[request : String, sender : String, receiver : String,
2.      content : view[service_id : String, service_name : String,
3.      params : sequence[view[name : String, val : String]], numOfParams : Integer],
4.      language : String, ontology : String, reply_with : String)
5.  {
6.      cConn : Connection[view[request : String, sender : String, receiver : String,
7.      content : view[service_id : String, service_name : String,
8.      params : sequence[view[name : String, val : String]], numOfParams : Integer],
9.      language : String, ontology : String, reply_with : String]];
10.     nConn : Connection[view[request : String, sender : String, receiver : String,
11.     content : view[service_id : String, service_name : String,
12.     params : sequence[view[name : String, val : String]], numOfParams : Integer],
13.     language : String, ontology : String, reply_with : String]];
14.
15.     via CU send nmQuery where {cConn renames aConn};
16.     via cConn receive cuQuery;
17.     via out send "**** NM ****\n\n";
18.     if (nmQuery::content::service_name == "service") do
19.     {
20.         via out send "--Runtime Reservation initiated for processing user request--\n";
21.         via rt_Reserve send nmQuery where {nConn renames rtConn};
22.     }
23. }

```

**Figure 4.9:** Negotiation Module

Figure 4.9 presents the  $\pi$ -ADL specification of the negotiation module where two connections have been established, that is the *cConn* and *nConn* with the *CU* and *rt\_Reserve* abstraction respectively from lines 6-14. As discussed in the Control Unit, the additional parameters can be requested in order to have a service binding between the service provider and the agent, hence the *nmQuery* is forwarded to the *CU* abstraction for the parameters request. If the value for the *service\_name* is equivalent to the service then the *nmquery* is forwarded to the *rt\_Reserve* abstraction, as presented in the line 21, for further processing.

```

1.  value rt_Reserve is abstraction(nmQuery : view[request : String, sender : String, receiver : String,
2.      content : view[service_id : String, service_name : String,
3.      params : sequence[view[name : String, val : String]], numOfParams : Integer],
4.      language : String, ontology : String, reply_with : String)
5.  {
6.      rtConn : Connection[view[request : String, sender : String, receiver : String,
7.      content : view[service_id : String, service_name : String,
8.      params : sequence[view[name : String, val : String]], numOfParams : Integer],
9.      language : String, ontology : String, reply_with : String]];
10.
11.     uddiResp : sequence[String];
12.     uddiRespConn : Connection[sequence[String]];
13.
14.     serviceInfo : view[service_name : String,
15.     params : sequence[view[name : String, val : String]], numOfParams : Integer, message : String];
16.     serviceInfoConn : Connection[view[service_name : String,

```



```

17.      params : sequence[view[name : String, val : String]], numOfParams : Integer, message : String];
18.
19.      via out send "***rtService***\n\n";
20.
21.      serviceInfo::service_name = "Service Name";
22.      serviceInfo::params = nmQuery::content::params;
23.      serviceInfo::numOfParams = nmQuery::content::numOfParams;
24.
25.      via out send "--Sending request to UDDI--\n\n";
26.      via UDDI send serviceInfo where {uddiRespConn renames respConn};
27.      via uddiRespConn receive uddiResp;
28.
29.      via out send "***rtService***\n\n--Service proposals from UDDI--\n\n";
30.      via out send uddiResp;
31.      via out send "\n\n";
32.
33.      via DE send serviceInfo where {serviceInfoConn renames reqServiceConn};
34.      via serviceInfoConn receive serviceInfo;
35.      via out send "***rtService***\n\n--Optimal service returned by DE--\n\n";
36.      via out send serviceInfo;
37.      via out send "\n\n";
38.
39.      via dynamic(serviceInfo::service_name) send serviceInfo
40.          where {serviceInfoConn renames reqServiceConn};
41.      via serviceInfoConn receive serviceInfo; //serviceInfo now contains additional parameters request
42.
43.      nmQuery::request = serviceInfo::service_name;
44.      nmQuery::sender = "";
45.      nmQuery::receiver = "";
46.      nmQuery::content::service_id = "";
47.      nmQuery::content::service_name = "";
48.      nmQuery::content::params = serviceInfo::params;
49.      nmQuery::content::numOfParams = serviceInfo::numOfParams;
50.      nmQuery::language = "";
51.      nmQuery::ontology = "";
52.      nmQuery::reply_with = "";
53.
54.      via rtConn send nmQuery; //sent to Actor
55.      via rtConn receive nmQuery; //received from Actor
56.
57.      serviceInfo::service_name = "service";
58.      serviceInfo::params = nmQuery::content::params;
59.      via serviceInfoConn send serviceInfo; //sent to service provider
60.      via serviceInfoConn receive serviceInfo; //received from service provider
61.
62.      nmQuery::reply_with = serviceInfo::message;
63.      via out send "\n\n***rtService***\n\n--Service invocation successful--\n\n";
64.      via rtConn send nmQuery;
65.  }

```

**Figure 4.10:** Runtime Agent

Figure 4.10 presents the  $\pi$ -ADL specifications of the Runtime agent declared as *rt\_Reserve* abstraction which takes *nmQuery* as an argument received from the *NM* abstraction as shown in lines 1 to 4. The *rtConn* connection is established with the *NM* abstraction for communication as presented in line 6. The *uddiResp* is the value retrieved from the UDDI by establishing a connection *uddiRespConn* with it as shown in line 12 of figure 4.10. Lines 25-27 depict the request sent to the UDDI and the *uddiResponse* received

as a result. There can be a number of services available in the UDDI that will be returned to the Runtime agent which will then forward it to the *DE* abstraction as shown in lines 33 to 37 which retrieve the most relevant service based on a threshold value. Lines 43 to 52 present the parameters that are required to identify the communication agent with the Web service. The service retrieved from the *DE* abstraction might request the additional parameters, therefore the request is sent back to the negotiation module which forwards that request back to the *CU* abstraction as shown in lines 54 and 55 that the *nmQuery* is sent through *rtConn* and the response is received. A message is sent back through *rtConn* to *NM* abstraction on the successful invocation of the service shown in line 64.

#### 4.7.9 Decision Engine

Once a number of services are available, then the decision engine selects the most appropriate service among the retrieved service based on a particular threshold value, which specifies the extent of similarity in the parameters of the service that was requested and the one that was retrieved. Based on this selection a particular service is retrieved which is then invoked by the run time agent.

The  $\pi$ -ADL specifications shown in figure 4.11 present the *DE* abstraction. The *reqServiceConn* is a connection declared in line 4 with a number of parameters including the *service\_name*, *params*, *numOfParams* and a *message*. The service parameters are matched to the parameters of the query generated by the agent as shown in lines 20 to 38. Then the most appropriate service is selected based on a *threshold* value shown in line 40 to select the most appropriate service based on the extent to which a service matches the requested service.

```

1.  value DE is abstraction (reqService : view[service_name : String,
2.      params : sequence[view[name : String, val : String]], numOfParams : Integer, message : String])
3.  {
4.      reqServiceConn : Connection[view[service_name : String,
5.          params : sequence[view[name : String, val : String]],
6.          numOfParams : Integer, message : String]];
7.
8.      services : sequence[view[service_name : String,
9.          params : sequence[view[name : String, val : String]],
10.         numOfParams : Integer, message : String]];
11.         numOfServices : Integer;
12.         i : Integer;
13.         j : Integer;
14.         paramMatchCount : Integer;
15.         fractionMatch : Float;
16.         numOfServices = n;
17.         i = 0;
18.
19.         via out send "***DE***\n\n";
20.         while (i < numOfServices) do
21.             {
22.                 j = 0;
23.                 paramMatchCount = 0;

```

```

24.         while (j < reqService::numOfParams) do
25.             {
26.         if (reqService::params(j)::val == services(i)::params(j)::val) do
27.                 paramMatchCount = paramMatchCount + 1;
28.                 j = j + 1;
29.             }
30.             fractionMatch = paramMatchCount / reqService::numOfParams;
31.
32.             if (fractionMatch >= threshold) do
33.                 {
34.                     reqService = services(i);
35.                     i = numOfServices;
36.                 }
37.                 i = i + 1;
38.             }
39.
40.         if (fractionMatch >= threshold) do
41.             {
42.                 compose
43.                 {
44.                     via out send "--";
45.                     via out send reqService::service_name;
46.                     via out send " Service Selected--\n";
47.                     via reqServiceConn send reqService;
48.                 and
49.                 done;
50.             }
51.         }
52.         else do
53.             {
54.                 done;
55.             }
56.     }

```

**Figure 4.11: Decision Engine**

## 4.8 Summary

This chapter gave an overview of the proposed solution and addresses the questions that were presented in chapter 2 of this thesis. The chapter presented the architectural view of the system from structural and design view. The first section gave the overview of the Ontology Gateway as a solution to overcome the semantic interoperability issues and to fulfill the requirements of agent and Web services communication. Based on the facts that agents utilize FIPA SL and Web services utilize the OWL as a content language, based on comparative analysis mappings were devised to preserve the semantics of both technologies when they communicate with one another. The last section of this chapter comprised of details about the architectural components of the Ontology Gateway. As was discussed in the previous chapter, the implementation of this whole system is a challenging and time consuming task therefore we use  $\pi$ -ADL for specifying the architectural components that will give the reader clearer and less unambiguous specifications, but will enable a better understanding of the architectural components. The next chapter will highlight the proposed architecture from a deployment view.

# Chapter 5

## Implementation and Validation

---

The aim of the previous chapter was to define the proposed solution from the structural and design perspective. This chapter provides insight about the details from the behavioural and deployment perspective of the proposed architecture. The functionality of each component is highlighted through use cases and their interactions in the form of sequence diagrams. Each component's functionality is explicitly stated with the type of input messages it receives and how it processes the input information and forwards it the collaborating module. Different scenarios have been discussed where agents and Web services request information for a particular service and then engage in a negotiation process. The applicability of this work can be beneficial for all the stake holders that want to integrate agents and the Web services and the work proposed is beneficial to a number of applications. Two case studies have been introduced as a proof of concept.

The various components of the Ontology Gateway comprise of a Control Unit which receives the input query from the initiating agent and then forward it to another component referred to as the Ontology agent. The Ontology agent on receiving the message takes further required steps to process the query if it is generated by the agent. The query can be submitted to either of these modules that is SL to OWL or ACL to SOAP. The SL to OWL converter converts the content of the query from FIPA Semantic Language to Web Ontology Language because this information has to be sent over the web. But once the content is transformed then it is also important to convert the protocols that are from ACL to SOAP. Similarly if the request is generated by the Web client or the Grid client then OWL to SL and SOAP to ACL are used for the conversion.

These steps are basically required for the communication process. However, when the agent has once retrieved a service through this communication then it will engage itself using a Negotiation Module that will then be further responsible for handling the negotiation process through the Runtime agent. In case additional parameters are requested the Control Unit will provide the additional parameters to the run time agent for service binding between the provider and the consumer. This chapter discusses the details of these components from the implementation specific perspective based on different scenarios.

## 5.1 Comparative Analysis of Service Discovery and Communication Protocols

!

This section provides details on service discovery and communication protocol translation. To understand these translations it is important to understand the resemblance in the structure of both technologies. The comparisons specifications of both Web Services Framework and Multi-agent Systems are presented in Table 5.1.

Directory Facilitator Service Description	UDDI Service Description
Agent Identifier	Business Entity
Contacting Body(Concept Schema)	Contact
Contact Name/Identifier(Concept Schema)	Contact Name
Contact_Phone(Concept Schema)	Phone
Contact_Email(Concept Schema)	Email
Contact_Address(Concept Schema)	Address
Service-Description_name	BusinessService_Name
Service-Description_type	Service Category
Service-Description property	Service Parameter
Property(Action Schema)	Input
Property(Predicate Schema)	Output

**Table 5.1.** Comparison between Directory Facilitator and UDDI Service Descriptions!

### 5.1.1 Analysis of service registry and discovery

!!!!According to the W3C clients have to perform the service discovery in a global registry called UDDI, whereas Multi-agents Systems provide the same service for discovery through its directory facilitator or the yellow pages for the service. Both the technologies have a different mechanism for storing information and how the discovery is performed. UDDI maintains some business related information along with the service description language that is WSDL. Whereas in the agent platforms the Directory Facilitator particularly stores the service description language and does not maintain any other information as the DF-AgentDescription is quite comprehensive.

The mechanism through which the services are retrieved varies between both technologies also. While performing a search in the UDDI, a web service client invokes a method that is specific to the WSDL related description or the other business related information stored by the UDDI. Whereas, it is somewhat different in the case of Multi-agent Systems as the agent has to initially substitute the required service parameters for the DF-AgentDescription object that is used for service description. As soon as the Directory Facilitator receives the search

request it performs the comparison between the request that was sent and the service related information that was stored with it.

!

### 5.1.2 Analysis of communication protocols

This section addresses the translation difference between the communication protocols that are used as a medium while communicating between the technologies, which are agents and Web services. In FIPA compliant Multi-agent Systems, agents use Agent Communication Language (ACL) whereas in Web services Simple Object Access Protocol (SOAP) is used as the communication protocol.

ACL	SOAP
Encoding	Encoding Style
Sender	From
Receiver	To
Content	Body
Performative	Fault
Reply-to	None
Ontology	None
Protocol	None
Conversation-identifier	None
Reply-with	None

**Table 5.2.** Comparison between communication protocols!!

Table 5.2 provides a comparative analysis of both protocols that are used by both these technologies. Web services use simple object access protocol as a mode of communication whereas the agents use the agent communication language for communication. !

The SOAP message contains the encoding details according to the specifications that are provided by W3C. Whereas, the ACL message contains in itself the parameters that are required according to the specifications of FIPA. The sender and receiver information is contained in the underlying HTTP protocol header that is used by SOAP whenever it is required. Figure 5.1 shows the conversion steps that are involved while converting ACL to SOAP communication protocol.

While dealing with ACL, the sender field contains the information about the agent that propagated the message and the receiver field contains the information of the recipients.

```

get ACL message
  get Sender & Receiver
    map Receiver with SOAP-Endpoint
    map Sender with Gateway address
get ACL Content
  get ontology

  if ontology has AgentAction schema instance, then
  {
    map AgentAction Schema name with Operation name
    Parse SLContent string and map to input parameter values of SOAP
  }
  if ontology has Predicate schema instance, then
  {
    map Predicate Schema with Operation name
    Parse SLContent string and map to output values of SOAP
  }

get ACL Performative and map to SOAP Fault
return SOAP message

```

**Figure 5.1:** ACL to SOAP conversion steps

!

The content of the message and operations are to be invoked and what were the inputs and output are contained within the body of the SOAP message. SOAP contains a section named 'Fault' in its body to inform the client about any errors/exceptions that occurred at server side, whereas in the case of ACL, it supports a number of performatives. List of features of SOAP ends here, but ACL does not. ACL is very comprehensive as compared to SOAP. It supports many other features as well which are given below.

!

There is an attributed named 'Reply-to' which is helpful during negotiation in which messages are to be directed to the agent named in the reply-to parameter, instead of to the agent named in the sender parameter. The ontology attribute contains the name of the ontology which is used to give meanings to symbols used in message content. The protocol attribute indicates the name of Interaction Protocol for negotiation being employed. There can be multiple negotiations going on among multiple agents therefore the conversation-identifier is used to identify individual conversation with multiple Agents. The Responding Agent uses reply-with attribute to reply this message. To perform the SOAP to ACL communication protocol translations it is necessary to understand the data contained in above mentioned fields and to retrieve this data the conversion steps are depicted in Figure 5.2.

!



```

get SOAP message
get SOAP/HTTP Header
    map SOAPEndPoint to Sender attribute of ACL
    map HTTP Sender to OntologyAgent ID in Receiver
get SOAP Body
    get Operation Name
        if it is SOAP request, then
        {
            map Operation Name to AgentAction Schema in ontology
            get input parameter names, values and map to SLContent
        }
        if it is SOAP response, then
        {
            map Operation Name to Predicate Schema in ontology
            get output parameter name, value and map to SLContent
        }
get SOAP Fault and map to ACL performative
initialize Reply-to with null
initialize Interaction-Protocol, conversation identifier, reply-with, in-
reply-to, reply-by with null
return ACL Message

```

!

**Figure 5.2:** SOAP to ACL conversion steps

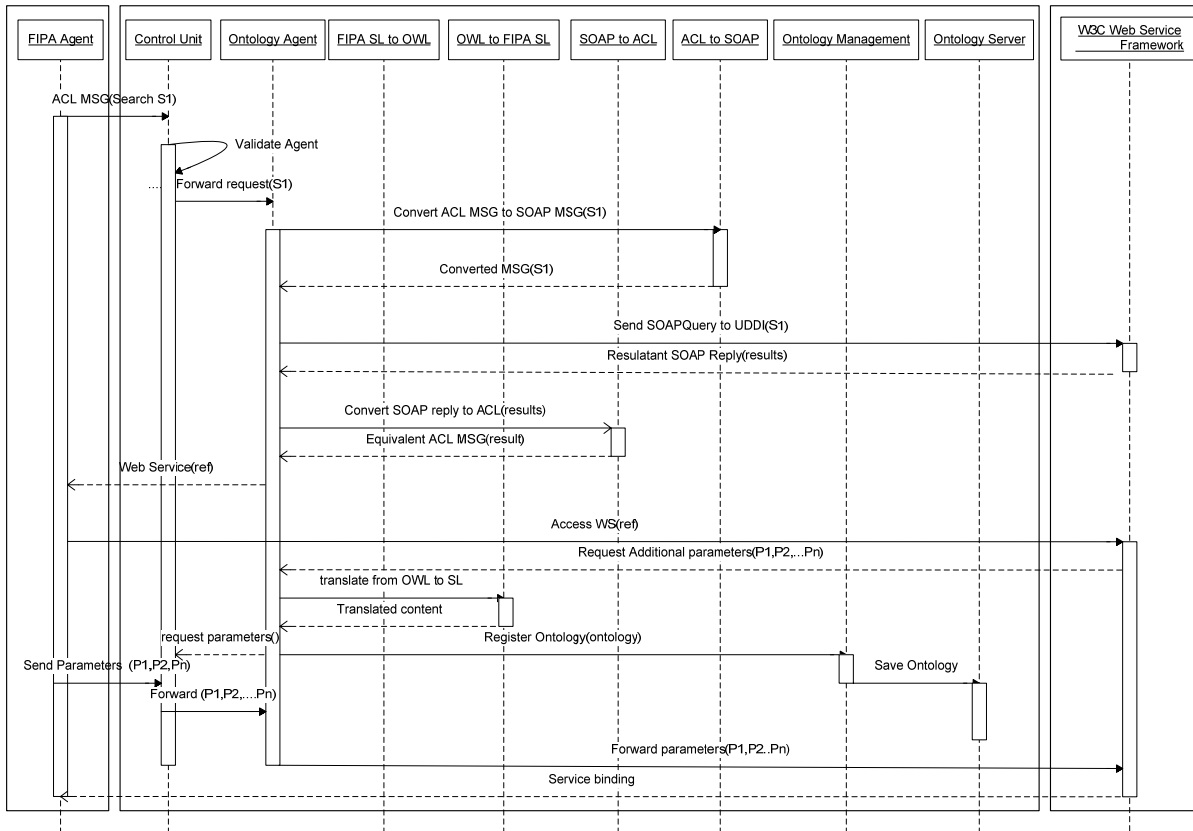
## 5.2 Implementation details of the proposed system

This section specifies the implementation details of the proposed system middleware; the aim is to solve the technical challenges that were faced during the translation between agents and OWL based Web services. A number of scenarios have been devised in order to understand the complete functionality of the system. The goal is to define the sequence of steps that have to be followed and what are the participating components required to perform a particular task. There are two main scenarios that are further bifurcated into a number of scenarios to understand the functionality at a fine grain level.

## 5.3 Agents interacting with the semantically enabled Services

This section highlights the detailed design of the scenario where FIPA compliant agents can interact with the semantically enabled Web services. The Ontology Gateway has been designed in a way to facilitate the agents to discover the services that have been specified in OWL. Figure 5.3 depicts a generic scenario where the agent initiates the communication with the OWL based Web services. In order to perform a search or discover the services that are available, the agent needs to have the basic vocabulary in terms of ontologies to carry out further processing.

The agent can perform a local search in its own directory facilitator in order to retrieve the services that are registered with it. These services can be limited and may not contain the data that was required by the agent. Therefore the agent's request is forwarded to the control unit for further processing. As soon as the control unit receives the request it forwards it to the Ontology Agent for further processing after validating that the request is from an authentic agent (an authentic agent is the one registered with in the platform).



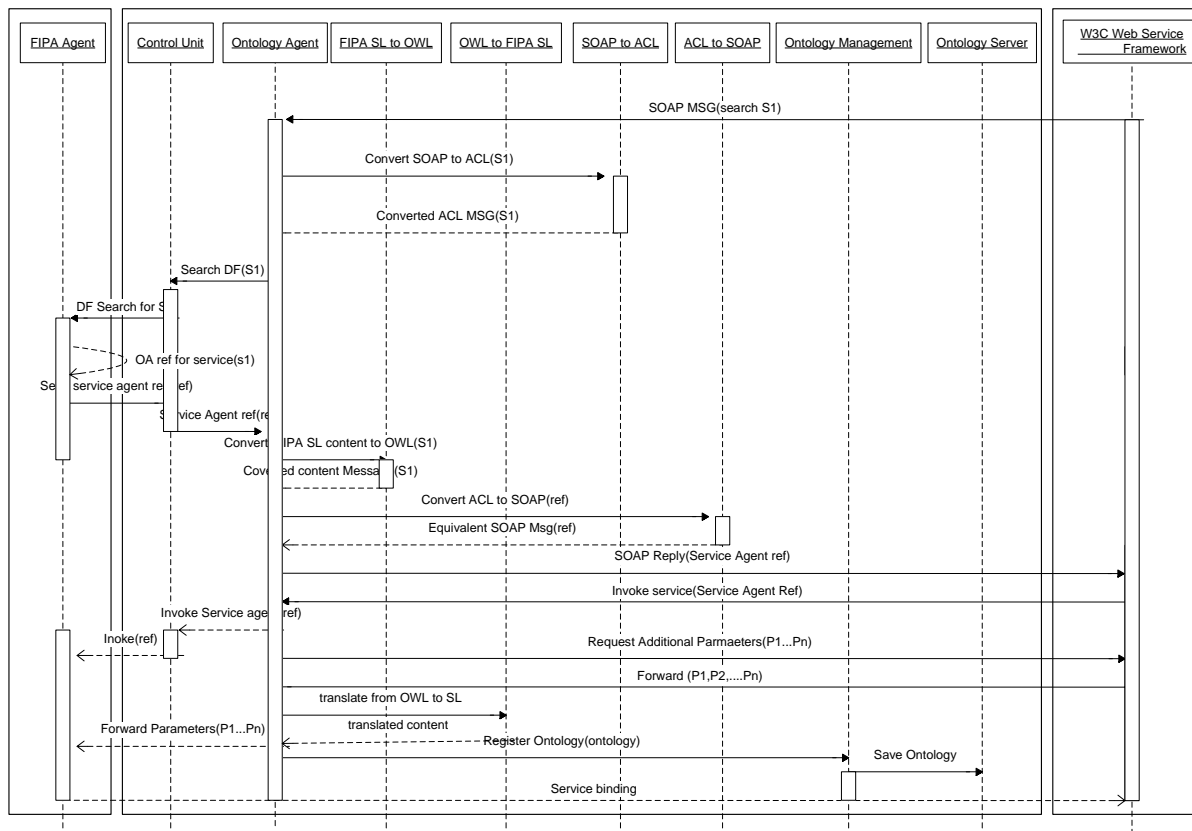
**Figure 5.3:** Agents interacting with the semantically enabled Web Services

In order for an agent to retrieve the web service the protocol transformation is necessary as agents use ACL for communicating with one another whereas the Web services use SOAP as a protocol. Therefore as soon as the ontology agent receives a request from the control unit which is in ACL it forwards that request to ACL to SOAP converter that generates the equivalent SOAP message that can be forwarded to the web service.

Moreover, the contents can be transformed from SL to OWL and appended in the message that is forwarded. After retrieving the results from the UDDI the ontology agent is responsible for translating the data from OWL to SL and also for resolving the protocol transformation that is SOAP to ACL. As the ontology agent is now aware of the existence of

the service and can access the published service, the WSDL service descriptions are then retrieved through the ontology agent to access and bind with the service.

The Ontology Agent is then responsible for saving the ontology to the ontology server. The agent can access these ontologies to resolve the semantic issues and can use them for updating their information regarding the different types of schemas, i.e. concept, predicate and aggregate schema. The whole ontology can be translated from OWL to FIPA ontology based on the mappings provided in Table 4.1 and Table 4.2.



**Figure 5.4:** Semantically Enabled Services/Grid Client Communicating with the Agents

## 5.4 Semantically Enabled Services/Grid Client Communicating with the Agents

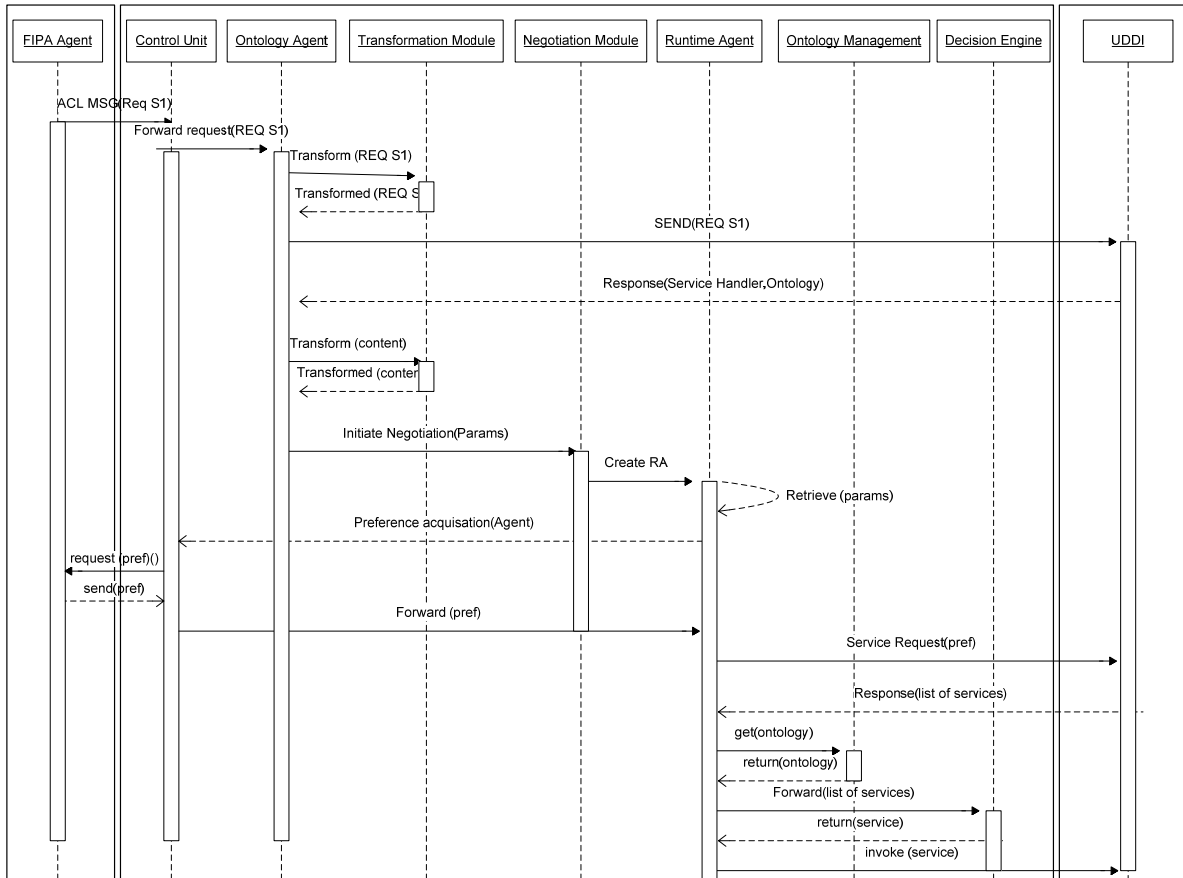
There are certain occasions when a service wants to know about the services available on the agent platform. According to the vision of the Autonomous Semantic Grid the services can be provided by anyone throughout the globe and the standards like OGSA and OGSi will

facilitate the identifying and consuming of the services that are present even on someone's desktop. Keeping this context in mind a detailed description of a scenario is given where a semantically enabled service or even a grid client can consume a service that is hosted on any Multi-agent Systems.

A scenario is presented in figure 5.4 where some Service or Grid client wants to retrieve a service that is hosted on Multi-agent Systems. In Multi-agent Systems the services are published in Directory Facilitator through which they can be discovered easily. In order for a web entity to discover services from an agent platform a SOAP request is sent to the Ontology Gateway where it is received by the Ontology Agent who continuously monitors the incoming request and forwards them to the appropriate module for further processing. As this request is coming from an outside entity that uses SOAP as a protocol, this request is forwarded to the SOAP to ACL modules which converts and generate it in an equivalent ACL message that is interpretable by the agents. The Ontology Agent then forwards this information to the Directory Facilitator.

If the service is found then the Directory Facilitator will generate an ACL message with the reference of the agents that maintain the service. This content is then forwarded to the Ontology Agent, that forwards it to the FIPA SL to OWL converter where different schemas, like concept, aggregate and action schemas, are mapped into equivalent OWL classes and forwarded back to the Ontology Agent which then sends it to ACL to SOAP for the protocol transformation so that the query can be forwarded to the client that requested the information about the service.

A single exchange of message might not be sufficient and therefore, a number of messages can be exchanged. For example the agent hosting the service might request additional parameters from the service which will be in OWL format in that case, they are forwarded to the Ontology Agent that forwards it to OWL to SL converter to process it and the translated ontology is stored by utilizing the ontology management module and is saved for future use and the binding is established between service provider and consumer.



**Figure 5.5:** Negotiation between agents and Web services.

## 5.5 Negotiation between agents and Web services

In order to have a meaningful conversation between the agents and Web services, the negotiation module plays a vital role in carrying on this process as depicted in figure 5.5. The Ontology Agent needs to know the reference to the service and the ontology that has to be used and by whom the request has been made. After processing the request through the transformation module which can be either the content translation or the protocol transformation the information is sent to the Negotiation module to initiate the negotiation process.


Once the prerequisites for the implementation of negotiation protocol are set by the Ontology Agent, it is now time for the negotiation module to initiate the negotiation process according to a formalized protocol i.e. Contract Net Protocol. The runtime agent acts as the initiator of the protocol and sends a service request/proposal with the requested agent

preferences. These user preferences were retrieved with the help of the requesting agent reference that was initially passed to the negotiation module by the Ontology Agent.

Along with getting these parameters from the requesting agent, the negotiation module needs to have the agent's ontology as well, so as to get a semantic understanding of what these parameters mean based upon a common vocabulary. It can also be accessed with the help of the agent's reference that this module has. Such an understanding of the semantics will help the negotiation module to map the information that the requesting agent possesses to the information required by the service method that is going to be invoked as a result of sending a service request. As a response to this, the participant would return all possible services (proposals) to the Runtime agent. Each of these responses is the handle of a Semantic web service that closely matches the requested service description i.e. if a request was made for retrieving flight reservation services then the services providing flight reservation will be retrieved according to the search criteria, which can be based on date, cost of the ticket etc.

These responses are passed to a decision engine. We assume that the decision engine is an independent component that uses artificial intelligence and semantic deduction rules to choose the best possible option out of many as for the closest match with user preferences. The decision engine sends the chosen option back to runtime agent which invokes the corresponding service. The service is executed as a result of this invocation and a response is sent to the runtime agent indicating whether the service has succeeded or failed. The runtime agent forwards this response to the negotiation module which stores the results in its knowledge base and returns control to the Ontology Agent. Finally, the requesting agent is informed of the results of negotiation along with all associated details through the control unit and this is how the negotiation process is carried out.

Contract Net Protocol



The Contract Net Protocol is an interaction protocol, where one agent initiates the request and acts as a central body and wishes to perform some task whereas the other agents submit proposals for cooperation to perform that task, while some refuse. Negotiation process is carried out by the agents who proposed for cooperation.

## 5.6. Validation

The previous section provides the behavioural aspect of the system. However, this section gives the validation results of the Ontology Gateway when tested for the translation of sample ontologies. A prototype implementation of some components has been proposed and developed at NUST-Comtec lab. A test bed composed of a machine having a Tomcat web server, Protégé OWL installed in it and Jena 2.4 to enable the web server to create and deploy OWL Web services. On the second machine, the proposed Ontology Gateway was installed to provide the required translation, registration and querying of translated ontologies which is a part of FIPA compliant agent platform SAGE (Ahmad et al., 2004), (Ghafoor et al., 2004), (Khan et al., 2005), on which an agent providing some services was deployed. For testing purposes, the Ontology Gateway was integrated with the SAGE platform and all the requests for communication were routed through this single machine. The ontology gateway converted the semantic annotations into a comprehensible form that was understood by agents and web services. In this test bed the machines used were Pentium IV core 2 duo with Windows XP and 4 GB RAM. The current version of the Ontology Gateway was tested to explore communication process and to demonstrate the performance of the Ontology Gateway itself.

FIPA Semantic Language representation of Concept Schema	OWL representation of the Concept Schema
<pre>ConceptSchema personSchema = new ConceptSchema(PERSON);</pre>	<pre>&lt;owl:Class rdf:ID="person"&gt; &lt;rdfs:subClassOf rdf:resource="PEOPLE_ONTOLOGY"/&gt;&lt;/owl:Class&gt;</pre>
FIPA Semantic Language representation of Predicate Schema	OWL representation of Predicate Schema
<pre>PredicateSchema fatherOfSchema = new PredicateSchema(FATHER_OF);</pre>	<pre>&lt;owl:ObjectProperty rdf:ID="fatherOf"&gt; &lt;rdfs:domain rdf:resource="#father"/&gt; &lt;rdfs:range rdf:resource="#children"/&gt; &lt;rdfs:subClassOf&gt;&lt;owl:Restriction&gt;&lt;owl:onProperty&gt; &lt;owl:ObjectProperty rdf:about="#fatherOf"/&gt; &lt;/owl:onProperty&gt; &lt;owl:someValuesFrom rdf:resource="#father"/&gt; &lt;/owl:Restriction&gt;&lt;/rdfs:subClassOf&gt; &lt;rdfs:subClassOf&gt;&lt;owl:Restriction&gt;&lt;owl:onProperty&gt; &lt;owl:ObjectProperty rdf:about="#fatherOf"/&gt;</pre>
FIPA Semantic Language representation of Agent-Action Schema	OWL representation of Agent-Action Schema
<pre>AgentActionSchema marrySchema = new AgentActionSchema(MARRY); marrySchema.add(HUSBAND, husbandSchema); marrySchema.add(WIFE, wifeSchema);</pre>	<pre>&lt;owl:Class rdf:ID="action"&gt; &lt;owl:unionOf rdf:parseType="Collection"&gt; &lt;owl:Class rdf:about="#husband"/&gt; &lt;owl:Class rdf:about="#wife"/&gt; &lt;/owl:unionOf&gt;&lt;/owl:Class&gt;</pre>
FIPA Semantic Language representation of Aggregate Schema	OWL representation of Aggregate Schema
<pre>AggregateSchema childrenSchema = new AggregateSchema(); childrenSchema.addSuperSchema(personSchema);</pre>	<pre>&lt;owl:Class rdf:ID="children"&gt; &lt;rdfs:subClassOf rdf:resource="#person"/&gt; &lt;/owl:Class&gt; &lt;owl:Class rdf:ID="Aggregate"/&gt; &lt;owl:equivalentClass&gt; &lt;owl:Class rdf:resource="#children"/&gt; &lt;/owl:equivalentClass&gt;&lt;/owl:Class&gt;</pre>

**Table 5.3.** Extended mappings between FIPA Ontology and OWL Ontology tags

The ontologies are composed of classes, relationships, properties and constraints which are translated from FIPA SL to OWL and from OWL to SL based on mappings provided in the previous chapter. However, Table 5.3 shows some of these translations of people ontology defined in FIPA to OWL ontology and vice versa. To represent any concept, we use the concept schema clause in the FIPA ontologies definition. Similarly for defining relationships between two concepts, the predicate concept is used. If the agent can perform



any particular actions then the Agent-action schema can be defined. Aggregate schemas are defined for groups of individual classes which inherit their capabilities from a parent class. The Concept schema, Predicate schema, Agent-action schema and Aggregate schema are translated to construct the equivalent OWL ontology and vice versa.

### 5.6.1 FIPA to OWL Translation

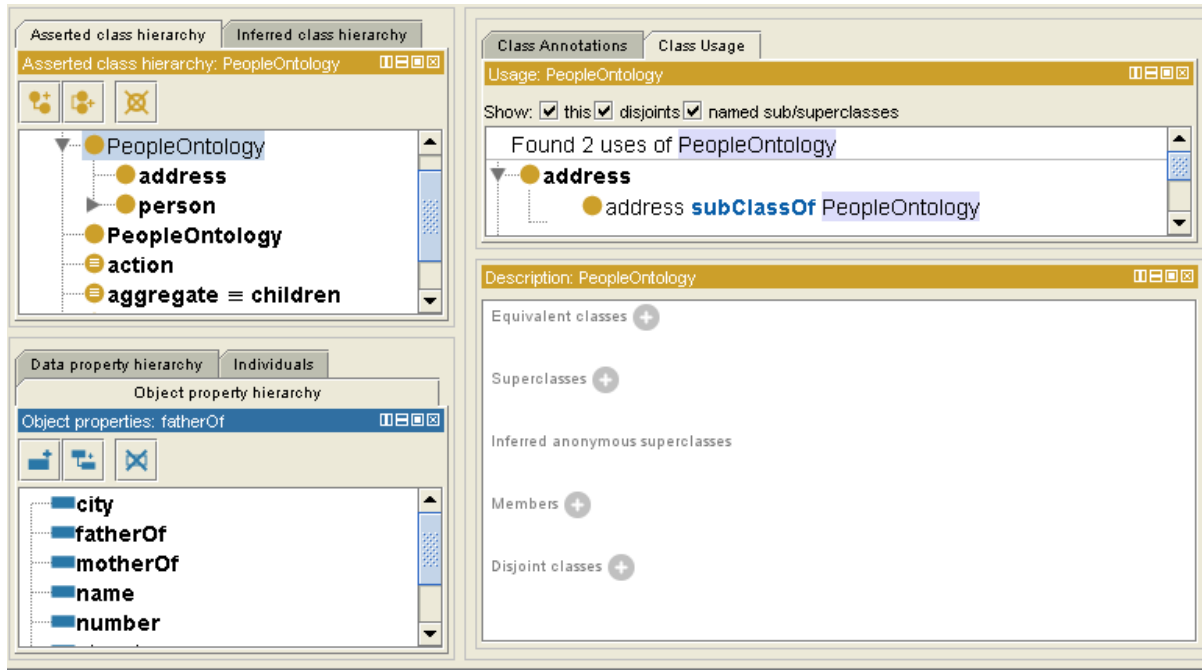
Figure 5.6 shows the algorithmic steps that are performed while the concepts from FIPA Ontology have to be translated into equivalent OWL ontology concepts and then the messages are propagated to the Ontology Agent.

```
get ACL Message
  get Sender and Receiver from ACL Message
pass the message to ACL to SOAP Component
  get the equivalent UDDI based SOAP query
retrieve the address of the Ontology from UDDI
  get OWL concepts (embedded in a SOAP message)
Send the SOAP message with embedded content to SOAP to ACL Component
get the transformed ACL message
  Send the transformed message to the Ontology Agent
  Extract OWL ontology from the content of ACL message
  Pass the OWL ontology to the OWL to FIPA component

/*The OWL to FIPA component, converts the OWL ontology into FIPA Ontology
  Extract the classes, restrictions and other information from the OWL
  Ontology by sequentially parsing each line generate an equivalent FIPA
  Ontology */
```

**Figure 5.6:** FIPA to OWL conversion steps

The People Ontology shown in figure 5.7, defined in FIPA SL is used to validate our system component, i.e. FIPA SL to OWL Component. In order to make translations between FIPA SL to OWL, there were certain challenging tasks that were catered.



**Figure 5.7: People Ontology in Protégé**

\* Ref. Please see APPENDIX A for Detailed Ontology

FIPA SL implementation of SAGE provides explicit support for defining concepts, relationships, aggregate classes and agent actions through the use of Concept Schema, Predicate Schema, Aggregate Schema and Agent-action Schema, that is not supported in OWL as it was not designed keeping the agent systems in view. The implementation component of FIPA SL to OWL translates these expressions into their equivalent OWL mappings in terms of classes, restrictions and attributes by preserving the semantics of mappings provided in this thesis. Figure 5.7 shows the snapshot of Protégé where certain concepts and their properties were shown.

### 5.6.2 OWL to FIPA Translation

Figure 5.8 illustrates the algorithmic steps that are involved while translating from the OWL concepts that are being sent by the SOAP message into their equivalent FIPA concepts.

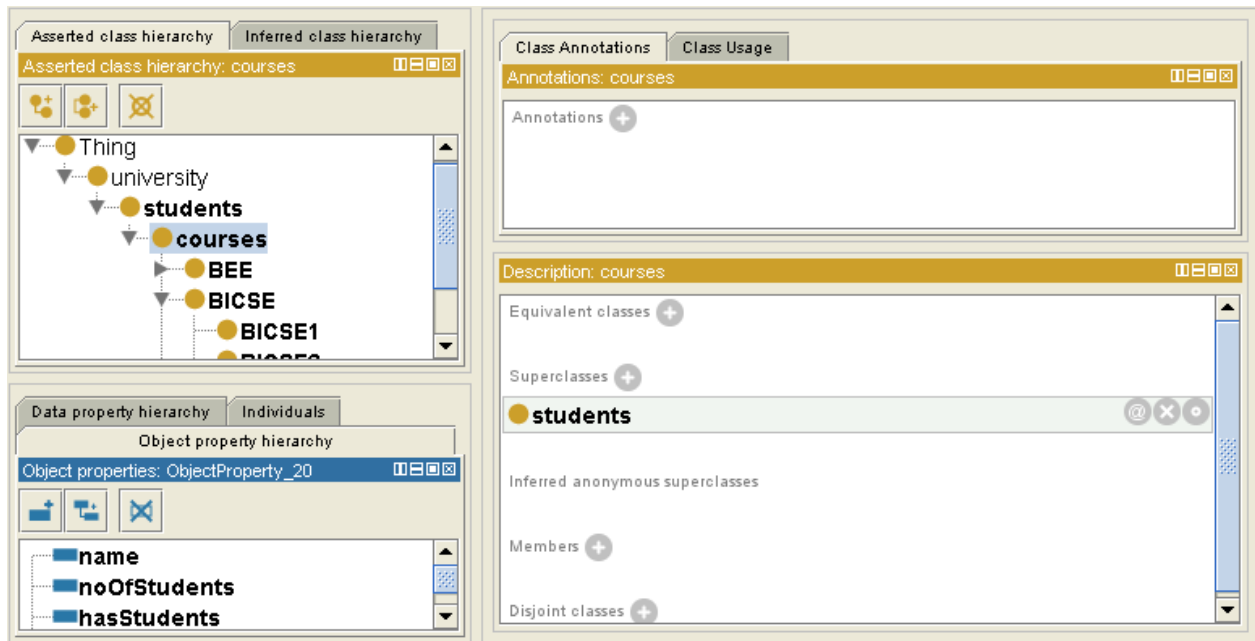
Similarly, the other components, of Ontology Gateway were also checked by translating other sample ontologies containing different classes, and subclasses; their properties and restriction on them, and their details are given based on their relationships with each other, which were converted to the equivalent ontology in FIPA SL. Figure 5.9 shows a sample ontology with concepts and their relationship with each other in Protégé.

```

get SOAP Message
  get Sender and Receiver from SOAP Message
pass the message to SOAP to ACL Component
  get the equivalent DF based ACL query
  send the message to Ontology Agent
search the address of the Ontology from DF
  if not found
  {
    pass the ACL message to the remote container of the Agent Platform to
    check if the service exists there
    if found on Remote Agent container
    {
      return the result to the Ontology Agent
    }
  }
  elseif found
  {
    return the address of the service to the Ontology Agent
  }
  }
get FIPA concepts(embedded in a ACL message)
extract the FIPA concepts from the ACL message
pass the FIPA ontology to the FIPA to OWL component
retrieve all the Concepts, PredicateSchemas, AggregateSchemas, AgentAction
Schema and attributes from the FIPA Ontology by sequentially parsing each
line
Generate an equivalent OWL ontology using
Embed the transformed Ontology in an ACL message
send the ACL message to ACL to SOAP component
Send the transformed SOAP message to the Request Client

```

**Figure 5.8:** FIPA to OWL conversion steps



**Figure 5.9:** University Ontology in Protégé

\* Ref. Please see APPENDIX B for Detailed Ontology

```
(query-ref
:sender (agent-identifier :name i )
:receiver (set (agent-identifier :name j ))
:content "( (all ?X ( STUDENT_OF :student ( :STUDENT :age
"24" :name "Pierre" :country France :city Vannes))) )"
:language fipa-sl2
:ontology University
:reply-with query1)
```

**Figure 5.10:** ACL Query to retrieve information for a particular entity.

Figure 5.10 represents the ACL messages that are generated by the agents to query the information regarding a particular entity from a translated ontology.

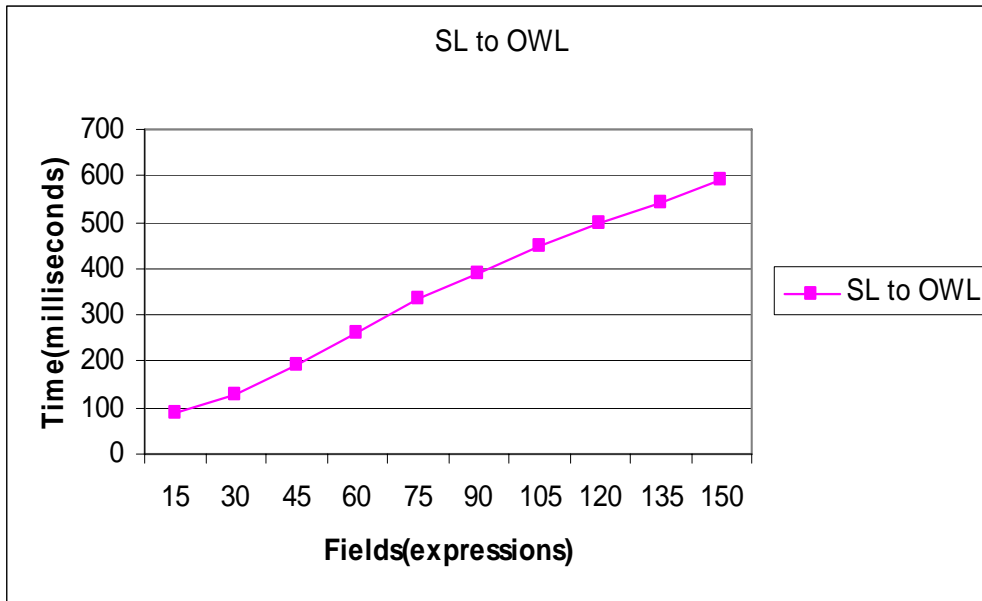
## 5.7. Performance Evaluation

This section contains the results of the prototypic implementation of the ontological conversion modules in terms of the amount of time for ontology to be converted into its equivalent OWL or FIPA. The evaluation was done on the basis of the number of concepts that were used in the ontologies and the amount of time taken in the conversion of these ontologies. Both of the graphs show the linear behaviour in translation, which indicates that the conversion modules works satisfactorily for a limited scale of ontology conversion. The ontologies that were tested for now have a uniform structure and common vocabulary and range from 1000 to 2000 lines.

### 5.7.1 FIPA SL to OWL Translation Analysis

Figure 5.11 shows the ontology translation time for the conversion of FIPA Semantic Language to the Web Ontology Language. When discussing the FIPA ontologies it should be clear that the information is in the form of java objects and is binary based. The ontology size varies on the basis of the concepts that it contains and the time taken in order to convert these concepts into equivalent OWL contents. The translation time as shown in figure 5.11 shows that increasing the classes or their properties does not affect the performance of the overall system. Therefore, if the Ontology Gateway is integrated with any FIPA compliant Multi-agent system will not cause an overhead to the system. One important aspect to be mentioned is that while translating from FIPA SL to OWL the accuracy of the translations was 94

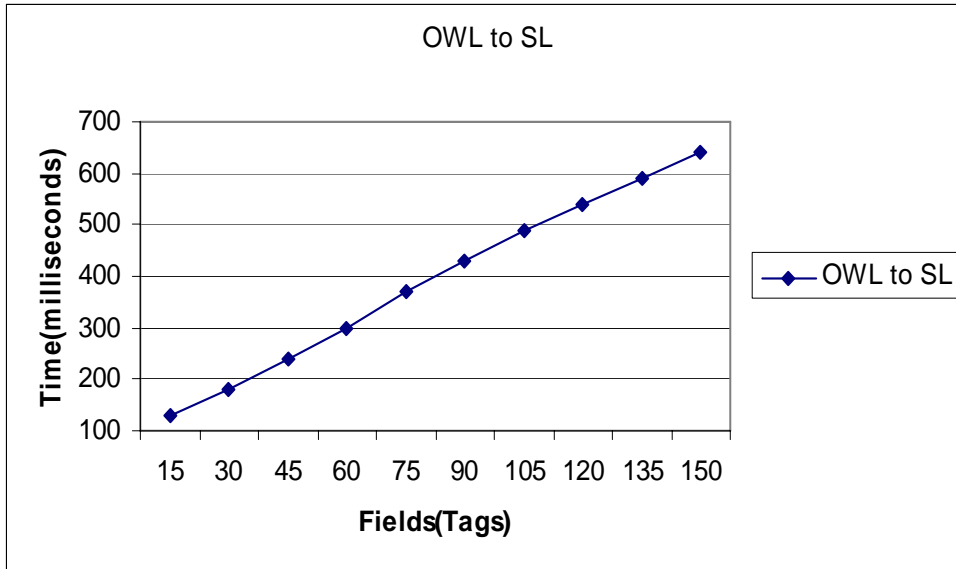
percent. The rest of the 6 percent is the content that is relevant to agent behaviour and is not required as we are using the web services for communication on the other end.



**Figure 5.11: FIPA SL to OWL Translation**

### 5.7.2 OWL to FIPA SL Translation Analysis

The section explores the time consumed in translation of the OWL ontology received from the web entity into its equivalent FIPA ontology. OWL ontologies are plain text based files and not in the form of objects or binary information therefore their parsing takes a little more time than the FIPA ontologies. The ontological translation time varies on the basis of the number of tags that the ontology contains. Figure 5.12 shows the ontological translation time for the conversion of OWL to FIPA SL ontologies.

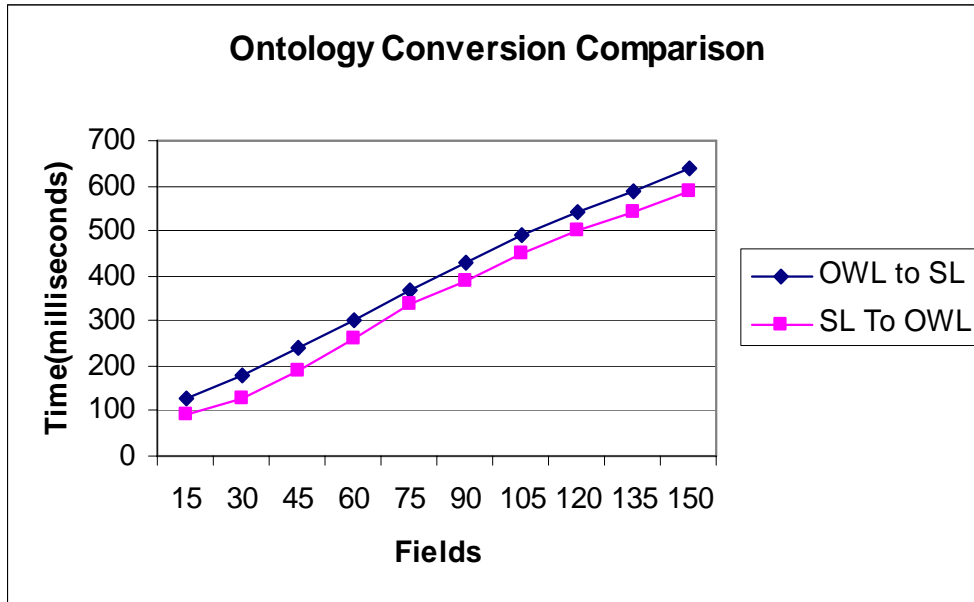


**Figure 5.12: OWL to FIPA SL Translation**

The time taken from OWL to FIPA SL is not only a little more but the accuracy is almost 90 percent somewhat less than that of FIPA SL to OWL translation this is because OWL is based on description logic and has comparatively less expressivity as discussed in the previous sections.

### 5.7.3 Ontology Translation Comparison between OWL and FIPA

Figure 5.13 shows the comparative analysis of time taken while converting the OWL to FIPA ontology and vice versa. As previously discussed, the information in the FIPA ontology is in the form of objects and is binary information, thus it takes comparatively less time in translation than that of OWL. OWL, however, contains the information in the form of plain text. Moreover, another factor involved in the overhead time is the delay caused by translating extending the query for checking the parent of a particular concept.



**Figure 5.13:** Ontology Translation comparison between OWL and FIPA

The results have shown that increasing size of the ontology does not affect the performance of system and has proven to be very effective. Therefore, when the Ontology Gateway is integrated with the Multi-agent Systems or Web/Grid clients it will not affect the performance of the system. Moreover, the scalability issue has not been considered in this implementation of the system and is left for exploration in our future work.

## 5.8 Classes of Applications

The previous section highlights the implementation specific details of the proposed architecture discussing the details regarding the scenarios analyzed for the communication between the agents and the Web services. Principally the Ontology Gateway can be used in a number of applications which can vary from, but are not limited to:

- Grid Systems
- E-Science
- E-Business
- E-Health
- E-Markets
- Information based systems

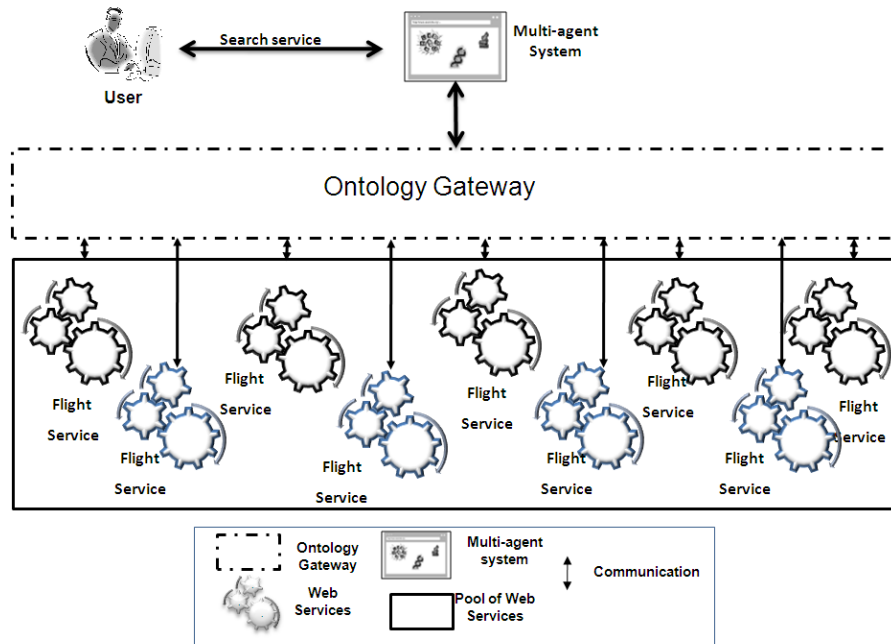
The current implementation of the Ontology Gateway has been implemented as a standalone application but in future we aim to integrate Ontology Gateway with the Grid middleware's like Globus toolkit for the categorization of data and storage intensive resources. As shown by the results in the previous section that capabilities of the Ontology gateway will not only apply the agent capabilities in the discovery and accessing process of

services located in the Grids but will also provide the monitoring and utilization of resources with minimum human intervention.

Based on our interest in this section we have presented two case studies as principal candidates for implementation of the proposed architecture and its verification. The first one is about the business to consumer (B2C) and has been formally specified using the  $\pi$ -ADL.Net. Whereas, the second case study gives an overall view of the Ontology Gateway working in a business to business (B2B) environment.

### 5.8.1 Flight reservation using Ontology Gateway

In this section, a flight reservation service case study has been discussed from a business to consumer perspective. The case study is shown in figure 5.14 and is explained based on the architectural components of Ontology Gateway and the functionality they will perform. An agent wants to retrieve a flight reservation service based on different criteria. The flight reservation service provides the services for online flight reservation. While choosing one of the available flights, factors such as time, fare, and date are negotiable parameters. We aim to carry out this negotiation between the requesting agent and the flight service. Given below is a sequence of how control shall be exchanged among various modules of an Ontology Gateway and what actions shall take place in between.



**Figure 5.14:** Flight reservation using Ontology Gateway



First of all, the meeting requesting agent sends a message to Ontology Gateway expressing its desire to engage in negotiation. The message is received by the Control Unit. The content of the message contains the description of a flight service. The Control Unit forwards this information to the Ontology agent which after translating the requested query sends a search request for the UDDI. The handle of the service is returned to Control Unit with the help of which the Control Unit fetches the profile of the service and its ontology.

The Control Unit feeds this ontology to its OWL to FIPA ontology translator, thus acquiring the FIPA Ontology equivalent of the OWL ontology. Then, the Control Unit passes the reference of the meeting scheduler agent and its FIPA ontology along with the profile of the semantic web service to the Negotiation Module. Negotiation Module creates a Runtime agent which extracts all negotiable parameters of the web service profile. Next, Negotiation Module acquires the user preferences for all the negotiable parameters with the help of the meeting scheduler reference that was sent to it by the Control Unit.

### Business to Consumer (B2C)



Business to Consumer is a way of doing business that aims at providing products and services to the consumers in a marketplace.

It is now required to invoke the web service's method that takes user requirements as input parameters and returns a list of matching flights. These parameters shall be passed in a format and sequence expressed in the service profile. The Runtime agent invokes the Web service. As a result, the Runtime agent shall receive a list of flights, closely matching user criteria. These responses are equivalent to the "proposals". The Runtime agent forwards these responses to the decision engine to choose a particular flight and respond to the Runtime agent. Runtime agent then invokes the chosen service. A response is sent to the Runtime agent indicating that the service has succeeded which in turn informs the Negotiation module. The Negotiation module returns control back to the Control Unit after storing results permanently for future use. The Control Unit informs the meeting scheduler agent of the reserved flight and all the associated details and the negotiation session comes to an end.

- 1.
2. **value** service is **abstraction** (reqService : **view**[service\_name : **String**,
3. params : **sequence**[**view**[name : **String**, val : **String**]],
4. numOfParams : **Integer**, message : **String**)
5. {
6. reqServiceConn : **Connection**[**view**[service\_name : **String**,
7. params : **sequence**[**view**[name : **String**, val : **String**]],

```

8.  numOfParams : Integer, message : String]];
9.
10.
11.  //register service request
12.  unobservable;
13.
14.  reqService = view (service_name : "Additional Parameters Request",
15.                    params : sequence (view (name : "Name", val : ""),
16.                    view (name : "Address", val : ""),
17.                    view (name : "Phone", val : "")),
18.                    numOfParams : 3, message : "");
19.
20.  via out send "***service**\n\n";
21.
22.
23.  compose
24.  {
25.      via out send "--Sending additional parameters request--\n";
26.      via reqServiceConn send reqService;
27.  and
28.      via reqServiceConn receive reqService;
29.      via out send "***service**\n\n";
30.      via out send "--Recieving additional parameters--\n";
31.      //process service request
32.      unobservable;
33.      via out send "--Initiating service--\n";
34.      reqService::message = "Service successfully initiated";
35.      via reqServiceConn send reqService;
36.  }
37.} //end service

```

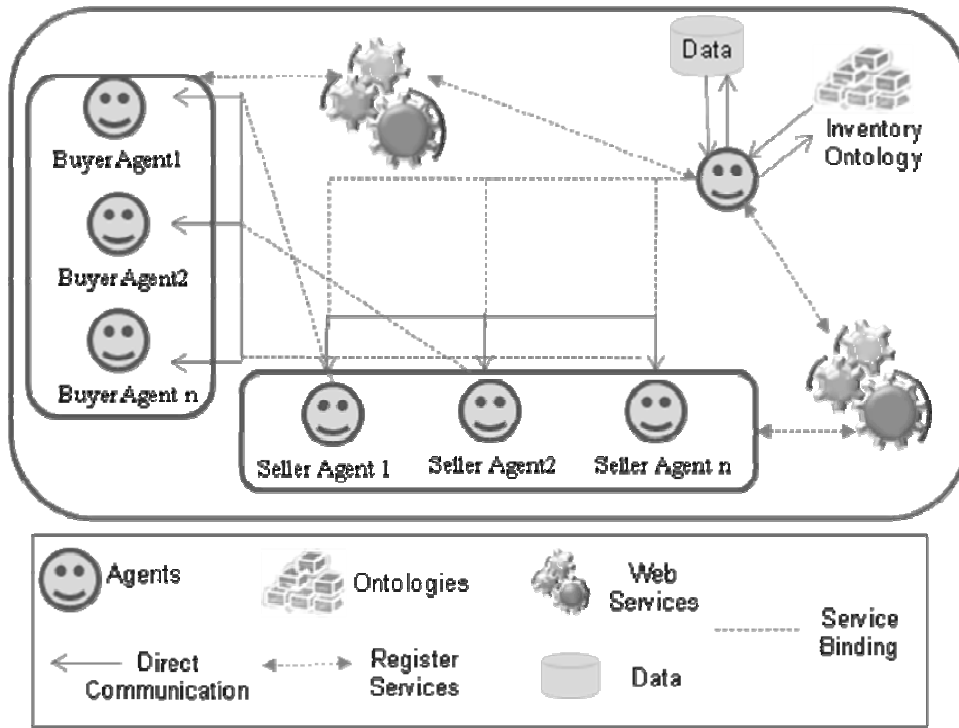
**Figure 5.15:**  $\pi$ -ADL code for a simple Web service

Figure 5.15 depicts the  $\pi$ -ADL specification of a Web service which has a name and certain other parameters. Service has been defined as an *abstraction* and a *view* is created for the service description and number of parameters as there could be multiple parameters associated with the services retrieved, therefore each service name and its value have been defined as a view. The number of parameters defines the fields required to bind to this service. For example, the number of parameters in the above example is three containing *name*, *address* and *phone* as the required fields. Similarly, some additional parameters can also be requested once the service has been invoked by the Runtime agent.

### 5.8.2 Semantically Enriched Agent Application using Ontology Gateway

An agent based application has also been analyzed as an example to show the working and significance of the proposed architecture from a business to business perspective. It has been named “Semantically Enriched Supply Chain Management” and is depicted in Figure 5.16. It

uses the ontology gateway for its interaction with the OWL Web services. It demonstrates the interoperability between FIPA-compliant software agents and the OWL-based Web services. Different agents communicate with each other and use information provided by the Web services to plan and coordinate their actions.



**Figure 5.16:** Supply Chain Management System Using Ontology Gateway.

Supply chain management (SCM) is the process of planning, implementing, and controlling the operations of the supply chain to satisfy customer requirements in an efficient and cost effective manner. SCM spans all movement and storage of raw materials, work-in-process inventory, and finished goods from point-of-origin to point-of-consumption (Maherzi, 1997). The supply chain consists of a worldwide network of factories, suppliers, warehouses, distribution centres, and retailers through which materials are acquired, transformed, and delivered to customers.



The software architecture is developed for managing supply chains at different levels. The architecture proposes that software agent technology should be used, and in such an application various agents will interact with one another to perform the planning and execution responsibilities assigned to them. Using Ontology Gateway, these agents can acquire data that is not available in their default Directory Facilitator or remote platforms. We developed a nontrivial agent-based supply-chain architecture which supports simple cooperative work and management.

This agent-based Supply Chain Management system only covers just the B2B (business-to-business) aspect of the supply chain. The main activities of buyer agents in B2B are to avail the economical offer in the market, for that it has to communicate with various agents around and on remote platforms. In this application, the buyer agent is able to find the lowest cost of products/services it wants to buy. A supplier agent aims to attract a buyer agent and then sell its goods to it. For this purpose it interacts with various buyer agents and submits its rates. If the buyer agent feels that this is the lowest bid, then the appropriate supplier agent is contacted.

In detail, the buyer agent checks the stocks of a company in a warehouse, and if it feels that the stock is below a certain level, then it decides to place an order for the required product. Each participating agent has its own ontology which manages the information in its own way and can have semantic heterogeneity. To place an order, it contacts various supplier agents and asks them to submit their bids. Web services are also contacted through Ontology Gateway. They are sent a message in which they are asked to submit their bids. At the end of the day, all supplier agents and Web services submit their bids. The agent or web service with the lowest price is requested to dispatch the product to its customer. Supplier agents, when receiving a message concerning the issues of submitting the rates, contact the sender agent and send them their rates and bids. At the end of the day, all the bids are checked and the agent that has submitted the best bid will be contacted and the order will be placed.

The OWL based Web services are registered in UDDI. The buyer agent will send its request to the Control Unit which will contact the Ontology Agent to search for OWL Web services with the lowest inventory price. The Ontology Agent will translate the request to the SOAP message with the help of ACL to SOAP component and forwards it to UDDI. The search results are returned to ontology agents after translation from the SOAP to ACL component. The Ontology Agent accesses the OWL web service, translates the ontology from OWL to FIPA with the help of the OWL to SL component and then saves the translated ontology on the local web server. The Ontology Agent sends the reference of the translated ontology to the agent which generated the request. Thus, ontologies are translated from OWL

to FIPA and used with the same semantics in the FIPA compliant Agents as they are defined in OWL.

Once the buyer agent has acquired the reference of the supplier agent hosting the service it will then engage itself in the negotiation process through the use of the negotiation module which will define a run time agent for the process of communicating with the supplier and additional information.

## 5.9 Summary

In this chapter the results of the current thesis research were presented in the form of three main scenarios. The prototypic implementation of semantic translations between agents and the Web services domain is provided based on which it can be concluded that the successful translation between agent and Web services content language is possible without losing the information required to discover and utilize the services. The important aspect that is worth mentioning is that the formal specification and validation of the overall system is done using  $\pi$ -ADL.NET to understand the architectural details and have a technology neutral solution. The middleware based approach comprises of a number of modules, but from an implementation specific point only those modules have been implemented that are required to perform the service discovery and translation. Modules, like negotiation component, are presented in detail in the form of  $\pi$ -ADL specifications for further development and deployment in the future. The middleware based approach can be integrated in the future with Grid technologies, since the Grid community is focusing on service orienting the Grid services and can use the agent capabilities for service discovery and translation. However, the ontological translations will provide the necessary information in order to overcome the semantic gap.

# Chapter 6

## Conclusion

---

This chapter summarizes and concludes the whole thesis by presenting the overview of issues and approaches that have been explained throughout this thesis. Section 6.1.1 revisits the research questions and the need for developing a system that could provide the solution to identified problems in this thesis. Section 6.1.2 highlights the contribution and significance of the proposed work and how it can contribute in resolving the issues that were previously unaddressed by other technologies. Section 6.2 provides the overview of future directions that emerged during the thesis; although some of them are relevant, they are beyond the scope of this research. The future directions are then divided into short-term objectives, medium-term objectives and long-term objectives. In the end, section 6.3 presents the concluding remarks of the thesis and how this work is relevant to achieving the vision of an Autonomous Semantic Grid.

### 6.1 Conclusions

The research objective in this thesis was to propose a new paradigm for interactions among Web services and software agents for communication. The previously provided solutions between agents and Web services (such as AgentWeb Gateway and Web Service Integration Gateway Service (Greenwood et al., 2004) are insufficient because it is on a structural and syntactic level. Semantic interoperability has not been addressed in these solutions. The aim of this thesis is to overcome the communication issues between agents and web services as a single solution preserving the semantics of the content exchanged between these entities. The key concern while developing the solution was to provide a middleware based approach for enabling communication without violating the standards for these technologies. To address these issues, architecture has been devised with various components at an abstract level that work together to achieve the ultimate goal. To enable the semantic communication between these two technologies, ontologies have been used for defining semantic annotations, and mappings have been devised between the two languages that are adopted by the agent community and the Web services community.

A prototypic implementation has been done to validate the ontological translations between the agents and the Web services. Moreover, the steps involved in the negotiation process between agents and Web services have been highlighted. Providing a complete implementation of such a large scale system is a complex task, therefore  $\pi$ -ADL, a formal language, has been used in order to provide the specification details of the architecture so that it can be implemented as a complete solution in the future.

### 6.1.1. Road map towards addressing the Research Questions

The research hypothesis addressed in this thesis is:

“It is possible to develop system architecture for interoperability between agents with the Web services without changing the existing specifications of both the governing bodies i.e. FIPA and W3C.”

The rationale of research is the belief that software agents can play a significant role in automation of managing Web services and will thus reduce complexity. Therefore, the aim is to integrate both technologies, i.e. agents and Web services. Since both of these technologies are based on distinct standards and have specified their own vocabulary, there are chances of misinterpretation while one technology wants to retrieve exchanged information with the other technology.

The following research questions have been addressed in this thesis to explore the hypothesis:

- Is it possible to integrate agents and Web services without modifications to their standards?

A middleware based approach has been proposed in the thesis which acts as mediator between the agents and Web services. The key aspect of the proposed solution is that it will take into account the existing standards of agents and Web services. The stakeholders can benefit from the proposed solution by introducing this middleware with their previously deployed technological solutions.

- What will be the role of ontologies in the integration of these technologies?

In order to have a meaningful conversation between two entities it is important that they share the same understanding of the knowledge. Ontologies are, therefore, an integral part of the proposed solution as they provide the means to solve the semantic interoperability issues, when agents and Web services want to communicate with one another. Another important point that is worth mentioning is that although a lot of efforts have been made in standardizing ontologies, all of these efforts were useless since there is no standardized ontology, nor complete semantic description of the web sources. However, in this thesis the ontological translations have been provided as a solution to solve semantic differences and use the agent capabilities to communicate with the Web services.



These questions have been further divided into sub-questions for complete understanding:

- How will the agents communicate with the Web services?

This thesis answers the question of agents and Web services communication by defining different architectural components, each having its own specialized functionality to enable successful communication between agents and Web services.

- Can the agents and Web services communicate without standards violation?

The middleware based approach ensures that the solution takes the standards that are specified by FIPA and W3C into account. The services that have been deployed previously on agent platforms or the Web will not be modified, however the stakeholders will introduce the middleware with their existing solutions to solve the semantic interoperability issues and can, therefore, benefit from the agents' intelligence by utilizing their characteristics for service discovery, translation, and invocation.

- Who will be responsible for registering the services in agents and Web services?

The agents and Web services have their own distinct directories for maintaining service references. In the agent domain, the directory that keeps records of service descriptions is known as a Directory Facilitator, all the agents register their services in Directory Facilitator. Similarly, for the Web Services, the service references are stored in the UDDI. The middleware introduces an agent for every service request, which is responsible for determining the actions to be taken for registering and discovering the services in the Directory Facilitator or UDDI.

- How will FIPA Semantic language (FIPA SL) and Web Ontology language (OWL) translation take place and what is the role of ontologies?

FIPA Semantic language and Web Ontology language are the languages proposed by FIPA and W3C as content description languages and defining the ontologies in agents and the Web services domain, respectively. In order to have meaningful communication and interactions between agents and Web services, it is important that their semantic representations of domain knowledge are interoperable with one another. These semantic representations

are defined in terms of ontologies; therefore, semantic mappings have been devised in this thesis to overcome the heterogeneity issues between FIPA SL and OWL ontologies. From an architectural point of view, two independent components have been introduced to perform this translation process based on the mappings devised between the FIPA SL and OWL and the type of translation required in a particular scenario.

- How will the delegation of work take place when a query is posed and what are the steps involved from user request to service retrieval?

In order to answer this question, the behavioural aspect of the architecture, which defines the various steps involved, starting from the service request generation to its discovery, translation and invocation, is modelled. Each component is explained based on its functionality, inputs it receives, and type of results generated etc.

- How will the negotiation take place between the agents and Web services once a particular service has been invoked?

The negotiation module has been explicitly introduced as a part of the middleware in order to deal with the negotiation process. The negotiation process invokes when the service is discovered, and is being invoked for utility. The negotiation process is different from the other two scenarios based on the functionality. The first two scenarios discuss the type of steps involved when agents or Web services want to retrieve information about a particular service, whereas the negotiation process initiates after the service has been discovered and establishes a binding between the consumer and provider. Similarly, if additional information is required by the agents or Web services during the service binding, then the negotiation module is responsible for handling such information.

- How can we have clear and unambiguous specifications of the proposed architecture?

Designing and implementing such a system is a complex, time consuming, and cumbersome task, therefore to obtain an early executable system the architectural specifications of all the components are formulated using a formal language called  $\pi$ -ADL in order to provide clear and unambiguous specifications. These formal specifications have been validated using  $\pi$ -ADL.NET compiler and exhibit the successful communication of

agents and Web services. Moreover, these formal descriptions can provide the behavioral details of the overall system.

### 6.1.2. Contribution

Despite all the efforts in the domain of agents, all the previous efforts of Web services and the Semantic web have failed to standardize a concrete ontology or to annotate every single web resource with metadata. Taking this problem into consideration, it is believed that exploiting the agent capabilities with the current systems can lead to future applications that will support features like autonomy and self regulation. To integrate agents into the existing services, it is important to address the issues concerning interactions and communication. These issues have raised the need of developing a solution that could clearly address the interaction and communication process among agents and Web services. Therefore, in this thesis we propose the details of Ontology Gateway from structural, behavioural, and deployment aspects. The work described in this thesis is built on top of the existing protocol translator and reuses it as a module of Ontology Gateway. Table 6.1 depicts the contribution of the Ontology Gateway in state of the art solution that try to address the communication problem between agents and web services in the context of Autonomous Semantic Grid.

<b>Solutions</b>	<b>FIPA Compliant</b>	<b>W3C Compliant</b>	<b>Protocol Translation</b>	<b>Ontological Translation</b>	<b>Negotiation Support</b>
<b>WSIGS</b>	Yes	Partial support	Partial Support	No	Partial Support
<b>WS2JADE</b>	Yes	No	Yes	No	No
<b>OWL-P</b>	No	Yes	No	No	Yes
<b>MOS</b>	Yes	Yes	No	Yes	No
<b>Ontology Gateway</b>	Yes	Yes	Yes	Yes	Yes

**Table 6.1 Contribution to state of art**

Ontology Gateway enables flexible and autonomous interaction between agents and Web services. Moreover, the translation process of Web Ontology Language and FIPA

Semantic Language and vice versa have been explained without semantic loss, and how the various components of Ontology Gateway are integrated with one another to achieve successful interactions and communication. The use of ontological translations between FIPA Semantic Language and Web Ontology Language supports the interaction and communication process by resolving the differences that are faced during agent and Web services communication. The middleware based approach ensures that the standards are not violated and the existing applications can adapt to this approach. The thesis overcomes the issues raised during agent and Web services communication and clearly presents the architectural components, ontological mappings, and the behaviour of the overall system. To provide unambiguous specifications of the overall system behaviour of the architectural components was formulated using  $\pi$ -ADL and validated by executing them in  $\pi$ -ADL.NET.

The summary of contribution in this thesis is presented as follows:

- Specifications of Ontology Gateway to enable agents and Web services communication and interactions.
- Ability to cope with two distinct technologies each having its own standard, i.e. FIPA and W3C. (Standards are not violated during the communication process.)
- Detailed ontological translations that will support the communication process.
- Support for the negotiation process by creating runtime agents.
- Clear and unambiguous specification of the system using a formal language that will facilitate the system implementation and will provide an early view of the implementable solution.

## 6.2 Future Work

Figure 6.1 presents the various aspects which we will consider in future work with further research possibilities in subsequent development of the Ontology Gateway. The work presented here highlights the improvements and enhancement categorized into short-term, mid-term, and long-term objectives, taking into consideration the current state of research proposed in this thesis.

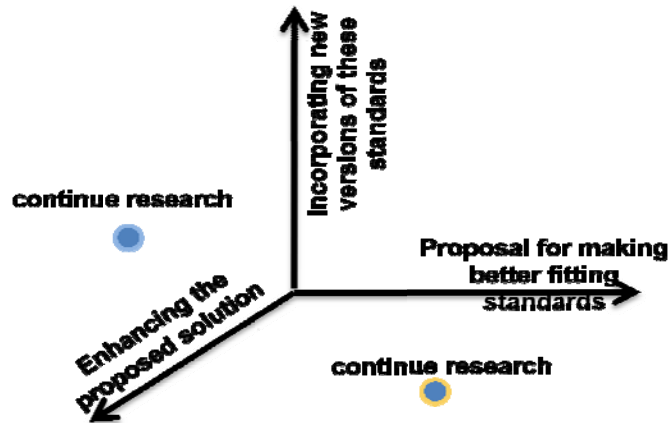


Figure 6.1: Future research directions

### 6.2.1 Short-term objectives

- *Implementation of the modules:*

The thesis highlights the modules and their specifications that are involved in having successful communication between the agents and the Web services, but due to the nature of the technologies involved, the complete implementation of such a system is a cumbersome and time consuming task and requires people of various domains to come together in order to completely implement the system, therefore  $\pi$ -ADL specifications of the system are provided that can lead to the implementation of the system.

- *Applications and Case Studies:*

The potential working of the Ontology Gateway can be validated by using more case studies and developing applications that integrate the agent capabilities and the strength of rich data sources present on the Web. Two case studies have been considered as the principle candidates based on two perspectives that is Business to Business and Business to Consumer. However, a number of other applications could be considered for experimentation with Ontology Gateway.

## 6.2.2 Mid-term objectives

- *Scalability and Distribution:*

The partial implementation of the Ontology Gateway provided in this thesis does not keep into account the scalability issue. One of the mid-term objectives is the deployment of the Ontology Gateway at a large scale. One other important aspect is to deal with distribution when Ontology Gateway is integrated into the Multi-agent systems running on a mesh of machines and thus giving complete autonomy and how it will affect the communication process to be carried between the agents and the Web Services. What will the tradeoffs be and how will this integration affect the performance of the overall system?

## 6.2.3 Long-term objectives

- *Integrating Ontology Gateway with Grid Middleware*

One of the long term objectives is the integration of the Ontology Gateway with Grid Middleware's for example Globus Toolkit. This integration will enable service descriptions and their discovery through the use of ontologies. Moreover, the flexible negotiation mechanisms from agent's domain can be applied to the current grids by using Ontology Gateway. This will facilitate the process of service description discovery and negotiation for resources that exist in Grid systems.

- *Ontology Management:*

The assumption in the current system regarding ontologies is that the participating data sources should have a consensus on the structure of the information presented in the form of ontologies, i.e. all the concepts and their relationships have to be determined before the communication takes place. Whereas, the Web is an enormous collection of data from very disparate sources, therefore one of the long term objectives is the integration of standardized ontologies from various sources and its utilization in a way so that useful hierarchies can be extracted and there should be no redundancy. Similarly, the versioning control of ontologies and the dynamic changes of the ontologies are other aspects that are still open issues.

- ***Decision engine:***

For now, we assume that a third party decision engine is required in order to select the most optimal Web service, whereas one of the long term objectives is the development of a decision engine which will be integrated as a part of the Ontology Gateway and will utilize some artificial intelligence techniques, like Bayesian classification, for improved result retrieval, and will thus reduce the communication overhead required to communicate.

- ***Additional Features:***

Moreover, some other additional features that can be added to Ontology Gateway that can enhance its potential capability are discussed as follows:

- Facilitating interoperability between FIPA and Web Services Modelling Ontologies respectively.
- Conversion of FIPA ACL query into SPARQL (SPARQL, 2007) query for execution on the Web sources.
- Support for multiple content languages.

### **6.3. Concluding Remarks**

The thesis clearly states the issues that are raised during the interoperability and Web services. To overcome these issues, a middleware based solution “Ontology Gateway” is proposed that facilitates service discovery, translation, and invocation. The ontologies play a vital role in the interaction and communication process by overcoming the semantic differences that hinder the meaningful communication between agents and Web services. The middleware based approach ensures that the standards are not violated while the agents and Web services communicate with one another. The solution provided in this thesis aims to provide a successful communication and interaction mechanism. The proposed solution can be integrated with the existing systems and can overcome the interoperability issues. Since the standards considered in this thesis are only FIPA and W3C therefore the stake holder should be compliant to these standards which can also be considered as a limitation of the system. The current implementation of the system is done on a limited scale; therefore, issues like scalability and distribution are open issues and can be considered in the future. Finally, all the efforts presented in this thesis are aimed at the realization of the Autonomous Semantic Grid vision.







## References:

- (Agentcities, 2003) Integrating Web services into agentcities, November, 2003.  
<http://www.mtakpa.hu/kpa/download/1174907.pdf>
- (Ahmad et al., 2003) H. F. Ahmad, H. Suguri, K. Iqbal, N. Baqir, A. Ali “Integration of Agents with Web Service and Grid Computing Environment”, 9th Assurance Symposium, Tokyo, Japan, 2003.
- (Ahmad et al., 2004) H. F. Ahmad, A. Ali, H. Suguri, Z. Abbas , M. Rehman, “Decentralized Multi Agent System: Basic Thoughts”, 11th Assurance System Symposium, Miyagi University, Sendai, Japan, 2004.
- (Baader et al.,2003) F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, eds. The Description Logic Handbook. Cambridge University Press, 2003.
- (Baresi et al.,2006) L. Baresi, E.Nitto, C. Ghezzi “Toward Open-World Software: Issues and Challenges”, Special Issue Introduction: The IEEE Computer Society's 60th Anniversary," Computer, vol. 39, 2006.
- (Bichler et al., 2003) M. Bichler, G. Kersten, S. Strecker , “Towards a Structured Design of Electronic Negotiations”, Group Decision and Negotiation, 2003.
- (Condor, 2007) Condor, 2007.  
<http://www.cs.wisc.edu/condor/>
- (CNP, 2002) FIPA Contract Net Interaction Protocol, December, 2002.  
<http://www.fipa.org/specs/fipa00029/SC00029H.pdf>
- (DAML, 2001) DAML Joint Committee, March, 2001.  
<http://www.daml.org/2001/03/daml+oil-index.html>
- (DAML-S, 2002) DAML-S, October, 2002.  
<http://www.ai.sri.com/daml/services/daml-s/0.7/>
- (Dean et al, 2004)M. Dean and G. Schreiber, “OWL Web Ontology Language Reference”, W3C Recommendation, 2004.
- (Fensel, 2003)D. Fensel, “Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce”, 2nd edition. Springer-Verlag, Berlin, 2003.
- (Ferber, 1999)J. Ferber. Multi Agent Systems, Addeson Wesley, Reading MA, 1999.
- (FIPA, 2002) FIPA Abstract Architecture Specification, December, 2002.

<http://www.fipa.org/specs/fipa00001/SC00001L.html>

(FIPA, 2003) Foundation for Intelligent Physical Agents, March, 2003.

<http://www.fipa.org/>

(Foster et al., 1999) I. Foster and C. Kesselman (editors), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, USA, 1999.

(Foster, 2001) I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of Supercomputer Applications, 2001.

(Foster et al., 2004) I. Foster, N. Jennings, C. Kesselman, “Brain meets brawn: Why Grid and agents need each other”, Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems, New York, USA, 2004.

(Ghafoor et al., 2004) A. Ghafoor, M. Rehman, Z. Abbas, H. F. Ahmad, A. Ali, “SAGE: Next Generation Multi-Agent System”, Proc. of IEEE International Conference on Parallel and Distributed Processing Techniques and Applications, 2004.

(Greenwood et al., 2004) D. Greenwood, M. Calisti, S. Zurich., “Engineering web services – Agent Integration”, IEEE International Conference on Systems, Man and Cybernetics, 2004.

(Gruber, 2004) T. R. Gruber “Toward principles for the design of ontologies used for knowledge sharing”, International Workshop on Formal Ontology, Padova, Italy, 1993.

(Heiler, 1995) S. Heiler, Semantic Interoperability, GTE Laboratories Incorporated, Waltham, Massachusetts, ACM Computing Surveys, Vol. 27, No 2, June 1995.

(IBM, 2007) Web Service Resource Transfer, 2007.

<http://www.ibm.com/developerworks/grid/library/gr-wsrfwsrt/index.html>

(IEEE, 2004) IEEE Spectrum, July, 2004.

<http://www.spectrum.ieee.org/jul04/3922>

(JADE, 2006) Java Agent Development Environment, April, 2006.

<http://jade.tilab.com/>

(Jena, 2007) Jena, 2007.

<http://jena.sourceforge.net/>

(Keahey, 2006) K. Keahey, J. S. Chase, I. Foster: Virtual playgrounds: managing virtual resources in the grid, IEEE International Parallel & Distributed Processing Symposium, 2006.

- (Khalid et al, 2007) N. Khalid, M. Pasha, S. Rehman: "Ontology Services between Agents and OWL Based Web Services" the 3rd International Conference on Semantics, Knowledge and Grid. IEEE Computer Society, 2007.
- (Khan et al, 2005) Z. A. Khan, H. F. Ahmad, A. Ali, H. Suguri, "Decentralized Architecture for Fault Tolerant Multi Agent System", International Symposium on Autonomous Decentralized Systems, 2005.
- (Lassila, 2005) Semantic Web Blog, June, 2005.  
<http://www.lassila.org/blog/archive/2005/06/>
- (Lee, 2001) T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web", Scientific American, 2001.
- (Laukkanen et al., 2003) M. Laukkanen, H. Helin, Composing workflows of Semantic web services. In Proc. of the 1st International Workshop on Web Services and Agent Based Engineering, Sydney, Australia, 2003.
- (Martin , 2007) D. Martin, Semantic Web Services, IEEE Intelligent Systems, 2007.
- (Martin, 2004) D. Martin, M. Burstein, O. Lassila, Describing Web Services using OWL-S and WSDL, 2004.  
<http://www.daml.org/services/owl-s/1.1/owl-s-wsdl.html>
- (Mcilraith et al., 2001) S. Mcilraith , T. Son, H. Zeng , Semantic Web Service, IEEE Intelligent Systems, 2001.
- (Maherzi, 1997) L. Maherzi, "World Communication Report: The media and the challenge of new technologies", UNESCO Publishing, 1997.
- (Milner, 1993) R. Milner, "The Polyadic  $\pi$ -Calculus: A Tutorial," Logic and Algebra of Specification, Springer-Verlag, 1993.
- (Mori, 1993) K. Mori, "Autonomous Decentralized Systems: Concept, Data Field Architecture and Future Trends", Proc. of the first International Symposium on Autonomous Decentralized Systems, IEEE, Kawasaki, Japan, 1993.
- (Moore, 2005) Moores's Law  
<http://www.intel.com/technology/mooreslaw/index.htm>
- (myGrid, 2007) myGrid Project, May, 2007.  
[www.ebi.ac.uk/mygrid/](http://www.ebi.ac.uk/mygrid/)
- (Nassuni et al, 2003) P. Nassuni and S. Katia , "Autonomous Semantic Web Services", IEEE Internet

Computing, 2003.

(Negri et al., 2006) A. Negri, A. Poggi, M Tomaiuolo, “Agents for e-Business applications”, Proceedings of the fifth international joint conference on Autonomous Agents and Multi-Agent System, Japan, 2006.

(Nguyen, 2005) X. T. Nguyen. Demonstration of WS2JADE. In Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-agent Systems, Utrecht, The Netherlands, 2005.

(OGSA, 2003) Open Grid Service Architecture, June, 2003.

<http://www.globus.org/ogsa/>

(Oquendo, 2004) F. Oquendo, “ $\pi$ -ADL: An Architecture Description Language based on the Higher Order Typed  $\pi$ -Calculus for Specifying Dynamic and Mobile Software Architectures,” ACM Software Engineering Notes, 2004.

(Oquendo, 2005) F. Oquendo, “Tutorial on ArchWare ADL – Version 2 ( $\pi$ -ADL Tutorial),” ArchWare European RTD Project IST-2001-32360, 2005.

(OWL-S, 2003) OWL-S, October, 2003 .

[www.daml.org/services/owl-s/OWL-S-SWSWPC2004-CameraReady.doc](http://www.daml.org/services/owl-s/OWL-S-SWSWPC2004-CameraReady.doc)

(Paolucci et al, 2003) M. Paolucci, K. Sycara, Autonomous Web Services, IEEE, 2003.

(Papazoglou et al, 2006) M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann,” Service-Oriented Computing:State of the Art and Research Challenges”, IEEE Computer Society, 2006.

(Park, 2004) Park, J., Ram S., “Information Systems Interoperability: What lies beneath”. ACM Transactions on Information systems, Vol. 22, No. 4, 2004.

(Pasha et al, 2006) M. Pasha, S. Rehman, F. Ahmad, A. Ali, H. Suguri, “Middleware between OWL and FIPA Ontologies in the Semantic Grid Environment”, The 2006 International Conference on Semantic Web and Web Services, USA, 2006.

(Pasha et al, 2006) M. Pasha, F. Ahmad, H. Suguri, A. Ali, “Ontological Translations in Semantic Grid”, 4th International workshop on Multi-Agent Systems and Semantic Grid, Pakistan, 2006.

(Qayyum, 2006) Z. Qayyum and F. Oquendo, “ $\pi$ -ADL Visual Notation and its Application to Formally Modeling the High Level Architecture,” 19<sup>th</sup> International Conference on Software and Systems Engineering and their Applications, Paris, France, 2006.

- (Qayyum , 2008) Z. Qayyum. "π.NET: A High Level Architecture Description Language Compiler for the .NET Platform". 7<sup>th</sup> WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems, 2008.
- (Russell, 1995) S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 1995.
- (Sabih et al, 2007) R. Sabih, M. Pasha, F. Ahmad, A. Ali, H. Suguri, "Ontology Gateway: Enabling Interoperability between FIPA compliant Agents and OWL Web Services", 9th International Conference on Enterprise Information Systems, Funchal-Madeira/Portugal, 2007.
- (Sangiorgi, 1992) D. Sangiorgi, "Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms". PhD Thesis, University of Edinburgh, 1992.
- (SAWSDL, 2007) Semantic Annotations for WSDL, January, 2007.  
<http://www.w3.org/2002/ws/sawSDL/>
- (SG, 2005) Semantic Grid, May, 2005.  
<http://www.semanticgrid.org/brief/>
- (SI, 2005) Semantic Interoperability, February, 2005.  
<http://colab.cim3.net/file/work/SICoP/WhitePaper/SICoP.WhitePaper.Module1.v5.4.kf.021605.doc>
- (Semantic Grid,2001) Semanti Grid, November, 2001.  
<http://www.semanticgrid.org/vision.html>
- (Shafiq et al., 2005) O. Shafiq , F. Ahmad ,H. Suguri , A. Ali , "Autonomous Semantic Grid: Principles of Autonomous Decentralized Systems for Grid Computing" IEICE/IEEE TRANS., VOL. E85-A/B/C/D, 2005.
- (Shafiq et al., 2006) O. Shafiq , Y. Ding , D. Fensel, "Bridging Multi Agent Systems and Web Services: towards interoperability between Software Agents and Semantic Web Services," 10th IEEE International Enterprise Distributed Object Computing Conference, 2006.
- (Sirin et al., 2003) E. Sirin, J. Hendler ,B. Parsia, Semiautomatic composition of web services using semantic description. Web Services: Modeling, Architecture and Infrastructure Workshop in Conjunction with ICEIS, 2003.
- (Singh et al., 2005) M. P. Singh and M. N. Huhns, Service-Oriented Computing, Wiley, 2005.
- (Smith et al., 2002) M. K. Smith, D. McGuinness, R. Volz, and C. Welty. Web Ontology Language (OWL) Guide version 1.0, 2002.
- (SOAP, 2007) Simple Object Access Protocol, March, 2007.  
<http://www.w3.org/TR/soap/>

(Spanoudakis et al.,2007) N. Spanoudakis, P. Moraitis, “An Ambient Intelligence Application Integrating Agent and Service-Oriented Technologies”, 27th SGAI International Conference on Artificial Intelligence (AI2007), UK, 2007.

(SPARQL, 2007) SPARQL, January, 2007  
<http://www.w3.org/TR/rdf-sparql-query/>

(Suguri et al., 2004) H. Suguri, H. F. Ahmad, M. O. Shafiq and A. Ali, “Agent Web Gateway - Enabling Service Discovery and Communication among Software Agents and Web Services”, Proc. of Third Joint Agent Workshop and Symposium, Japan, 2004.

(Swartout et al., 1996) B. Swartout, R. Patil, K. Knight and T. Russ, “Toward Distributed Use of Large-Scale Ontologies”, Proc. of the 10th Knowledge Acquisition Workshop,1996.

(SWSF,2005) Semantic Web Services Framework, September, 2005.  
<http://www.w3.org/Submission/SWSF/>

(Sycara et al., 2004) K. Sycara, M. Paolucci, J. Soundary, N. Srinivasan, Dynamic discovery and coordination of agent-based semantic web services. IEEE Internet Computing, 2004.

(Tuecke et al., 2003) S. Tuecke, K. Czajkowski, I. Foster,J. Frey, S.Graham, C. Kesselman, ,T. Maquire,T. Sandholm, D.Snelling, P. Vanderbilt "Open Grid Services Infrastructure (OGSI) Version 1.0", 2003.

(UDDI, 2005) Universal Description, Discovery and Integration, February, 2005.  
[http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=uddi-spec](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec)

(Walton et al., 2004) C. Walton and A. Barker, "An Agent-based e-Science Experiment Builder", 1st International Workshop on Semantic Intelligent Middleware for the Web and the Grid, Valencia, Spain, 2004.

(W3C, 2001) W3C, The Semantic Web, May, 2001.  
<http://www.w3.org/2001/sw/>

(Wooldridge, 2002) M. Wooldridge, “An Introduction to Multi-Agent Systems”, John Wiley & Sons, 2002.

(WS, 2002) Web Services Architecture Working Group, October, 2002.  
<http://www.w3.org/2002/ws/arch/>

(WSCL,2002) Web Services Conversation Language, March, 2002.  
<http://www.w3.org/TR/wscl10/>

(WSDL, 2001) Web Services Description Language, March, 2001.

<http://www.w3.org/TR/wsdl>

(WSMO, 2006) Web service Modeling Ontology, October, 2006.

<http://www.wsmo.org/TR/d2/v1.3/>

(WSRF, 2004) Web Service Resource Framework, March, 2004.

<http://www.globus.org/wsr/faq.asp>, March 2004

(WSRT, 2006) Web Service Resource Transfer, August, 2006.

<http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-rt/ws-rt-spec.pdf>, August 2006.



## APPENDIX A

### A.1 People Ontology Defined In FIPA

```
import acl.ontology.BasicOntology;
import acl.ontology.Ontology;
import acl.schema.AgentActionSchema;
import acl.schema.AggregateSchema;
import acl.schema.ConceptSchema;
import acl.schema.PredicateSchema;
import acl.schema.PrimitiveSchema;

public class PeopleOntology extends Ontology {
    //A symbolic constant, containing the name of this ontology.
    public static final String ONTOLOGY_NAME = "PEOPLE_ONTOLOGY";

    // Concepts
    public static final String PERSON = "PERSON";
    public static final String MAN = "MAN";
    public static final String WOMAN = "WOMAN";
    public static final String ADDRESS = "ADDRESS";

    // Slots
    public static final String NAME = "NAME";
    public static final String STREET = "STREET";
    public static final String NUMBER = "NUMBER";
    public static final String CITY = "CITY";

    // Predicates
    public static final String FATHER_OF = "FATHER_OF";
    public static final String MOTHER_OF = "MOTHER_OF";

    // Roles in predicates
    public static final String FATHER = "FATHER";
    public static final String MOTHER = "MOTHER";
    public static final String CHILDREN = "CHILDREN";

    // Actions
    public static final String MARRY = "MARRY";

    // Arguments in actions
    public static final String HUSBAND = "HUSBAND";
    public static final String WIFE = "WIFE";

    //private static PeopleOntology theInstance = new PeopleOntology(ACLOntology.getInstance());
    private static PeopleOntology theInstance = new PeopleOntology();

    public static PeopleOntology getInstance() {
        return theInstance;
    }

    public PeopleOntology() {
        super(ONTOLOGY_NAME, BasicOntology.getInstance());
    }

    try {
```

```

PrimitiveSchema stringSchema = (PrimitiveSchema)getSchema(BasicOntology.STRING);
PrimitiveSchema integerSchema = (PrimitiveSchema)getSchema(BasicOntology.INTEGER);

ConceptSchema addressSchema = new ConceptSchema(ADDRESS);
addressSchema.add(STREET, stringSchema);
addressSchema.add(NUMBER, integerSchema);
addressSchema.add(CITY, stringSchema);

ConceptSchema personSchema = new ConceptSchema(PERSON);
personSchema.add(NAME, stringSchema);
//personSchema.add(ADDRESS, addressSchema);

ConceptSchema manSchema = new ConceptSchema(MAN);
manSchema.addSuperSchema(personSchema);

ConceptSchema womanSchema = new ConceptSchema(WOMAN);
womanSchema.addSuperSchema(personSchema);

ConceptSchema motherSchema = new ConceptSchema(MOTHER);
motherSchema.addSuperSchema(womanSchema );

ConceptSchema fatherSchema = new ConceptSchema(FATHER);
fatherSchema.addSuperSchema(manSchema );

ConceptSchema husbandSchema = new ConceptSchema(HUSBAND);
husbandSchema.addSuperSchema(manSchema );

ConceptSchema wifeSchema = new ConceptSchema(WIFE);
wifeSchema.addSuperSchema(womanSchema );

add(personSchema);
add(manSchema);
add(womanSchema);
add(addressSchema);

AggregateSchema childrenSchema = new AggregateSchema(BasicOntology.SEQUENCE);
childrenSchema.addSuperSchema(personSchema);

PredicateSchema fatherOfSchema = new PredicateSchema(FATHER_OF);
fatherOfSchema.add(FATHER, fatherSchema);
fatherOfSchema.add(CHILDREN, childrenSchema);

PredicateSchema motherOfSchema = new PredicateSchema(MOTHER_OF);
motherOfSchema.add(MOTHER, motherSchema);
motherOfSchema.add(CHILDREN, childrenSchema);

add(fatherOfSchema);
add(motherOfSchema);

AgentActionSchema marrySchema = new AgentActionSchema(MARRY);
marrySchema.add(HUSBAND, husbandSchema);
marrySchema.add(WIFE, wifeSchema);

add(marrySchema);

```

```

    } catch(Exception oe) { oe.printStackTrace(); }

}

}

```

## A.2 People Ontology Defined In OWL

```

<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.owl-ontologies.com/PEOPLE_ONTOLOGY.owl">
  <owl:Ontology rdf:about=""/>
    <rdfs:comment xml:lang="en">This is an Automatically Generated OWL ontology. Generated Through
OntologyGateway</rdfs:comment>
  </owl:Ontology>
  <owl:Class rdf:ID="PEOPLE_ONTOLOGY">
    <owl:Class rdf:ID="address">
      <rdfs:subClassOf rdf:resource="PEOPLE_ONTOLOGY"/>
    </owl:Class>
    <owl:Class rdf:ID="person">
      <rdfs:subClassOf rdf:resource="PEOPLE_ONTOLOGY"/>
    </owl:Class>
    <owl:Class rdf:ID="man">
      <rdfs:subClassOf rdf:resource="#person"/>
    </owl:Class>
    <owl:Class rdf:ID="woman">
      <rdfs:subClassOf rdf:resource="#person"/>
    </owl:Class>
    <owl:Class rdf:ID="mother">
      <rdfs:subClassOf rdf:resource="#woman"/>
    </owl:Class>
    <owl:Class rdf:ID="father">
      <rdfs:subClassOf rdf:resource="#man"/>
    </owl:Class>
    <owl:Class rdf:ID="husband">
      <rdfs:subClassOf rdf:resource="#man"/>
    </owl:Class>
    <owl:Class rdf:ID="wife">
      <rdfs:subClassOf rdf:resource="#woman"/>
    </owl:Class>
    <owl:Class rdf:ID="children">
      <rdfs:subClassOf rdf:resource="#person"/>
    </owl:Class>
    <owl:ObjectProperty rdf:ID="fatherOf">
      <rdfs:domain rdf:resource="#father"/>
      <rdfs:range rdf:resource="#children"/>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#fatherOf"/>
          </owl:onProperty>
          <owl:someValuesFrom rdf:resource="#father"/>
        </owl:Restriction>
      </rdfs:subClassOf>
    </owl:ObjectProperty>

```

```

</rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#fatherOf"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#children"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="motherOf">
  <rdfs:domain rdf:resource="#mother"/>
  <rdfs:range rdf:resource="#children"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#motherOf"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#mother"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#motherOf"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#children"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#motherOf"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#children"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  </owl:ObjectProperty>
<owl:Class rdf:ID="action">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#husband"/>
    <owl:Class rdf:about="#wife"/>
  </owl:unionOf>
</owl:Class>
<owl:Class rdf:ID="Aggregate"/>
  <owl:equivalentClass>
    <owl:Class rdf:resource="#children"/>
  </owl:equivalentClass>
</owl:Class>
<owl:ObjectProperty rdf:ID="street">
  <rdfs:domain rdf:resource="#address"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="number">
  <rdfs:domain rdf:resource="#address"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="city">
  <rdfs:domain rdf:resource="#address"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="name">

```

```
<rdfs:domain rdf:resource="#person"/>
</owl:ObjectProperty>
</owl:Class>
</rdf:RDF>
```

## APPENDIX B

### B.1 University Ontology Defined In OWL

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns="http://www.owl-ontologies.com/unnamed.owl#"
  xml:base="http://www.owl-ontologies.com/unnamed.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="university"/>
  <owl:Class rdf:ID="courses">
    <rdfs:subClassOf rdf:resource="#university"/>
  </owl:Class>
  <owl:Class rdf:ID="BIT">
    <owl:disjointWith>
      <owl:Class rdf:ID="BEE"/>
    </owl:disjointWith>
    <rdfs:subClassOf rdf:resource="#courses"/>
    <owl:disjointWith>
      <owl:Class rdf:ID="BICSE"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:ID="BEE2">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#BEE"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:ID="BEE1"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:ID="BIT1">
    <owl:disjointWith>
      <owl:Class rdf:ID="BIT4"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="BIT5"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="BIT2"/>
    </owl:disjointWith>
    <rdfs:subClassOf rdf:resource="#BIT"/>
    <owl:disjointWith>
      <owl:Class rdf:ID="BIT3"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:ID="BICSE2">
    <owl:disjointWith>
      <owl:Class rdf:ID="BICSE1"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="BISCE3"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#BICSE"/>
    </rdfs:subClassOf>
  </owl:Class>
```

```

<owl:Class rdf:about="#BIT5">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >bit batch no 5 </rdfs:comment>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >bit5</rdfs:label>
  <owl:disjointWith rdf:resource="#BIT1"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#BIT2"/>
  </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#BIT"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#BIT3"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#BIT4"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:about="#BIT2">
  <owl:disjointWith>
    <owl:Class rdf:about="#BIT4"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#BIT5"/>
  <owl:disjointWith rdf:resource="#BIT1"/>
  <rdfs:subClassOf rdf:resource="#BIT"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#BIT3"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:about="#BEE1">
  <owl:disjointWith rdf:resource="#BEE2"/>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#BEE"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#BICSE1">
  <owl:disjointWith>
    <owl:Class rdf:about="#BISCE3"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#BICSE"/>
  </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#BICSE2"/>
</owl:Class>
<owl:Class rdf:about="#BEE">
  <owl:disjointWith rdf:resource="#BIT"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#BICSE"/>
  </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#courses"/>
</owl:Class>
<owl:Class rdf:about="#BIT3">
  <owl:disjointWith rdf:resource="#BIT1"/>
  <owl:disjointWith rdf:resource="#BIT2"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#BIT4"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#BIT5"/>
  <rdfs:subClassOf rdf:resource="#BIT"/>
</owl:Class>
<owl:Class rdf:about="#BISCE3">
  <rdfs:subClassOf>

```

```

    <owl:Class rdf:about="#BICSE"/>
  </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#BICSE1"/>
  <owl:disjointWith rdf:resource="#BICSE2"/>
</owl:Class>
<owl:Class rdf:about="#BIT4">
  <owl:disjointWith rdf:resource="#BIT1"/>
  <owl:disjointWith rdf:resource="#BIT2"/>
  <owl:disjointWith rdf:resource="#BIT5"/>
  <owl:disjointWith rdf:resource="#BIT3"/>
  <rdfs:subClassOf rdf:resource="#BIT"/>
</owl:Class>
<owl:Class rdf:about="#BICSE">
  <rdfs:subClassOf rdf:resource="#courses"/>
  <owl:disjointWith rdf:resource="#BEE"/>
  <owl:disjointWith rdf:resource="#BIT"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="noOfStudents">
  <rdfs:domain rdf:resource="#BIT"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasStudents">
  <rdfs:domain rdf:resource="#BIT"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasTeacher">
  <rdfs:domain rdf:resource="#courses"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="ObjectProperty_20">
  <rdfs:domain rdf:resource="#BIT"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="DatatypeProperty_17"/>
</rdf:RDF>

```

## B.2 University Ontology Defined In FIPA

```

import acl.*;
import acl.ontology.management.*;
import acl.sl.codec.*;
import ams.*;
import df.*;
import acl.schema.*;
import acl.ontology.BasicOntology;
import acl.ontology.Ontology;
import acl.schema.AggregateSchema;
import acl.schema.ConceptSchema;
import acl.schema.PredicateSchema;
import acl.schema.PrimitiveSchema;
public class FIPAOntology extends Ontology{
public static final String ONTOLOGY_NAME="courses";// NAMES OF CONCEPTS
    public static final String BICSE1 = "BICSE1";
    public static final String BICSE2 = "BICSE2";
    public static final String BIT5 = "BIT5";
    public static final String BIT3 = "BIT3";
    public static final String BIT1 = "BIT1";
    public static final String BIT4 = "BIT4";
    public static final String BIT2 = "BIT2";
    public static final String BEE1 = "BEE1";
    public static final String BEE2 = "BEE2";
    public static final String courses = "courses";
    public static final String BICSE = "BICSE";
    public static final String BEE = "BEE";

```



```

public static final String BIT = "BIT";

// NAMES OF ATTRIBUTES
public static final String ObjectProperty_20 = "ObjectProperty_20";
public static final String hasTeacher = "hasTeacher";
public static final String hasStudents = "hasStudents";
public static final String noOfStudents = "noOfStudents";

private static FIPAontology theInstance = new FIPAontology(FIPAontology.getInstance());
public static FIPAontology getInstance() {
    return theInstance;
}
public FIPAontology(Ontology base) {
super(ONTOLOGY_NAME, BasicOntology.getInstance());
try {
    PrimitiveSchema stringSchema = (PrimitiveSchema)getSchema(BasicOntology.STRING);
    PrimitiveSchema integerSchema = (PrimitiveSchema)getSchema(BasicOntology.INTEGER);

    ConceptSchema coursesSchema = new ConceptSchema(courses);
    ConceptSchema BICSESchema = new ConceptSchema(BICSE);
    BICSESchema.add(hasTeacher, stringSchema);

    BICSESchema.addSuperSchema(coursesSchema);
    ConceptSchema BEESchema = new ConceptSchema(BEE);
    BEESchema.add(hasTeacher, stringSchema);

    BEESchema.addSuperSchema(coursesSchema);
    ConceptSchema BITSchema = new ConceptSchema(BIT);
    BITSchema.add(hasTeacher, stringSchema);

    BITSchema.addSuperSchema(coursesSchema);
    ConceptSchema universitySchema = new ConceptSchema(university);
    ConceptSchema coursesSchema = new ConceptSchema(courses);

    coursesSchema.addSuperSchema(universitySchema);
    ConceptSchema BEESchema = new ConceptSchema(BEE);
    ConceptSchema BEE1Schema = new ConceptSchema(BEE1);

    BEE1Schema.addSuperSchema(BEESchema);
    ConceptSchema BEE2Schema = new ConceptSchema(BEE2);

    BEE2Schema.addSuperSchema(BEESchema);
    ConceptSchema BITSchema = new ConceptSchema(BIT);
    ConceptSchema BIT5Schema = new ConceptSchema(BIT5);
    BIT5Schema.add(ObjectProperty_20, stringSchema);
    BIT5Schema.add(hasStudents, stringSchema);
    BIT5Schema.add(noOfStudents, stringSchema);

    BIT5Schema.addSuperSchema(BITSchema);
    ConceptSchema BIT3Schema = new ConceptSchema(BIT3);
    BIT3Schema.add(ObjectProperty_20, stringSchema);
    BIT3Schema.add(hasStudents, stringSchema);
    BIT3Schema.add(noOfStudents, stringSchema);

    BIT3Schema.addSuperSchema(BITSchema);
    ConceptSchema BIT1Schema = new ConceptSchema(BIT1);
    BIT1Schema.add(ObjectProperty_20, stringSchema);
    BIT1Schema.add(hasStudents, stringSchema);
    BIT1Schema.add(noOfStudents, stringSchema);

    BIT1Schema.addSuperSchema(BITSchema);

```

```

ConceptSchema BIT4Schema = new ConceptSchema(BIT4);
BIT4Schema.add(ObjectProperty_20, stringSchema);
BIT4Schema.add(hasStudents, stringSchema);
BIT4Schema.add(noOfStudents, stringSchema);

BIT4Schema.addSuperSchema(BITSchema);
ConceptSchema BIT2Schema = new ConceptSchema(BIT2);
BIT2Schema.add(ObjectProperty_20, stringSchema);
BIT2Schema.add(hasStudents, stringSchema);
BIT2Schema.add(noOfStudents, stringSchema);

BIT2Schema.addSuperSchema(BITSchema);
ConceptSchema BICSESchema = new ConceptSchema(BICSE);
ConceptSchema BISCE3Schema = new ConceptSchema(BISCE3);

BISCE3Schema.addSuperSchema(BICSESchema);
ConceptSchema BICSE1Schema = new ConceptSchema(BICSE1);

BICSE1Schema.addSuperSchema(BICSESchema);
ConceptSchema BICSE2Schema = new ConceptSchema(BICSE2);

BICSE2Schema.addSuperSchema(BICSESchema);
}
catch(Exception oe) { oe.printStackTrace(); }
}
}

```

## APPENDIX C

### Subset of Ontology constructs by FIPA

**class:** is a set of entities. Each of the entities in a class is said to be an *instance of* the class. An entity can be an instance of multiple classes, which are called its *types*.

**individuals:** entities that are not classes are referred to as *individuals*. Thus, the domain of discourse consists of individuals and classes.

**domain:** this indicates the domain over which the function is defined. The arguments passed to the function must belong to the set identified by the domain.

**range:** this indicates the range to which the function maps the symbols of the domain. The result of the function is a symbol belonging to the set identified by the range.

**thing:** is the root of the class hierarchy for a knowledge base, meaning that *THING* is the superclass of every class in every knowledge base.

**frame:** this is the mandatory name of this entity that must be used to represent each instance of this class.

**equivalent:** two classes are equivalent if they share the same instances and properties.

**subclass-of:** the *subclass-of* relation for classes is defined in terms of the relation *instance-of* where a class *csub* is a subclass of class *csuper* if and only if all instances of *csub* are also instances of *csuper*.

**content-description:** specifies the constraints that are associated with the different classes and their subclasses.

**same-values:** facet specifies that a slot of a frame has the same values as other slots of that frame or as the values specified by *slot chains* starting at that frame. Each value of this facet is either a slot or a slot chain.

**value-type:** facet specifies a type restriction on the values of a slot of a frame. Each value of the *:VALUE-TYPE* facet denotes a class.

**not:** the operator specifies the negation of a particular class or an expression.

**and:** the operator specifies the conjunction of two classes or expressions. Its value is true if both the expressions have a true value.

**or:** the operator specifies the disjunction between two classes or expressions. In case of expressions its value is false if both the expressions have a false value.

**maximum-cardinality:** facet specifies the maximum number of values that may be asserted for a slot of a frame.

**minimum-cardinality:** facet specifies the minimum number of values that may be asserted for a slot of a frame.

**set-of:** expression allows the specification of an explicitly enumerated set of possible values for the slot.

**slot-value-type:** specifies the classes of which values of a slot must be an instance. Each value of the slot-value-type denotes a class.

**slot-maximum-cardinality:** specifies the maximum number of values that may be asserted for a slot for entities in the slot's domain.

**slot-minimum-cardinality:** specifies the minimum number of values for a slot for entities in the slot's domain.

**slot-same-values:** specify that a slot has the same values as either other slots or as slot chains for entities in the slot's domain.

**slot-not-same-values:** specify that a slot does not have the same values as either other slots or as slot chains for entities in the slot's domain.

**slot-subset-of-values:** specify that the values of a slot are a subset of either other slots or of slot chains for entities in the slot's domain. Each value of slot-subset-of-values is either a slot or a slot chain.

**slot-some-values:** specify a subset of the values of a slot for entities in the slot's domain. Each value of slot-same-values of a slot frame must be in the domain of the slot represented by the slot frame.

**slot-cardinality:** specifies the exact number of values that may be asserted for a slot for entities in the slot's domain. The value of slot-cardinality is a nonnegative integer.

**slot-inverse:** specifies inverse relations for a slot. Each value of slot-inverse is a slot.

## APPENDIX D

### Subset of Ontology constructs by OWL

**Class:** A class defines a group of individuals that belong together because they share some properties.

**subClassOf:** Class hierarchies may be created by making one or more statements that a class is a subclass of another class.

**Thing:** is the class of all individuals and is a superclass of all OWL classes.

**Individual:** individuals are instances of classes, and properties may be used to relate one individual to another.

**domain:** a domain of a property limits the individuals to which the property can be applied. If a property relates an individual to another individual, and the property has a class as one of its domains, then the individual must belong to the class.

**range:** The range of a property limits the individuals that the property may have as its value. If a property relates an individual to another individual, and the property has a class as its range, then the other individual must belong to the range class.

**restricton:** allows restrictions to be placed on how properties can be used by instances of a class.

**allValuesFrom:** the restriction allValuesFrom is stated on a property with respect to a class. It means that this property on this particular class has a local range restriction associated with it.

**someValuesFrom:** the restriction someValuesFrom is stated on a property with respect to a class. A particular class may have a restriction on a property that at least one value for that property is of a certain type.

**cardinality:** cardinality is provided as a convenience when it is useful to state that a property on a class has both minCardinality 0 and maxCardinality 0 or both minCardinality 1 and maxCardinality 1.

**minCardinality:** cardinality is stated on a property with respect to a particular class. If a minCardinality of 1 is stated on a property with respect to a class, then any instance of that class will be related to at least one individual by that property.

**maxCardinality:** cardinality is stated on a property with respect to a particular class. If a maxCardinality of 1 is stated on a property with respect to a class, then any instance of that class will be related to at most one individual by that property.

**equivalentClass :** two classes may be stated to be equivalent. Equivalent classes have the same instances. Equality can be used to create synonymous classes.

**subPropertyOf:** property hierarchies may be created by making one or more statements that a property is a subproperty of one or more other properties.

**equivalentProperty:** two properties may be stated to be equivalent. Equivalent properties relate one individual to the same set of other individuals. Equality may be used to create synonymous properties.

**intersectionOf:** OWL allows intersections of named classes and restrictions.

**oneOf:** classes can be described by enumeration of the individuals that make up the class. The members of the class are exactly the set of enumerated individuals; no more, no less.

**datatype:** property links individual to values.

**hasValue (property values):** a property can be required to have a certain individual as a value (also sometimes referred to as property values).

**disjointWith:** classes may be stated to be disjoint from each other. From this disjointWith statement, a reasoner can deduce an inconsistency when an individual is stated to be an instance of both.

**unionOf:** property links a class to a list of class descriptions. An owl:unionOf statement describes an anonymous class for which the class extension contains those individuals that occur in at least one of the class extensions of the class descriptions in the list.

**inverseOf:** property provides the support for specifying the direction from domain to range.

**complementOf:** property links a class to precisely one class description. An owl:complementOf statement describes a class for which the class extension contains exactly those individuals that do *not* belong to the class extension of the class description.

## APPENDIX E

### $\pi$ -ADL code for Flight reservation service example

```
Actor names behaviour
{
    qConn : Connection[view[request : String, sender : String, receiver : String,
        content : view[service_id : String, service_name : String,
            params : sequence[view[name : String, val : String]], numOfParams :
Integer],
        language : String, ontology : String, reply_with : String]];
    query : view[request : String, sender : String, receiver : String,
        content : view[service_id : String, service_name : String,
            params : sequence[view[name : String, val : String]], numOfParams :
Integer],
        language : String, ontology : String, reply_with : String];

    query::request = "Flight Information";
    query::sender = "771@cern1-7";
    query::receiver = "624@cern1-7";
    query::content = view(service_id : "1", service_name : "Flight reservation", params :
sequence (view(name : "date", val : "15/09/2008"), view(name : "time", val :
"12h00"),
        view(name : "price", val : "Rs. 40,000"),
        view(name : "source", val : "CDG"), view(name : "destination", val :
"LHR")), numOfParams : 5);

    via out send "**** Actor ****\n\n--User input--\n";
    via out send query;
    via out send "\n\n";

    compose
    {
        via out send "--Send user input to CU--\n";
        via CU send query where {qConn renames aConn};
    and
        via qConn receive query;
        via out send "**** Actor ****\n\n--Response recieved--\n";
        via out send query;
        via out send "\n\n";

        query::content::params(0)::val = "Maruf";
        query::content::params(1)::val = "Appt. 34, 7 Sqr. Montagne, Vannes";
    }
```



```

        query::content::params(2)::val = "+33-2-97639764";

        via qConn send query;
        via qConn receive query;
        via out send "**** Actor ****\n\n--";
        via out send query::reply_with;
        via out send "--\n";
    }
}

//CU = Control Unit

value CU is abstraction (cuQuery : view[request : String, sender : String, receiver : String,
    content : view[service_id : String, service_name : String,
        params : sequence[view[name : String, val : String]], numOfParams :
Integer],
    language : String, ontology : String, reply_with : String)
{
    aConn : Connection[view[request : String, sender : String, receiver : String,
        content : view[service_id : String, service_name : String,
            params : sequence[view[name : String, val : String]], numOfParams :
Integer],
        language : String, ontology : String, reply_with : String]];
    isAgentValidated : Boolean;
    vConn : Connection[Boolean];

    //validate sender, service_id and serviceName
    unobservable;

    //this is where we invoke ontology agent
    via out send "**** CU ****\n\n--Validating Agent Authenticity--\n";
    via OA send cuQuery where {vConn renames vConn};
    via vConn receive isAgentValidated;
    via out send "**** CU ****\n\n";
    if (!isAgentValidated) do
    {
        via out send "--Unrecognized Agent--\n";
        cuQuery::reply_with = "Unrecognized request";
        via aConn send cuQuery;
        done;
    }
    else do
    {
        via out send "--Forwarding service profile parameters to NM--\n";
    }
}

```

```

        via NM send cuQuery where {aConn renames cConn};
    }
} //end CU

//OA = Ontology Agent
value OA is abstraction (oaQuery : view[request : String, sender : String, receiver : String,
    content : view[service_id : String, service_name : String,
        params : sequence[view[name : String, val : String]], numOfParams :
Integer],
    language : String, ontology : String, reply_with : String])
{
    vConn : Connection[Boolean];
    result : Boolean;
    oaQueryConn : Connection[view[request : String, sender : String, receiver : String,
        content : view[service_id : String, service_name : String,
            params : sequence[view[name : String, val : String]], numOfParams :
Integer],
    language : String, ontology : String, reply_with : String]];

    result = true; //default initialization
    via out send "**** OA ****\n\n";
    //lookup service availability from different transformation components, and update the value
of result
    via OwlToSl send oaQuery where {oaQueryConn renames convQueryConn};

    unobservable;

    via SlToOwl send oaQuery where {oaQueryConn renames convQueryConn};

    unobservable;

    via AclToSoap send oaQuery where {oaQueryConn renames convQueryConn};

    unobservable;

    via out send "--Communicating service availability to CU--\n";
    compose
    {
        via vConn send result;
    and
        done;
    }
} //end OA

```

```

/** transformation components

value OwlToSl is abstraction (convQuery : view[request : String, sender : String, receiver : String,
    content : view[service_id : String, service_name : String,
    params : sequence[view[name : String, val : String]], numOfParams : Integer],
    language : String, ontology : String, reply_with : String])
{
    convQueryConn : Connection[view[request : String, sender : String, receiver : String,
    content : view[service_id : String, service_name : String,
    params : sequence[view[name : String, val : String]], numOfParams :
Integer],
    language : String, ontology : String, reply_with : String]];
    unobservable;
}

value SlToOwl is abstraction (convQuery : view[request : String, sender : String, receiver : String,
    content : view[service_id : String, service_name : String,
    params : sequence[view[name : String, val : String]], numOfParams : Integer],
    language : String, ontology : String, reply_with : String])
{
    convQueryConn : Connection[view[request : String, sender : String, receiver : String,
    content : view[service_id : String, service_name : String,
    params : sequence[view[name : String, val : String]], numOfParams :
Integer],
    language : String, ontology : String, reply_with : String]];
    unobservable;
}

value AclToSoap is abstraction (convQuery : view[request : String, sender : String, receiver : String,
    content : view[service_id : String, service_name : String,
    params : sequence[view[name : String, val : String]], numOfParams : Integer],
    language : String, ontology : String, reply_with : String])
{
    convQueryConn : Connection[view[request : String, sender : String, receiver : String,
    content : view[service_id : String, service_name : String,
    params : sequence[view[name : String, val : String]], numOfParams :
Integer],
    language : String, ontology : String, reply_with : String]];
    unobservable;
}

value SoapToAcl is abstraction (convQuery : view[request : String, from : String, to : String,
    body : view[class : String, fields : sequence[view[name : String, val : String]],
    properties : sequence[any], subclasses : sequence[any], restrictions : sequence[any]],

```

```

        fault : String, reply_with : String])
{
    convQueryConn : Connection[view[request : String, sender : String, receiver : String,
        content : view[service_id : String, service_name : String,
            params : sequence[view[name : String, val : String]], numOfParams :
Integer],
    language : String, ontology : String, reply_with : String]];
    soapView : view [encoding_style : String, from_http : String, to_http : String, body : String,
fault : String];
    aclView : view [encoding : String, sender : String, receiver : String, content : String,
performative : String,
        reply_to : String, ontology : String, protocol : String, conversation_identifier
: String,
        reply_with : String, in_reply_to : String, reply_by : String];
    unobservable;
}

**** transformation components end

//NM = Negotiation Module
value NM is abstraction(nmQuery : view[request : String, sender : String, receiver : String,
    content : view[service_id : String, service_name : String,
        params : sequence[view[name : String, val : String]], numOfParams :
Integer],
    language : String, ontology : String, reply_with : String])
{
    cConn : Connection[view[request : String, sender : String, receiver : String,
        content : view[service_id : String, service_name : String,
            params : sequence[view[name : String, val : String]], numOfParams :
Integer],
    language : String, ontology : String, reply_with : String]];
    nConn : Connection[view[request : String, sender : String, receiver : String,
        content : view[service_id : String, service_name : String,
            params : sequence[view[name : String, val : String]], numOfParams :
Integer],
    language : String, ontology : String, reply_with : String]];

    via cConn receive cuQuery;
    via out send "**** NM ****\n\n";
    if (nmQuery::content::service_name == "Flight reservation") do
    {
        via out send "--Runtime Reservation initiated for processing user request--\n";
        via rtFlight_Reservation send nmQuery where {nConn renames rtConn};
    }
}

```

```

}

/**runtime agents**/
value rtFlight_Reservation is abstraction(nmQuery : view[request : String, sender : String, receiver :
String,
    content : view[service_id : String, service_name : String,
        params : sequence[view[name : String, val : String]], numOfParams :
Integer],
    language : String, ontology : String, reply_with : String])
{
    rtConn : Connection[view[request : String, sender : String, receiver : String,
        content : view[service_id : String, service_name : String,
            params : sequence[view[name : String, val : String]], numOfParams :
Integer],
        language : String, ontology : String, reply_with : String]];

    uddiResp : sequence[String];
    uddiRespConn : Connection[sequence[String]];

    serviceInfo : view[service_name : String,
        params : sequence[view[name : String, val : String]], numOfParams : Integer,
message : String];
    serviceInfoConn : Connection[view[service_name : String,
        params : sequence[view[name : String, val : String]], numOfParams : Integer,
message : String]];

    via out send "***rtFlight Reservation***\n\n";

    serviceInfo::service_name = "Flight Reservation";
    serviceInfo::params = nmQuery::content::params;
    serviceInfo::numOfParams = nmQuery::content::numOfParams;

    via out send "--Sending request to UDDI--\n";
    via UDDI send serviceInfo where {uddiRespConn renames respConn};
    via uddiRespConn receive uddiResp;

    via out send "***rtFlight Reservation***\n\n--Service proposals from UDDI--\n";
    via out send uddiResp;
    via out send "\n\n";

    via DE send serviceInfo where {serviceInfoConn renames reqServiceConn};
    via serviceInfoConn receive serviceInfo;
    via out send "***rtFlight Reservation***\n\n--Optimal service returned by DE--\n";
    via out send serviceInfo;

```

```

via out send "\n\n";

via dynamic(serviceInfo::service_name) send serviceInfo where {serviceInfoConn renames
reqServiceConn};
via serviceInfoConn receive serviceInfo; //serviceInfo now contains additional parameters
request

nmQuery::request = serviceInfo::service_name;
nmQuery::sender = "";
nmQuery::receiver = "";
nmQuery::content::service_id = "";
nmQuery::content::service_name = "";
nmQuery::content::params = serviceInfo::params;
nmQuery::content::numOfParams = serviceInfo::numOfParams;
nmQuery::language = "";
nmQuery::ontology = "";
nmQuery::reply_with = "";

via rtConn send nmQuery; //sent to Actor
via rtConn receive nmQuery; //received from Actor

serviceInfo::service_name = "Flight Reservation";
serviceInfo::params = nmQuery::content::params;

via serviceInfoConn send serviceInfo; //sent to sp_norwegian
via serviceInfoConn receive serviceInfo; //received from sp_norwegian

nmQuery::reply_with = serviceInfo::message;
via out send "\n\n***rtFlight Reservation***\n\n--Service invocation successful--\n";
via rtConn send nmQuery;
}

value UDDI is abstraction (serviceQuery : view[service_name : String,
                        params : sequence[view[name : String, val : String]], numOfParams : Integer,
message : String])
{
  resp : sequence[String];
  respConn : Connection[sequence[String]];

  via out send "***UDDI***\n\n";
  if (serviceQuery::service_name == "Flight Reservation") do
  {
    unobservable; //search UDDI database on parameters criteria
    via out send "--Returning service proposals to runtime agent--\n";
  }
}

```

```

        compose
        {
            via respConn send sequence("sp_easyJet", "sp_norwegian", "sp_qatar");
            and
            done;
        }
    }
}

value DE is abstraction (reqService : view[service_name : String,
                        params : sequence[view[name : String, val : String]], numOfParams : Integer,
message : String])
{
    reqServiceConn : Connection[view[service_name : String,
                                    params : sequence[view[name : String, val : String]], numOfParams : Integer,
message : String]];

    services : sequence[view[service_name : String,
                            params : sequence[view[name : String, val : String]], numOfParams : Integer,
message : String]];
    numOfServices : Integer;
    i : Integer;
    j : Integer;
    paramMatchCount : Integer;
    fractionMatch : Float;

    services = sequence(view(service_name : "sp_easyJet",
                            params : sequence(view(name : "date", val : "20/09/2008"), view(name :
"time", val : "12h00"),
                            view(name : "price", val : "Rs. 60,000"),
                            view(name : "source", val : "CDG"), view(name : "destination", val :
"LHR")), numOfParams : 5, message : ""),
                        view(service_name : "sp_norwegian",
                            params : sequence(view(name : "date", val : "15/09/2008"), view(name :
"time", val : "15h00"),
                            view(name : "price", val : "Rs. 40,000"),
                            view(name : "source", val : "CDG"), view(name : "destination", val :
"LHR")), numOfParams : 5, message : ""),
                        view(service_name : "sp_qatar",
                            params : sequence(view(name : "date", val : "20/09/2008"), view(name :
"time", val : "12h00"),
                            view(name : "price", val : "Rs. 60,000"),
                            view(name : "source", val : "ORY"), view(name : "destination", val :
"LHR")), numOfParams : 5, message : ""));

```

```

numOfServices = 3;
i = 0;

via out send "***DE***\n\n";
while (i < numOfServices) do
{
    j = 0;
    paramMatchCount = 0;
    while (j < reqService::numOfParams) do
    {
        if (reqService::params(j)::val == services(i)::params(j)::val) do
            paramMatchCount = paramMatchCount + 1;
        j = j + 1;
    }
    fractionMatch = paramMatchCount / reqService::numOfParams;

    if (fractionMatch >= 0.8) do
    {
        reqService = services(i);
        i = numOfServices;
    }
    i = i + 1;
}

if (fractionMatch >= 0.8) do
{
    compose
    {
        via out send "--";
        via out send reqService::service_name;
        via out send " Service Selected--\n";
        via reqServiceConn send reqService;
    }
    and
    done;
}
else do
{
    done;
}
} //end DE

value sp_norwegian is abstraction (reqService : view[service_name : String,

```



```

        params : sequence[view[name : String, val : String]], numOfParams : Integer,
message : String])
{
    reqServiceConn : Connection[view[service_name : String,
        params : sequence[view[name : String, val : String]], numOfParams : Integer,
message : String]];

    //register service request
    unobservable;

    reqService = view (service_name : "Additional Parameters Request",
        params : sequence (view (name : "Name", val : ""), view (name : "Address",
val : ""),
        view (name : "Phone", val : "")), numOfParams : 3, message : "");

    via out send "***sp_norwegian***\n\n";

    compose
    {
        via out send "--Sending additional parameters request--\n";
        via reqServiceConn send reqService;
    and
        via reqServiceConn receive reqService;
        via out send "***sp_norwegian***\n\n";
        via out send "--Recieving additional parameters--\n";
        //process service request
        unobservable;
        via out send "--Initiating service--\n";
        reqService::message = "Service successfully initiated";
        via reqServiceConn send reqService;
    }
} //end sp_norwegian

```

## APPENDIX F

### List of Publications

- Ontology-based Semantic Interoperability Facilitator among Task Group, The 4<sup>th</sup> International IEEE conference on Intelligent Systems, September 2008, Varna, Bulgaria.
- Agents Negotiating with Semantic Web Services, International Conference on Education and Information Technology, October 2008, San Francisco, USA.
- Autonomous Semantic Grid: Architecture and Implementation, Open Grid Forum 19, February 2007, Charlotte, USA.
- Ontology Services between Agents and OWL Based Web Services, The 3rd IEEE International Conference on Semantics, Knowledge and Grid (SKG), October 2007, China.
- Pushing Semantic Web Service Profiles to Subscribers for Efficient Service Discovery, the 3rd IEEE International Conference on Semantics, Knowledge and Grid (SKG), October 2007, China.
- Ontology Gateway: Enabling Interoperability between FIPA compliant Agents and OWL Web Services, 9th International Conference on Enterprise Information Systems, Funchal-Madeira Portugal, 2007.
- Policy Based Migration of Mobile Agents in Disaster Management Systems, 2nd IEEE International Conference on Emerging Technologies, November 2006, Pakistan.
- An Ontology Gateway for efficient communication of Agents with Web services, IASTED International Conference on Artificial Intelligence and Soft Computing, August 2006, Palma De Mallorca, Spain.
- Ontological Translations in Semantic Grid, 4th International workshop on Multi-Agent Systems and Semantic Grid, December 2006, Pakistan.
- Semantic Interoperability between Agents and Web Services Communication, 18<sup>th</sup> Assurance Systems Symposium, 2006, Hyogo Prefecture University, Japan.
- Middleware between OWL and FIPA Ontologies in the Semantic Grid Environment, International Conference on Semantic Web and Web Services (SWWS'06), June 2006, Las Vegas, USA.

- Semantic Grid: Interoperability between OWL and FIPA SL, Lecture Notes in Computer Science, ISBN 978-3-540-36707-9, Volume 4088/2006.
- Efficient architecture for Content Language coding in Autonomous Decentralized Systems.(submitted).
- Formal Specification and Verification of Agent and Web services Communication using  $\pi$ -ADL.(to be submitted)

**Book Chapter**

- Foundation for Autonomous Semantic Grid, Nova Publishers, 2008.