AUTONOMOUS SEMANTIC GRID
Concept, Architecture, Implementation and Evaluation

By

Muhammad Omair Shafiq
2001-NUST-BIT-805

A project report submitted in the partial fulfillment of
the requirement for the degree of
Bachelors in Information Technology

In

NUST Institute of Information Technology
National University of Sciences and Technology
Rawalpindi, Pakistan
(2005)

Certificate

It is certified that the contents and form of thesis titled "**Autonomous Semantic Grid: Concept, Architecture, Implementation and Evaluation**" submitted by Muhammad Omair Shafiq have been found satisfactory for the requirement of the degree.

Advisor: _____
Associate Professor (Dr. H. Farooq Ahmad)

Co-Advisor: _____
Director General (Prof. Dr. Arshad Ali)

Committee Member 1: _____
Associate Professor (Dr. Waqar Mahmood)

Committee Member 2: _____
Dean (Dr. Syed Muhammad Hassan Zaidi)

Dedication

In the name of Almighty Allah
the Most Beneficent and the Most Merciful

To my dear parents

ACKNOWLEDGEMENTS

First of all I am thankful to The Allah Almighty for giving me will power, courage and sprit to complete this highly challenging task and to compete with international research community. I am also grateful to my family, especially my parents who have supported and encouraged through their devotion towards me and my studies and their prayers are always there with me.

I am highly thankful for the prompt guidance and help I received from Dr. H. Farooq Ahmad, Prof. Dr. Arshad Ali and Dr. Hiroki Suguri. I am thankful to Prof. Dr. Arshad Ali for his continuous and valuable suggestions and guidance, especially for the provision of all kinds of facilities throughout this research work. His ability of management and foresightedness taught me a lot of things which will be more helpful for me in practical life.

I am also thankful to Dr. Syed Muhammad Hassan Zaidi and Dr. Waqar Mahmood, for their keen interest, guidance and feedback in this research work.

I would like to express my gratitude to Mr. Aatif Kamal, Mr. Ali Hammad Akbar, Mr. Ejaz Ahmad and Wing Cdr, Maqsood-ul-Hassan for their valuable suggestions and comments to improve the dissertation and quality of work.

I am highly thankful to all of my teachers whom had been guiding me through out my course work and increased my knowledge. Their knowledge, guidance and training helped me a lot to carry out this research work.

Muhammad Omair Shafiq

ABSTRACT

Distributed Computing has three major emerging areas as Web Services Framework, Grid Computing and Multi Agent Systems. Web Service Framework is based on principles of service oriented computing for providing loosely coupled, implementation neutral and heterogeneous resources. Grid Computing focuses on coordinated resource sharing among dynamic virtual organizations. Basic Grid Computing infrastructure then evolved to OGSA by adopting Web Services Framework. It can be defined as Open Grid Services Architecture (OGSA) that supports creation, termination, management, and invocation of state-full, transient services as named, managed entities with dynamic, managed lifetime via standard interfaces and conventions. Multi Agent Systems are being evolved as distributed system in context of Autonomic Computing which provides autonomous behavior, semantic interoperability among different entities i.e. Software Agents. The Agents are autonomous entities that can control their own state. Different Agents can share a common goal or they can pursue their own interests. Multi Agent Systems develop communications languages, interaction protocols, and agent architectures. FIPA Multi Agents Systems uses its own encoding specifications and standards which are not widely accepted. On the other hand, Web Services use XML as basis which is widely accepted as industry standard for enterprise application integration. We are aimed to provide the emerging applications with a distributed system which provides autonomous behavior and semantic interoperability over widely accepted industry standards. Autonomous Semantic Grid project is aimed to provide the framework as open distributed system and is based on synergy of Web Services, Grid Computing and Multi Agent Systems. This thesis presents first milestone

of this project as AgentWeb Gateway. It is an initiative for dynamic and seamless integration of Software Agents in FIPA Multi Agent Systems and Web Services in W3C Web Services Framework. It acts as middleware between Multi Agent Systems and Web Services and facilitates the required integration without changing existing specifications. By integration, we mean enabling two way service discovery, service publishing and service invocation. This thesis presents overview of the technologies, comparative analysis, abstract architecture of the proposed system, detailed design, implementation details, evaluation and results.

TABLE OF CONTENTS

DEDICATION ..... **iii**

ACKNOWLEDGEMENTS ..... **iv**

ABSTRACT ..... **v**

CHAPTER 1: INTRODUCTION **19**

1.1 SCOPE OF RESEARCH AREA 23

1.2 RATIONAL FOR RESEARCH 25

1.3 PROBLEM STATEMENT 26

CHAPTER 2: SERVICE ORIENTED ARCHITECTURE **27**

2.1 ENTITIES OF SERVICE ORIENTED ARCHITECTURE 30

2.1.1 Service Consumer 31
2.1.2 Service Provider 31
2.1.3 Service Registry 31
2.1.4 Service Contract 31
2.1.5 Service Proxy 32
2.1.6 Service Lease 33

2.2 CHARACTERISTICS OF SERVICE ORIENTED ARCHITECTURE 34

2.2.1 Loose coupling 34
2.2.2 Implementation Neutrality 35
2.2.3 Flexible configurability 35
2.2.4 Long lifetime 35
2.2.5 Granularity 35
2.2.6 Teams 36
2.2.7 Location Transparency 36

2.3 MAJOR BENEFITS OF SERVICE-ORIENTED COMPUTING 37

2.4 WEB SERVICES 38

2.4.1 Web Services Description Language (WSDL) 41
2.4.2 Simple Object Access Protocol (SOAP) 43
2.4.3 Universal Description Discovery and Integration (UDDI) 46

CHAPTER 3: GRID COMPUTING **48**

3.1 DIFFERENCE BETWEEN GRID, CLUSTER AND THE WEB 49

3.2 GRID SERVICES 50

3.3 SEMANTIC GRID 52

3.4 SEMANTIC WEB FOR GRID COMPUTING 53

3.4.1 Semantic Grid services 53
3.4.2 Information Integration 54

3.5 SEMANTIC WEB FOR GRID APPLICATIONS 55
3.5.1 Provenance, Quality, Trust and Proof 56
CHAPTER 4: MULTI AGENT SYSTEMS 59
4.1 AGENT 60
4.2 AGENT PLATFORM 60
4.3 FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS (FIPA) 61
4.4 AGENT MANAGEMENT SYSTEM (AMS) 61
4.5 MESSAGE TRANSPORT SERVICE (MTS) 62
4.6 AGENT COMMUNICATION LANGUAGE (ACL) 62
4.7 ONTOLOGY 63
4.8 DIRECTORY FACILITATOR 63
4.9 VISUAL MANAGEMENT AGENT (VMA) 64
CHAPTER 5: LITERATURE REVIEW 65
5.1 SERVICE REGISTRIES 67
5.1.1 Universal Description Discovery and Integration (UDDI) 67
5.1.2 Directory Facilitator (DF) 69
5.2 SERVICE DESCRIPTION LANGUAGES 70
5.2.1 Web Services Description Language (WSDL) 70
5.2.2 Directory Facilitator Agent Description (DF-Agent-Description) 71
5.3 COMMUNICATION PROTOCOLS 73
5.3.1 Simple Object Access Protocol (SOAP) 73
5.3.2 Agent Communication Language (ACL) 75
CHAPTER 6: PROPOSED ARCITECTURE 78
6.1 INITIAL PROXY BASED ARCHITECTURE 80
6.1.2 Test bed 83
6.1.3 Working 84
6.2 SOFTWARE AGENT INTERACTION WITH WEB SERVICE 84
6.2.1 Agent performing service discovery in UDDI 85
6.2.2 Agent understanding a Web Service 86
6.2.3 Agent invoking a Web Service 88
6.3 WEB SERVICES INTERACTION WITH SOFTWARE AGENTS 89
6.3.1 Web Service client performing service discovery in DF 89
6.3.2 Web Service client understanding service provided by an Agent 91
6.3.3 Web Service client accessing an Agent 92
CHAPTER 7: COMPARITIVE ANALYSIS OF TECHNOLOGIES 95
7.1 SERVICE REGISTRATION AND DISCOVERY 95

7.2 SERVICE DESCRIPTION LANGUAGES 96
7.3 COMMUNICATION PROTOCOLS 99
CHAPTER 8: DETAILED DESIGN 102
8.1 SERVICE DISCOVERY CONVERTER 104
8.1.1 DF to UDDI search query conversion 105
8.1.2 UDDI to DF search query conversion 107
8.2 SERVICE DESCRIPTION CONVERTER 109
8.2.1 WSDL to DF-Agent-Description conversion 109
8.2.2 DF-Agent-Description to WSDL conversion 111
8.3 COMMUNICATION PROTOCOL CONVERTER 114
8.3.1 ACL to SOAP conversion 114
8.3.2 SOAP to ACL conversion 116
CHAPTER 9: TESTING OF PROPOSED SYSTEM 119
9.1 EVALUATION OF SERVICE DISCOVERY TRANSFORMATION 119
9.2 EVALUATION OF SERVICE DESC. TRANSFORMATION 121
9.3 EVALUATION OF COMM. PROTOCOL TRANSFORMATION 124
CHAPTER 10: PERFORMANCE ANALYSIS AND RESULTS 128
10.1 SERVICE DISCOVERY CONVERTER 128
10.2 SERVICE DESCRIPTION CONVERTER 129
10.3 COMMUNICATION PROTOCOL CONVERTER 131
10.4 TIME DISTRIBUTION AMONG TRANSFORMATION 133
10.5 TRANSFORMATION DELAY PER TRANSACTION 134
CHAPTER 11: APPLICATION OF AGENTWEB GATEWAY 136
11.1 GEOGRAPHICAL DISTRIBUTION OF SERVICES 137
11.2 ANALYSIS OF TIME DELAY IN COMMUNICATION 138
11.2.1 Evaluation scenario 138
11.2.2 Analysis of Agents interaction wit Google Web Services 138
11.2.3 Analysis of Agents communication with Grid Service 140
CHAPTER 12: FUTURE RESEARCH DIRECTIONS 141
REFERENCES 147
APPENDIX 151
A.1 INTERNATIONAL JOURNAL PUBLICATIONS 151
A.2 INTERNATIONAL CONFERENCE PUBLICATIONS 151
A.3 APPLICATION DEMONSTRATIONS IN CONFERENCES 154
A.4 RESEARCH PROPOSALS 154

LIST OF FIGURES

Figure 1.1: Scope of research..... 24
Figure 1.2: Middleware for Integration of Multi Agent Systems and Web Services 26
Figure 2.1: Software architecture describing a system's components and connectors. 27
Figure 2.2: Technologies implementing SOA 29
Figure 2.3: The "publish, find, execute" paradigm..... 30
Figure 2.4: A service proxy 33
Figure 2.5: Components of W3C compliant Web Services Framework 40
Figure 2.6: Web Services Description Language 42
Figure 2.7: Simple Object Access Protocol 44
Figure 3.1: Semantic Grid..... 52
Figure 4.1: Core components of a FIPA compliant Multi Agent System..... 59
Figure 4.2: Agents interact with environments through sensors and effectors..... 60
Figure 5.1: UDDI Schema 68
Figure 5.2: Web Services Description Language 70
Figure 5.3: Directory Facilitator Agent Description..... 72
Figure 5.4: Simple Object Access Protocol 74
Figure 5.5: Agent Communication Language..... 76
Figure 6.1: Role of Software Agents in Grid computing..... 79
Figure 6.2: Detailed design of proposed solution 82
Figure 6.3: Software Agent searching for required service in UDDI 86
Figure 6.4: Agent understanding WSDL 87
Figure 6.5: Software Agent invoking a service 89
Figure 6.6: Grid client searching for required service in Agent Platform 90
Figure 6.7: Agent publishing its services in UDDI to make it visible for Web Service clients 92
Figure 6.8: Web Service client consuming services provided by Agent 93
Figure 8.1: AgentWeb Gateway middleware..... 102
Figure 8.2: AgentWeb Gateway system architecture 103
Figure 8.3: Software Agent searching for required service in UDDI 106

Figure 8.4: Grid client searching for required service in Agent Platform 108
Figure 8.5: WSDL to DF-Agent-Description conversion: Agent understanding WSDL 110
Figure 8.6: DF-Agent-Description to WSDL conversion: Agent publishing its services in UDDI 112
Figure 8.8: SOAP to ACL conversion: WS client consuming services provided by Agent 117
Figure 10.1: Performance analysis of service discovery transformation 128
Figure 10.2: Performance analysis of service description transformation 130
Figure 10.3: Performance analysis of communication protocol transformation 132
Figure 10.4: Time distribution among required transformations 134
Figure 10.5: Time distribution among required transformations 135
Figure 11.1: Conference planner application using AgentWeb Gateway 136
Figure11.2: Geographical monitoring service for Multi Agent Systems 137
Figure 11.3: Network Delay Between Comtec Japan and Google USA 138
Figure 11.4: Between NUST Pakistan and Google USA 139
Figure 11.5: Between Comtec Japan and NUST Pakistan 140
Figure 12.1: Evolution of Service Description Language for Autonomous Semantic Grid 143
Figure 12.2: Asynchronous Invocation support for Web Services 144
Figure 12.3: Global and geographical monitoring service for Multi Agent Systems .. 145

LIST OF TABLES

Table 7.1: Comparison of UDDI and Directory Facilitator 95
Table 7.2: Comparison of WSDL and DFAgentDescription 97
Table 7.3: Comparison of SOAP and ACL 99

LIST OF ABBREVIATIONS

[A]

ACC	Agent Communication Channel
ACL	Agent Communication Language
AI	Artificial Intelligence
AID	Agent Identifiers
AMS	Agent Management System
API	Application Programming Interface

[B]

B2B	Business to Business

[C]

CORBA	Common Object Resource Broker Architecture

[D]

DAML	DARPA Agent Markup Language
DAML-OIL	DARPA Agent Markup Language – Ontology Inference Layer
DAML-S	DARPA Agent Markup Language for Services
DOM	Document Object Model
DF	Directory Facilitator

[E]

EDS	Encoding Decoding Service

[F]

FIPA	Foundation for Intelligent Physical Agents

[G]

GRAM	Grid Resource Allocation Manager
GSI	Grid Security Infrastructure
GT3	Globus Toolkit 3
GUI	Graphical User Interface
GWSDL	Grid Web service Description Language

[H]

HTTP	Hyper Text Transfer Protocol
HTTPS	Secure HTTP

[I]

IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol

[J]

JADE	Java Agent Development Framework
JAXP	Java API for XML Processing
JAX-RPC	Java API for XML-RPC

[K]

KQML	Knowledge Query Manipulation Language

[L]

LDAP	Lightweight Directory Access Protocol

[M]

MAS	Multi Agent Systems
MTS	Message Transport Service

[O]

OGSA	Open Grid services Architecture
OGSI	Open Grid services Infrastructure
OIL	Ontology Inference Layer
ORB	Object Resource Broker
OS	Operating System
OWL	Ontology Web Language
OWL-S	Ontology Web Language for Services

[P]

P2P	Peer to Peer

[Q]

QoS	Quality of Service

[R]

RMI	Remote Method Invocation
RPC	Remote Procedural Call
RDF	Resource Description Framework

[S]

SAAJ	SOAP with attachment API for Java
SAGE	Scalable Fault Tolerant Agent Grooming Environment
SAX	Simple API for XML
SL	Semantic Language
SOA	Service Oriented Architectures
SOAP	Simple Object Access Protocol

[T]

TCP/IP	Transmission Control Protocol/ Internet Protocol
[U]	
UDDI	Universal Description and Discovery Integration
[V]	
VMA	Visual Monitoring Agent
[W]	
WSCL	Web service Conversational Language
WSDL	Web Services Description Language
WSFL	Web service Flow Language
WSML	Web Services Modling Language
WSMO	Web Services Modling Ontology
WSMX	Web Services Execution Environment
WWW	World Wide Web
[X]	
XSD	XML Schema Definition
XML	Extensible Markup Language

Chapter 1

INTRODUCTION

In this modern era, mankind is exploiting computer networks to carryout its daily tasks. Whether it is a wireless network, LAN, WAN or Internet spanned across the globe, the basic purpose has remained the same since the very beginning, which is to retrieve information from distributed location and represent it in the desired form. Typical examples of such networks are mobile networks, corporate networks, factory networks, campus networks, home networks, in-car networks etc. So, whether a person is connected to a network using a desktop computer or through a mobile device from a remote location, the computer network will always be studied under the area of distributed systems.

A distributed system is a networked environment in which software or hardware components communicate and coordinate their actions only by passing messages to each other. These network elements are concurrent in nature, with no shared global clock and they are independent of each other's failures. They may be on separate continents, in the same building or in the same room. A distributed system is formed to share resources like information repositories, hardware and software to perform calculations as per the demands of a particular domain. At present, the Internet has made these distributed systems heterogeneous in nature as many solutions exist to perform tasks of similar nature. Despite of its heterogeneity, the theme is to offer continuous services to the users of Internet with high performance. The solutions that exist to form up a distributed system comprise of policies to spread out its components over a network and hardware/software technology to implement it.

Today, distributed applications and technologies that are popular among the developers and users have come a long way to become ubiquitous as the domain of distributed systems has always faced some serious challenges in its design. A solution for a particular domain that caters these issues during its development phase is used widely and becomes mature by getting feedbacks. The challenges to be confronted during distributed system development are briefly discussed below:

Heterogeneity: Participants of a distributed system can use different kinds of computer hardware, operating systems and programming languages to become part of it. This variety over the network gives rise to interoperability issues. Furthermore, middleware used in development of a distributed application always plays an important role to provide the world an interface to communicate with. Similarly, mobile codes also point towards heterogeneity problems which should be catered during the design.

Openness: Openness of a distributed system refers to extensible functionality. A distributed system provides openness if resource sharing services can be added in it for use by a variety of client programs.

Security: Designers have always faced problems to manage security risks like denial of services attacks, unauthorized access of resources like information, hardware and software. Moreover, mobile code over a network needs to be handled carefully as well.

Scalability: It refers to the effectiveness of a distributed system, if there is an increase in its number of resources or users. Through the use of efficient data structures, algorithms and policies, scalability problems can be solved.

Failure Handling: It includes design of a system in such a way so that failures can be detected, masked and tolerated. It also deals with the recovery from failures once they are tolerated. A good design always tries to provide continuous services to its users in unfavorable circumstances.

Concurrency: A distributed application can be accessed by many clients at the same time. Any object representing a shared resource in a distributed system must be responsible enough to ensure proper operation in a concurrent environment. So, an information repository in a distributed application must not provide misleading or inconsistent information to its users.

Transparency: A distributed application should act like a single logical entity to the outside world no matter how big the span of its distribution is. Transparency refers to the concealment of distributed components from the users of the system. So, a distributed application must be transparent via use of standard interfaces.
Distributed systems are the backbone of information services on the Internet. However, rapidly evolving and highly diversified world of information services requires huge information processing capacity and service provision on the Internet time scale. But the state of the art of distributed systems is human dominated administered, which cannot meet Internet time scale and quality of service for e-commerce. A critical prerequisite for distributed system technology to comply with the new challenge is that it must be completely self-tuning with autonomous adaptation to evolving workload with “zero” human administration.

As we know that the Internet was originally designed to share the information between a small numbers of users, with no quality of service requirements.

However, due to the emergence of e-commerce, there is an urgent need to change fundamental philosophy of the underlying system. Information services have become mission critical as heavy loss may result if the system does not provide required functionality and resources to achieve QoS under changing conditions, such as changing workload. The system needs to provide guaranteed quality of services at application levels, not at low level like guaranteed packet delivery. There are different concerns in quality of service, such as timeliness, reliability, and fault tolerance for information service utilization and provision. A system is called a high-assurance system, when heterogeneous and changing requirement levels of QoS are satisfied. 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. Consequently, using information services on the Internet is frustrating experience for most of the users. Many information services on the Internet return poor results- inconsistent, arbitrarily inaccurate or completely irrelevant data or the performance is so poor that the whole service becomes useless. We conclude that current information service systems on the Internet do not provide guaranteed quality of services, customization and situation based information services. There is urgent need for new models for information services for e-commerce in the Internet. If the research community fails to provide necessary technology and framework, the success of e-commerce may be delayed or even may become questionable.

This fosters an urgent need to design an information service system with high-assurance that provides information services to meet the above-mentioned requirements.

1.1 SCOPE OF RESEARCH AREA

This project comes in the domain of Distributed Computing. Distributed Computing has three major emerging areas like Web Services Framework, Grid Computing and Multi Agent Systems. Web Service Framework is based on principles of service oriented computing for providing loosely coupled, implementation neutral and heterogeneous resources. Grid Computing focuses on coordinated resource sharing among dynamic virtual organizations. Basic Grid Computing infrastructure then evolved to OGSA by adopting Web Services Framework. It can be defined as Open Grid Services Architecture (OGSA) that supports creation, termination, management, and invocation of state-full, transient services as named, managed entities with dynamic, managed lifetime via standard interfaces and conventions. OGSA specifications [13] have been re-factored as Web Services Resource Framework (WSRF) [2] due to recent developments in Web Services. The Web Services Resource Framework is inspired by the work of the Global Grid Forum's Open Grid Services Infrastructure (OGSI) Working Group. Indeed, it can be viewed as a straightforward re-factoring of the concepts and interfaces developed in the OGSI version 1.0 specification, in a manner that exploits recent developments in Web services architecture (e.g., WS-Addressing) to express these concepts and interfaces in a manner that is fully aligned with current Web services directions [2]. The WSRF specifications have not been standardized till

yet. That is why we have assumed Web Services (that would act as basis for Grid service in WSRF) as substitute to Grid Service in our design and implementation.

The Semantic Web is an idea of WWW inventor Tim Berners-Lee that the Web as a whole can be made more intelligent and perhaps even intuitive about how to serve a user's needs. Berners-Lee observes that although search engines index much of the Web's content, they have little ability to select the pages that a user really wants or needs. He foresees a number of ways in which developers and authors can use self-descriptions and other techniques so that context-understanding programs can selectively find what users want. In Semantic Grid, Web Services further combines with Semantic Web technologies to enable Dynamic web service discovery, invocation, composition, interoperation and execution monitoring.

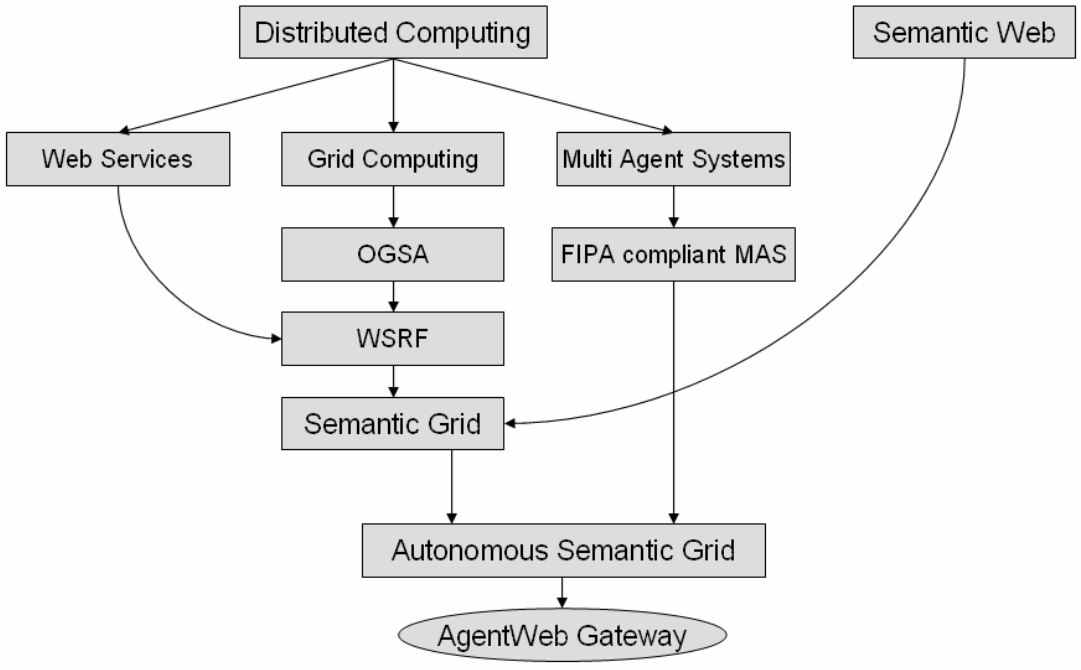

Figure 1.1: Scope of research

Multi Agent Systems focuses on systems in which intelligent Software Agents interact with each other. The Agents are autonomous entities that can control their own state. Different Agents can share a common goal or they can pursue their own interests. Multi Agent Systems develop communications languages, interaction protocols, and agent architectures.

Autonomous Semantic Grid project is aimed to provide a framework for open distributed systems and is based on synergy of Web Services, Grid Computing and Multi Agent Systems. By synergy we mean that it would combine properties of the three technologies without disturbing existing specifications to enable semantic interoperability of autonomous entities with each other, and semantically rich description of resources in Grid environment for better utilization. Software Agents will be able to discover resources in grid, form dynamic workflow, compose services, negotiate with other services or Agents to fulfill the agenda of goals.

1.2 RATIONAL FOR RESEARCH

According to ultimate Semantic Grid goals, Software Agents would be able to dynamically discover, compose, invoke and monitor web services. Software Agents and Multi Agent Systems specifications are governed by FIPA (Foundation of Intelligent Physical Agents) and specifications of Web Services are governed by W3C, hence there is a lot of difference among specifications of both technologies and hence Software Agents and Web Service cannot communicate with each other.

1.3 PROBLEM STATEMENT

To design and develop a solution that should act as middleware between Multi Agent System and Web Services Framework and without changing existing specifications of both technologies.

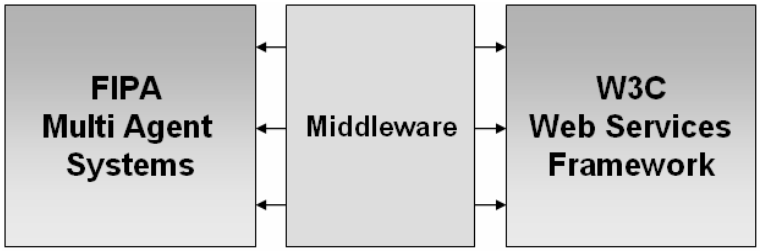

Figure 1.2: Middleware for Integration of Multi Agent Systems and Web Services

The required solution is named as AgentWeb Gateway. It facilitates require integration by providing Service Discovery transformation, Service Description transformation and Communication Protocol transformation. Which means that using AgentWeb Gateway, without changing any specification of FIPA and W3C (agents and web services).

Chapter 2

SERVICE ORIENTED ARCHITECTURE

We have explored about distributed computing and its evolution. Now focus is to provide an environment for systems that is loosely coupled and interoperable globally. It is facilitated by Service Oriented Architecture (SOA). Many of the concepts for SOA have come from principles of Service Oriented Computing. SOA configures entities (services, registries, contracts, and proxies) to maximize loose coupling and reuse. This chapter describes these entities and their configuration in an abstract way and discuss about fully implemented SOA entails. Following issues are examined here:

- What is SOA? What are its entities?
- What are the properties of SOA?
- How do I design an interface for a service?

Before analyzing the details of SOA, it is important to first explore the concept of software architecture, which consists of the software's coarse-grained structures. Software architecture describes the system's components and the way they interact at a high level.

Figure 2.1: Software architecture describing a system's components and connectors

These components are not necessarily entity beans or distributed objects. They are abstract modules of software deployed as a unit onto a server with other

components. The interactions between components are called connectors. The configuration of components and connectors describes the way a system is structured and behaves, as shown in Figure 2.1. Rather than creating a formal definition for software architecture in this chapter, we will adopt this classic definition: "The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them."

Service-oriented architecture is a special kind of software architecture that has several unique characteristics. It is important for service designers and developers to understand the concepts of SOA, so that they can make the most effective use of Web services in their environment. SOA is a relatively new term, but the term "service" as it relates to a software service has been around since at least the early 1990s, when it was used in Tuxedo to describe "services" and "service processes" (Herzum 2002). Sun defined SOA more rigorously in the late 1990s to describe Jini, a lightweight environment for dynamically discovering and using services on a network. The technology is used mostly in reference to allowing "network plug and play" for devices. It allows devices such as printers to dynamically connect to and download drivers from the network and register their services as being available.

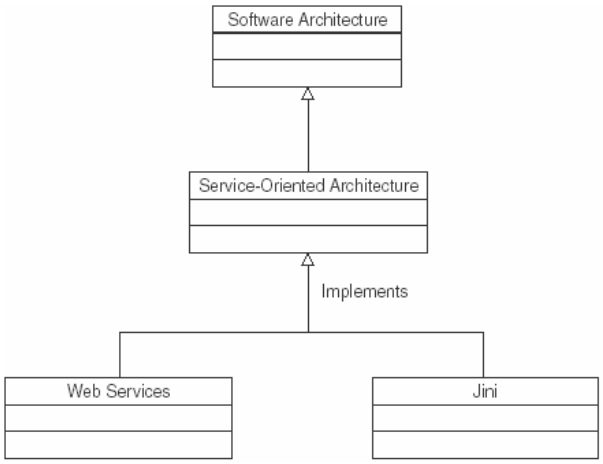

Figure 2.2: Technologies implementing SOA

The goal in developing Jini was to create a dynamically networked environment for devices, services, and applications. In this environment, services and devices could be added to and removed from the network dynamically (Sun Microsystems, Jini Network Technology, www.sun.com/jini). There is more interest lately in the software development community about the concepts behind SOA because of the arrival of Web services.

Figure 2 shows that other technologies can be used to implement service oriented architecture. Web services are simply one set of technologies that can be used to implement it successfully. The most important aspect of service-oriented architecture is that it separates the service's implementation from its interface. In other words, it separates the "what" from the "how." Service consumers view a service simply as an endpoint that supports a particular request format or contract. Service consumers are not concerned with how the service goes about executing their requests; they expect only that it will. Consumers also expect that their interaction with the service will follow a contract, an agreed-upon interaction between two parties. The way the service

executes tasks given to it by service consumers is irrelevant. The service might fulfill the request by executing a servlet, a mainframe application, or a Visual Basic application. The only requirement is that the service sends the response back to the consumer in the agreed-upon format.

2.1 ENTITIES OF SERVICE ORIENTED ARCHITECTURE

The "find, bind, and execute" paradigm as shown in Figure 3 allows the consumer of a service to ask a third-party registry for the service that matches its criteria. If the registry has such a service, it gives the consumer a contract and an endpoint address for the service. SOA consists of the following six entities configured together to support the find, bind, and execute paradigm.

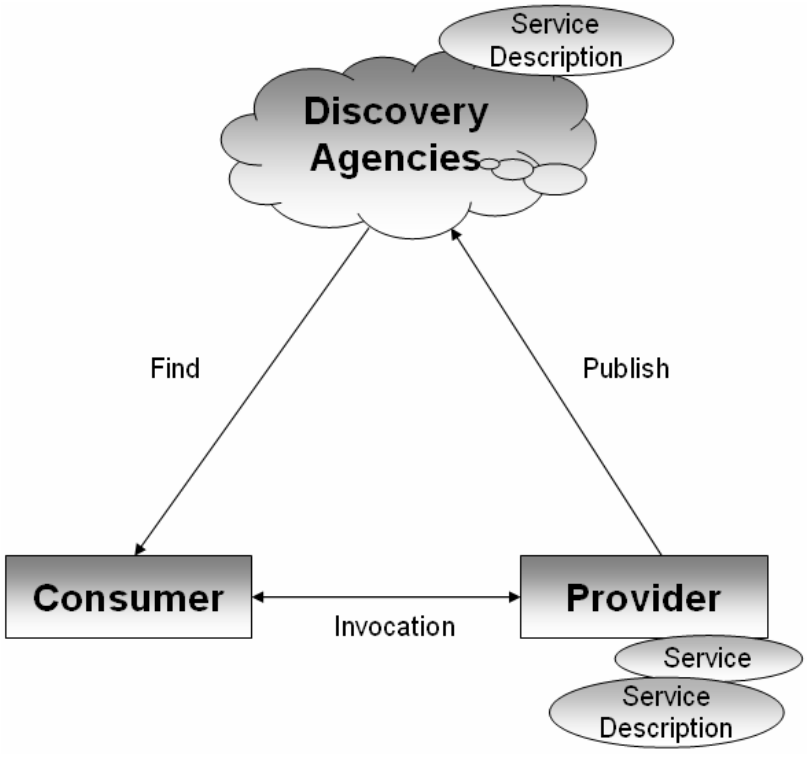

Figure 2.3: The "publish, find, execute" paradigm.

2.1.1 Service Consumer

The service consumer is an application, service, or some other type of software module that requires a service. It is the entity that initiates the locating of the service in the registry, binding to the service over a transport, and executing the service function. The service consumer executes the service by sending it a request formatted according to the contract.

2.1.2 Service Provider

The service provider is the service, the network-addressable entity that accepts and executes requests from consumers. It can be a mainframe system, a component, or some other type of software system that executes the service request. The service provider publishes its contract in the registry for access by service consumers.

2.1.3 Service Registry

A service registry is a network-based directory that contains available services. It is an entity that accepts and stores contracts from service providers and provides those contracts to interested service consumers.

2.1.4 Service Contract

A contract is a specification of the way a consumer of a service will interact with the provider of the service. It specifies the format of the request and response from the service. A service contract may require a set of preconditions and post-conditions. The preconditions and post-conditions specify the state that the service must be in to execute a particular function. The contract may also specify quality of

service (QoS) levels. QoS levels are specifications for the nonfunctional aspects of the service. For instance, a quality of service attribute is the amount of time it takes to execute a service method.

2.1.5 Service Proxy

The service provider supplies a service proxy to the service consumer. The service consumer executes the request by calling an API function on the proxy. The service proxy, shown in Figure 2.4, finds a contract and a reference to the service provider in the registry. It then formats the request message and executes the request on behalf of the consumer. The service proxy is a convenience entity for the service consumer. It is not required; the service consumer developer could write the necessary software for accessing the service directly. The service proxy can enhance performance by caching remote references and data. When a proxy caches a remote reference, subsequent service calls will not require additional registry calls. By storing service contracts locally, the consumer reduces the number of network hops required to execute the service. In addition, proxies can improve performance by eliminating network calls altogether by performing some functions locally. For service methods that do not require service data, the entire method can be implemented locally in the proxy. Methods such as currency conversion, tip calculators, and so on, can be implemented entirely in the proxy. If a method requires some small amount of service data, the proxy could download the small amount of data once and use it for subsequent method calls. The fact that the method is executed in the proxy rather than being sent to the service for execution is transparent to the service consumer. However, when using this technique it is important that the proxy support only methods the service itself provides.

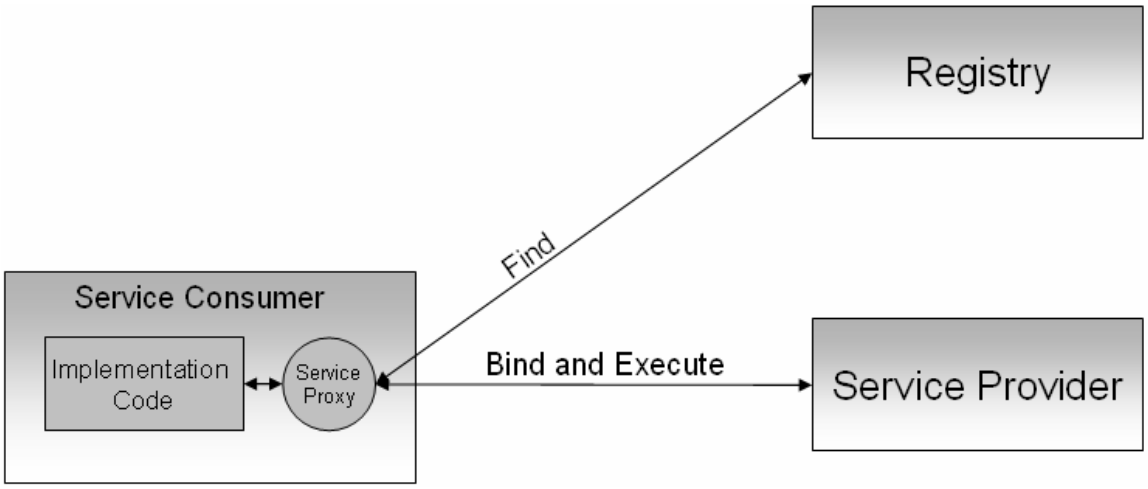

Figure 2.4: A service proxy

The proxy design pattern (Gamma et al. 2002) states that the proxy is simply a local reference to a remote object. If the proxy in any way changes the interface of the remote service, then technically, it is no longer a proxy. A service provider will provide proxies for many different environments. A service proxy is written in the native language of the service consumer. For instance, a service provider may distribute proxies for Java, Visual Basic, and Delphi if those are the most likely platforms for service consumers. Although the service proxy is not required, it can greatly improve both convenience and performance for service consumers.

2.1.6 Service Lease

The service lease, which the registry grants the service consumer, specifies the amount of time the contract is valid: only from the time the consumer requests it from the registry to the time specified by the lease (Sun Microsystems, Jini Technology Core Specification, 2001). When the lease runs out, the consumer must request a new lease from the registry. The lease is necessary for services that need to

maintain state information about the binding between the consumer and provider. The lease defines the time for which the state may be maintained. It also further reduces the coupling between the service consumer and the service provider, by limiting the amount of time consumers and providers may be bound.Without the notion of a lease, a consumer could bind to a service forever and never rebind to its contract again. This would have the effect of a much tighter coupling between the service consumer and the service provider. With a service lease, if a producer needs to somehow change its implementation, it may do so when the leases held by the services consumers expire. The implementation can change without affecting the execution of the service consumers, because those consumers must request a new contract and lease. When the new contract and lease are obtained, they are not guaranteed to be identical to the previous ones. They might have changed, and it is the service consumer's responsibility to understand and handle this change.

2.2 CHARACTERISTICS OF SERVICE ORIENTED ARCHITECTURE

To realize the above advantages, SOAs impose the following requirements:

2.2.1 Loose coupling

No tight transactional properties would generally apply among the components. In general, it would not be appropriate to specify the consistency of data across the information resources that are parts of the various components. However, it would be reasonable to think of the high-level contractual relationships through which the interactions among the components are specified.

2.2.2 Implementation Neutrality

The interface is what matters. We cannot depend on the details of the implementations of the interacting components. In particular, the approach cannot be specific to a set of programming languages.

2.2.3 Flexible configurability

The system is configured late and flexibly. In other words, the different components are bound to each other late in the process. The configuration can change dynamically.

2.2.4 Long lifetime

We do not necessarily advocate a long lifetime for our components. However, since we are dealing with computations among autonomous heterogeneous parties in dynamic environments, we must always be able to handle exceptions. This means that the components must exist long enough to be able to detect any relevant exceptions, to take corrective action, and to respond to the corrective actions taken by others. Components must exist long enough to be discovered, to be relied upon, and to engender trust in their behavior.

2.2.5 Granularity

The participants in an SOA should be understood at a coarse granularity. That is, instead of modeling actions and interactions at a detailed level, it would be better to capture the essential high-level qualities that are (or should be) visible for the purposes of business contracts among the participants. Coarse granularity reduces

dependencies among the participants and reduces communications to a few messages of greater significance.

2.2.6 Teams

Instead of framing computations centrally, it would be better to think in terms of how computations are realized by autonomous parties. In other words, instead of a participant commanding its partners, computation in open systems is more a matter of business partners working as a team. That is, instead of an individual, a team of cooperating participants is a better modeling unit. A team-oriented view is a consequence of taking a peer-to-peer architecture seriously.

2.2.7 Location Transparency

Location transparency is a key characteristic of service-oriented architecture. Consumers of a service do not know a service's location until they locate it in the registry. The lookup and dynamic binding to a service at runtime allows the service implementation to move from location to location without the client's knowledge. The ability to move services improves service availability and performance. By employing a load balancer that forwards requests to multiple service instances without the service client's knowledge, we can achieve greater availability and performance. As mentioned earlier, a central design principle in object-oriented systems is separation of implementation from interface. This means that an object's interface and its implementation may vary independently. The primary motivation for this principle is to control dependencies between objects by enforcing the interface contract as their only means of interaction. Service-oriented architecture takes this

principle one step further, by reducing the consumer's dependency on the contract itself. This reduced dependency through the use of dynamic binding also has the effect of making the service's location irrelevant. Because the service consumer has no direct dependency on the service contract, the contract's implementation can move from location to location.

2.3 MAJOR BENEFITS OF SERVICE-ORIENTED COMPUTING

It is worth considering the major benefits of using standardized services here. Clearly anything that can be done with services can be done without. So what are some reasons for using services, especially in standardized form? The following are the main reasons that stand out.

Services provide higher-level abstractions for organizing applications in large-scale, open environments. Even if these were not associated with standards, they would be helpful as we implemented and configured software applications in a manner that improved our productivity and improved the quality of the applications that we developed.

Moreover, these abstractions are standardized. Standards enable the interoperation of software produced by different programmers. Standards thus improve our productivity for the service use cases described above.

Standards make it possible to develop general-purpose tools to manage the entire system lifecycle, including design, development, debugging, monitoring, and so on. This proves to be a major practical advantage, because without significant tool support, it would be nearly impossible to create and field robust systems in a feasible manner. Such tools ensure that the components developed are indeed interoperable,

because tool vendors can validate their tools and thus shift part of the burden of validation from the application programmer.

The standards feed other standards. For example the above basic standards enable further standards, e.g., dealing with processes and transactions.

2.4 WEB SERVICES

XML is the foundation for a Web Service framework within which automated, decentralized services can be defined, deployed, manipulated and evolved. It is based on principles of service oriented computing. It provides a structure for integration and a foundation for protocols that will support the needs of distributed applications. The goal is a scalable, layered architecture, one that could meet the needs of both simple and robust deployments.

While most descriptions of Web based solutions emphasize their distributed characteristics, their decentralized nature, they have distinct management and control environments and communicate across trust domains and have much more impact on architecture of this framework and the requirements of the underlying protocols. The focus of the framework is defining a model for describing, discovering and exchanging information that is independent of application implementations and the platforms on which applications are developed and deployed. It also focuses on to connect applications on a worldwide basis. Such applications will necessarily be built in a variety of programming languages, using a range of operating systems, database, and middleware technologies. The interoperability can only be achieved when based on standard data formats and protocols, not APIs. By focusing "on the wire", we define just the specifications needed for interconnection. We believe this approach provides

the greatest benefit in the shortest time, and does not impinge of software vendors' flexibility and enterprise autonomy.

The components in the framework are correlated and can be organized into three parts: communication protocol, service description and service discovery. It is also the case that some components depend on others. For example, the communication protocol will provide the basis for the discovery and description stacks.

Communication Protocol – It represent what is sent during a given exchange, the combination of the data, the envelope and all other metadata necessary for the successful transmission of a message.

- Message envelope
- Message Exchange
- Business (Long-Running) Transactions
- Digital Signature
- Encryption

Service Description – It is the collection of specifications that provide the formal definition of the format, use, or application, of the specs in the "Wire" stack.

- XML Schemas
- Service Description
- Process Flow Orchestration

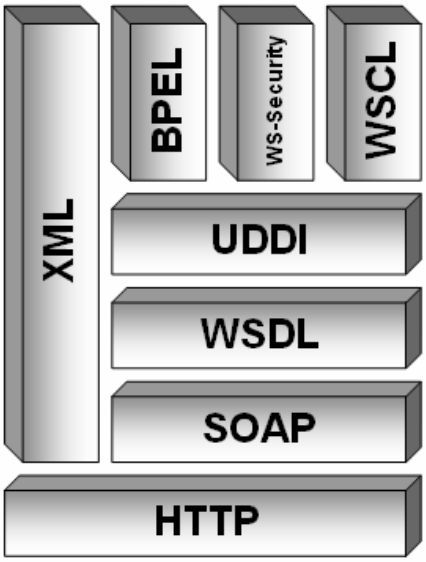

Figure 2.5: Components of W3C compliant Web Services Framework

The description module is modular i.e. the offerings are both layered and ordered. Each technology is built on the ones above, and simpler ones provide useful function by themselves. Therefore, we anticipate that schemas and WSDL will be deployed first to provide descriptions for individual messages and message pairs. Over time, tools will become available to support the increasingly rich descriptions enabled by the other standards.

Service Discovery – It provides a means for manual or automated searching and discovery of the components in the other two stacks.

- Inspection
- Discovery

Below is a brief description of the platform elements. It should be noted that while vendors try to present the emergent web services platform as coherent, it's

really a series of in-development technologies. Often at the higher levels there are, and may remain, multiple approaches to the same problem.

- WSDL (expression of service characteristics)
- SOAP (service invocation)
- UDDI (trader, service registry, discovery agency)

2.4.1 Web Services Description Language (WSDL)

WSDL is an XML based document for describing services provided by a Web Service as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information [26]. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints. WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate.

WSDL defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitute a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Hence, a WSDL document uses the following elements in the definition of network services:

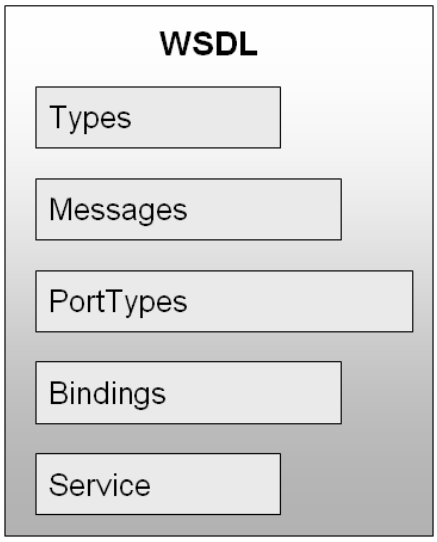

Figure 2.6: Web Services Description Language

- Types: a container for data type definitions using some type system (such as XSD).
- Message: an abstract, typed definition of the data being communicated.
- Operation: an abstract description of an action supported by the service.
- Port Type: an abstract set of operations supported by one or more endpoints.
- Binding: a concrete protocol and data format specification for a particular port type.
- Port: a single endpoint defined as a combination of a binding and a network address.
- Service: a collection of related endpoints.

The main structure of a WSDL document looks like this:

```
<definitions>
<types>
```

```
  definition of types.....
</types>
<message>
  definition of a message....
</message>
<portType>
  definition of a port.......
</portType>
<binding>
  definition of a binding....
</binding>
</definitions>
```

2.4.2 Simple Object Access Protocol (SOAP)

SOAP is a simple XML based protocol to let applications exchange information over HTTP. SOAP is a communication protocol for accessing a Web Service. It stands for Simple Object Access Protocol. It is used for communication between applications and provides a format for sending messages via Internet. SOAP is platform independent, language independent as based on XML. It is simple and extensible. It allows you to get around firewalls. SOAP will be developed as a W3C standard.

It is important for application development to allow Internet communication between programs. Today's applications communicate using Remote Procedure Calls (RPC) between objects like DCOM and CORBA, but HTTP was not

designed for this. RPC represents a compatibility and security problem; firewalls and proxy servers will normally block this kind of traffic.

A better way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this.

SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

A SOAP message is an ordinary XML document containing the following elements:

- A required Envelope element that identifies the XML document as a SOAP message
- An optional Header element that contains header information
- A required Body element that contains call and response information
- An optional Fault element that provides information about errors that occurred while processing the message

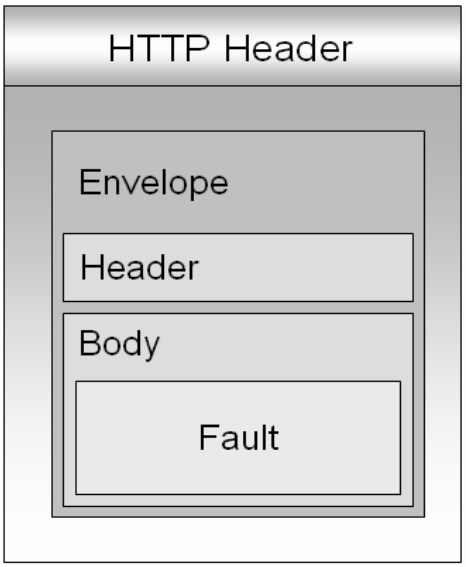

Figure 2.7: Simple Object Access Protocol

SOAP Example

The SOAP request:

```
POST /InStock HTTP/1.1
Host: www.url.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
<Envelope>
  <Body>
    <OperationName>
      <Input_param_value>  … </Input_param_value>
    </OperationName>
  </Body>
</Envelope>
```

A SOAP response:

```
HTTP/1.1 200 OK
Content-Type: application/soap; charset=utf-8
Content-Length: nnn
<Envelope>
  <Body>
    <Operation_name>
      <Output_param_value> … </Output_param_value>
    </Operation_name >
  </Body>
</Envelope>
```

2.4.3 Universal Description Discovery and Integration (UDDI)

Universal Description, Discovery, and Integration (UDDI) provides the definition of a set of services supporting the description and discovery of (1) businesses, organizations, and other Web Services providers, (2) the Web Services they make available, and (3) the technical interfaces which may be used to access those services. The idea is to "discover" organizations and the services that organizations offer, much like using a phone book or dialing information.

UDDI was first developed by UDDI.org and then transferred to OASIS. UDDI.org was comprised of more than 300 business and technology leaders working together to enable companies and applications to quickly, easily, and dynamically find, and use Web Services.

UDDI is based on a common set of industry standards, including HTTP, XML, XML Schema, and SOAP. It provides an infrastructure for a Web Services-based software environment for both publicly available services and services only exposed internally within an organization. The UDDI Business Registry system consists of three directories:

UDDI white pages: basic information such as a company name, address, and phone numbers, as well as other standard business identifiers like Dun & Bradstreet and tax numbers.

UDDI yellow pages: detailed business data, organized by relevant business classifications. The UDDI version of the yellow pages classifies businesses according to the newer NAICS (North American Industry Classification System) codes, as opposed to the SIC (Standard Industrial Classification) codes.

UDDI green pages: information about a company's key business processes, such as operating platform, supported programs, purchasing methods, shipping and billing requirements, and other higher-level business protocols.

Chapter 3

GRID COMPUTING

WWW has facilitated unprecedented ways of speedy global information sharing. The Grid technologies build on this by allowing facilitating the global sharing of not just information, but also of tangible assets (computational and data storage resources) to be used at a distance. E mail and WWW provide basic mechanisms that allow communities that span states, countries and continents to work together. But what if they could link their data, computers and other resources into a single virtual office? - Grid seeks to make this possible by providing the protocols, services and software development kits needed to enable flexible, controlled resource sharing on a large scale. At the heart of 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 primarily file exchange, but rather direct, controlled (i.e. within the authorization, security, copyright, etc. restrictions) access to computers, software, data and other resources, as is required by a range of collaborative problem solving and resource brokering strategies emerging in industry, science and engineering.

Grid computing is an innovative approach that leverages existing IT infrastructure to optimize compute resources and manage data and computing workloads. According to Gartner, "a grid is a collection of resources owned by multiple organizations that is coordinated to allow them to solve a common problem." Gartner further defines three commonly recognized forms of grid:

1. Computing grid - multiple computers to solve one application problem

2. Data grid - multiple storage systems to host one very large data set
3. Collaboration grid - multiple collaboration systems for collaborating on a common issue

Grid computing is not a new concept but one that has gained recent renewed interest and activity for a couple of main reasons:

1. IT budgets have been cut, and grid computing offers a much less expensive alternative to purchasing new, larger server platforms.
2. Computing problems in several industries involve processing large volumes of data and/or performing repetitive computations to the extent that the workload requirements exceed existing server platform capabilities.

Some of the industries that are interested in grid computing including life sciences, computer manufacturing, industrial manufacturing, financial services, and Government.

3.1 DIFFERENCE BETWEEN GRID COMPUTING, CLUSTER COMPUTING AND THE WEB

Cluster computing focuses on platforms consisting of often homogeneous interconnected nodes in a single administrative domain.

1. Clusters often consist of PCs or workstations and relatively fast networks
2. Cluster components can be shared or dedicated
3. Application focus is on cycle-stealing computations, high-throughput computations, distributed computations

Web focuses on platforms consisting of any combination of resources and networks which support naming services, protocols, search engines, etc. Web

consists of very diverse set of computational, storage, communication, and other resources shared by an immense number of users. Application focus is on access to information, electronic commerce, etc.

Grid focus on ensembles of distributed heterogeneous resources used as a platform for high performance computing

- Some grid resources may be shared, other may be dedicated or reserved
- Application focus is on high-performance, resource-intensive applications

3.2 GRID SERVICES

Grid middleware should enable new capabilities to be constructed dynamically and transparently from distributed services. In order to engineer new Grid applications it is desirable to be able to reuse existing components and information resources and to assemble and co-ordinate these components in a flexible manner. Partly for this reason the Grid is moving away from a collection of protocols to a service-oriented approach: the Open Grid services Architecture (OGSA). This unites Web services with Grid requirements and techniques.

The Grid's requirements mean that Grid services extend Web services considerably. Grid service configurations are:

- dynamic and volatile A consortium of services (databases, sensors, compute servers) participating in a complex analysis may be switched in and out as they become available or cease to be available;
- ad-hoc. Service consortia have no central location, no central control, and no existing trust relationships;
- large. Hundreds of services could be orchestrated at any time;

- long-lived. A simulation could take weeks.

These requirements make strenuous demands on fault tolerance, reliability, performance and security. Whereas Web services are presumed to be available and stateless, Grid services are presumed to be transient and stateful.

Grid services are broadly organised into four tiers:

1. Fabric (security, data transport, certification, remote access, network monitoring, ownership and digital watermarking, authentication);
2. Base (resource scheduling, data access, event notification, metadata management, provenance, versioning);
3. High Level (workflow, database management, personalisation);
4. Application (a gene sequence alignment, a Swiss-Prot database, a gene finding algorithm).

Each tier relies on metadata. To achieve the flexible assembly of Grid services requires information about the functionality, availability and interfaces of the various services. Service discovery and brokering uses metadata descriptions. Service composition is controlled and supported by metadata descriptions. Metadata is the key to achieving the Grid services vision.

The Grid technologies build on Web allows facilitating the global sharing of not just information, but also of tangible assets (computational and data storage resources) to be used at a distance. Grid seeks to make this possible by providing the protocols, services and software development kits needed to enable flexible, controlled resource sharing on a large scale.

Semantic Grid is an initiative 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 Semantic Web.

3.3 SEMANTIC GRID

Until very recently the Grid and the Semantic Web communities were separate, despite the convergence of their respective visions. Both have a need for computationally accessible and sharable metadata to support automated information discovery, integration and aggregation. Both operate in a global, distributed and changeable environment.

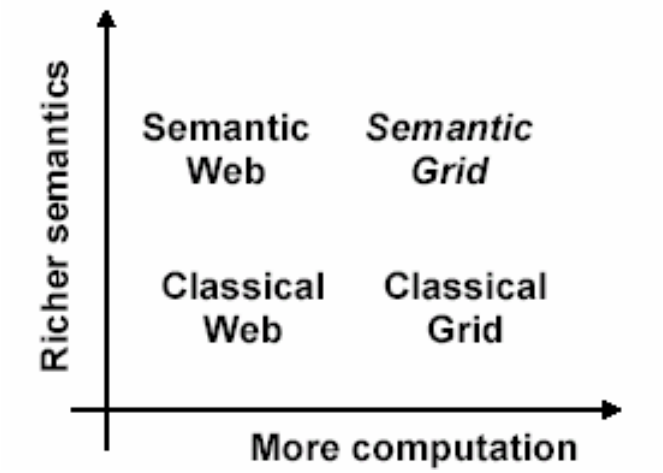

Figure 3.1: Semantic Grid

The Semantic Web base services can be Grid Base Services. The Semantic Web fabric is the means by which the Grid could represent metadata: both for Grid infrastructure, driving the machinery of the Grid fabric, and its base and high level

services, and for Grid applications, representing the knowledge and operational know-how of the application domain.

3.4 SEMANTIC WEB FOR GRID COMPUTING

3.4.1 Semantic Grid services

The description of a service is essential for automated discovery and search, selection, matching, composition and interoperation, invocation and execution monitoring. This choice depends on service metadata. Classification of services based on the functionality they provide has been widely adopted by diverse communities as an efficient way of finding suitable services, e.g. UDDI. Reasoning over service descriptions has a role to play when classifying and matching services. In Condor a matching mechanism is used to choose computational resources. In an architecture where the services are highly volatile, and configurations of services are constantly being disbanded and re-organised, knowing if one service is safely substitutable by another is essential.

At the time of writing, the current state of describing Grid services through semantics is by using the names assigned the portType and serviceType elements of a WSDL document, linked to a specification document. Bringing together the Semantic Web and Web services has already attracted attention. DAML+OIL has been explored in myGrid. The myGrid service ontology extends the DAML-S ontologies. Service classifications are more expressive than UDDI's simple hierarchies and services are queried and matched by subsumption reasoning over the service descriptions. However, Grid services dynamically create and destroy service instances, have soft state registration and form long-lived service configurations. How this affects

the way Semantic Web technologies can describe and discover Grid services is a challenge yet to be adequately addressed.

3.4.2 Information Integration

Complex questions posed by scientists require the fusion of evidence from different, independently developed and heterogeneous resources. In biology, for example, the hundreds of data repositories in active service have different formats, interfaces, structures, coverage. The Web and the Data Grid guarantee a certain level of interoperability in retrieving and accessing data. The next level of interoperability is not just making data available, but understanding what the data means so that it can be linked in appropriate and insightful ways, and providing automated support for this integration process.

Scientists typically link resources in two ways:

1. Workflow orchestration: Process flows, or workflows coordinating and chaining services using a systematic plan, are the manifestation of in silico experiments, allowing us to represent the e-Scientist's experimental process explicitly;
2. Database integration: dynamic distributed query processing, or the creation of integrated databases through virtual federations or warehouses.

Information mediation is not restricted to traditional scientific databases. Computational resources are discovered, allocated and disbanded dynamically and transparently to the user. The problem of mediation between different Grid compute resource brokering models, such as Unicore and Globus, closely resembles mediation between two database schemas.

Semantic Web and Database technologies offer great possibilities. A common data model for aggregating results drawn from different resources or instruments could use RDF. Domain ontologies for the semantic mediation between database schema, an application's inputs and outputs, and workflow work items could use DAML+OIL/RDF(S). Domain ontologies and rules can be used for constraining the parameters of machines or algorithms, and inferring allowed configurations. Execution plans, workflows and other combinations of services benefit from reasoning to ensure the semantic validity of the composition.

So we can use Semantic Web services for:

- The classification of computational and data resources, performance metrics, job control; schema integration, workflow descriptions;
- Typing data and service inputs and outputs;
- Problem solving selection and intelligent portals;
- Infrastructure for authentication, accounting and access management.

Turning this around, we can envisage that the Base and Application services of the Semantic Web are implemented as Grid services.

3.5 SEMANTIC WEB FOR GRID APPLICATIONS

The ultimate purpose of the Grid is to support knowledge discovery. The Semantic Web is often presented as a global knowledge base. Consider a scenario: A scientist posing the question "what ATPase superfamily proteins are found in mouse?" might get the answers (a) The protein accession number from the Swiss-Prot database she has permission to access; (b) InterPro is a pattern database but needs permission and payment. (c) Attwood's project is in nucleotide binding proteins (ATPase

superfamily proteins are a kind of nucleotide binding protein); (d) Smith published a new paper on something similar in Nature Genetics two weeks ago; (e) Jones in your lab already asked this question last week.

A scientist may be advised of equipment or algorithm parameter settings, helped to choose and plan appropriate experiments and resources based on her aims and shared best practice, and ensure that conclusions are not drawn that are not fully justified by the techniques used. These are all applications of, or for, the Semantic Web, and include personalized agents or services, semantic portals onto services, recommender systems and a variety of other knowledge services.

The scientific community has embraced the Web. The result is commonly publication of information without accompanying accessibility. Many resources have simple call interfaces without APIs or query languages and only "point and click" visual interfaces. Scientific knowledge is often embodied in the literature and in free text "annotations" attached to raw data. The presumption that a scientist will read and interpret the texts makes automatic processing hard and is not sustainable given the huge amount of data becoming available. The Semantic Web is about making the computationally inaccessible accessible and to automate information discovery.

3.5.1 Provenance, Quality, Trust and Proof

Both the results and the way they are obtained are highly valued. Where data came from, who created it, when, why and how was it derived is as important as the data itself for user and service provider. These are applications of the Proof, Trust and Digital Signatures of the Semantic Web. In molecular biology, data is repeatedly copied, corrected and transformed as it passes through numerous databases. Published

data is actively curated automatically and by hand. Complex assemblies of programs create results from base data. Annotating results with commentaries, linking results with their sources, asserting which parameters were used when running an algorithm and why, are possible applications of Semantic Web and database technologies.

Assertions are also qualitative. Scientific knowledge is contextual and opinionated. Contexts change and opinions disagree. New information may support or contradict current orthodoxy leading to a revision of beliefs. Inferences on assertions can give new knowledge but inferences must be exposed or else the scientist will not use them. Dealing with multiple (diverging) assertions over resources, and inference engines capable of tolerating discrepancies, is a challenge of the Semantic Web.

So Semantic Web services can be for:

- annotating results, workflows, database entries and parameters of analyses with: personal notes, provenance data, derivation paths of information, explanations or claims;
- linking in silico and 'at the bench' experimental components: literature, notes, code, databases, intermediate results, sketches, images, workflows, the person doing the experiment, the lab they are in, the final paper;
- describing people, labs, literature, tools and scientific knowledge

Scientific knowledge is replicated and archived for safe-keeping. It is essential to be able to recall a snapshot of the state of understanding at a point in time in order to justify a scientific view held at that time. This raises questions: What does it mean to garbage collect the 'Semantic Grid', and how do we recover a snapshot?

Grid services come and go, which is why event notification is a Grid base service. As data collections and analytical applications evolve, keeping track of the impact of changes is difficult. Scientists rerun their queries if base data changes or new knowledge questions the underlying premise of an analysis. Mistakes or discredited information are propagated and difficult to eliminate. The ontologies and rules change. When an ontology changes in line with new beliefs, this does not wipe the old inferences that no longer hold (and how do we propagate those changes?). They must continue to co-exist and be accessible. Monitored events and items can be described using ontologies; database triggers can implement the notification mechanism.

Chapter 4

MULTI AGENT SYSTEMS

Multi-agent systems (MAS) are one of the landmark technologies in software-based framework that provide collaborative environment for a community of social agents for the provision of continuous and dynamic services.

Multi-agent systems are systems composed of multiple agents, which interact with one another, typically by exchanging messages through some computer network infrastructure. MAS provide proper execution environment to agents so that they can assure the provision of services to other agents by cooperating, coordinating, and negotiating.

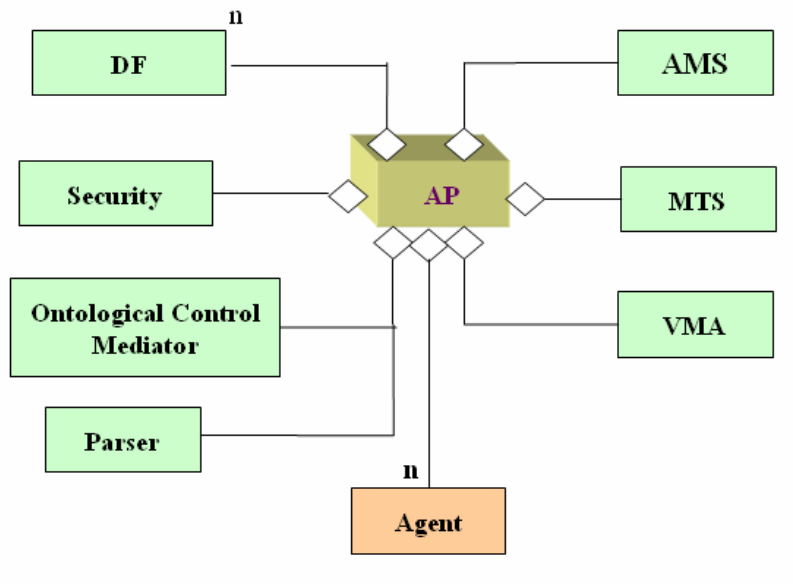

Figure 4.1: Core components of a FIPA compliant Multi Agent System

MAS 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 to auctions, etc.)

4.1 AGENT

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through effectors. A human agent has eyes, ears and other organs for sensors and hands, legs, mouths and other body parts for effectors.

An Agent is a computer system that is capable of independent action on behalf of its user or owner. Agents in a multi-agent system will be representing or acting on behalf of users or owners with very difficult goals or motivations.

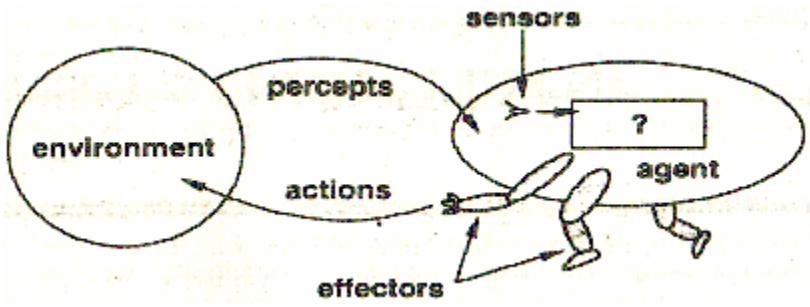

Figure 4.2: Agents interact with environments through sensors and effectors

4.2 AGENT PLATFORM

Software agents provide multiple services. For the provision of these services, 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 agents can be created and can behave according to their specification is called Agent Platform. Many Agent Platforms provide environment for the community of agents for the provision of dynamic services.

4.3 FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS (FIPA)

Foundation for Intelligent Physical Agents (FIPA) is a standard governing body for Agent development community. It provides abstract architecture of a complete Multi- agent System. Concrete realization of the abstract architecture will be according to the choice of the developer. Till now many FIPA compliant MAS have been implemented, JADE is one of the examples of FIPA compliant MAS.

4.4 AGENT MANAGEMENT SYSTEM (AMS)

Expected growth of Multi Agent Systems (MAS) with community of social agents in heterogeneous applications has made it focal point for research. All the agents within MAS are managed by Agent Management System (AMS) which is the mandatory supervisory authority of any MAS. A single agent platform can be distributed over several machines which provide scalability and load balancing etc. But with centralized AMS, this infrastructure lacks fault tolerance, which is a key feature of high assurance. Absence of fault tolerance is the main reason for the small number of deployments of MASs. Failure of AMS leads towards abnormal behavior in the distributed platform.

Virtual Agent Cluster Paradigm (VAC) is proposed in this regard which strongly supports decentralized distributed AMS to achieve fault tolerance in distributed MAS. VAC is an autonomous distributed infrastructure which provides fault tolerance by using separate communication layers among distributed peers. Experimental results show that it improves performance, brings autonomy and supports fault recovery along with load balancing in distributed MAS.

4.5 MESSAGE TRANSPORT SERVICE (MTS)

Message Transport Service is the backbone of any MAS. It supports the sending and receiving of ACL messages between agents. The agents involved may be local to a single Agent Platform or on different Agent Platforms. Two modes of communication are involved for message transportation.

1. Intra-platform Communication (MTS)
2. Intra-machine
3. Inter-machine
4. Inter-platform Communication (ACC)

4.6 AGENT COMMUNICATION LANGUAGE (ACL)

The ACL package is responsible for creation of a message that's 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 ACL Message.

Agent Communication Languages provides agents with a means of exchanging information and knowledge, which is really the essence of all forms of interaction in multi-agent systems. The result of which, was the FIPA ACL. ACL is an outer language that specifies message format and include 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 SL which provides rich semantics. Every agent has common semantics to talk each other that is based on shared ontology.

4.7 ONTOLOGY

"A formal, explicit specification of a shared conceptualization". "A hierarchically structured set of terms to describe a domain that can be used as a skeletal foundation for a knowledge base". Ontology is a shared vocabulary. Ontology can be considered as different "concepts" linked together.

Components of Ontology:

1. Concept
2. Predicate
3. Action

4.8 DIRECTORY FACILITATOR

Directory Facilitator (DF) is an optional component of multi agent system. It is responsible to provide 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. Agent is responsible to provide information related to service e.g. servie_type, service_name etc. Furthermore, an agent can also deregister or modify service

Any agent can interact with a DF in the following situations:

To make its services public, To identify agents that provides a particular service through the yellow-pages

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.

4.9 VISUAL MANAGEMENT AGENT (VMA)

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 dministrative interaction with the AMS agent, the DF agent and are also be 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.
- Sniff a particular agent (optional).
- 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 AMS agent and DF agent through passing ACL messages. For the creation of ACL messages VMA package will use ACL package and will compose ACL message. After that the ACL message will be send to the Message Transport Service that will forward that message to the respective agent.

Chapter 5

LITERATURE REVIEW

The related work in this area is regarding integration of Software Agents and Web Services.

The tool in [4] generates the ontology that describes the web service call signature and the java code of an example agent that can be deployed in any Jade agent platform. The generated ontology can be published for client agents that can then send requests to and accept responses from the example agent and performs all the necessary translations between agent communication language messages (agent requests, responses) and SOAP messages (web service calls, results) before and after it calls the web service, respectively.

In [5], authors examine the question of how agent technology can be used to personalize web services. In particular, they address the challenge of how a customer can assign a delegate that will programmatically interact with web services according to context when acting on the behalf of a customer. They have identified a number of issues that web service and agent platforms must evolve to address in order for the two paradigms to work together, and propose a personalization component that can be integrated with existing web service infrastructures.

In [16], the purpose was to demonstrate that one can use agent technology to assist in the construction and enactment of e-Science experiments. They have constructed a tool based on this language which allows experiments to be rapidly constructed, verified, and enacted. The language proposed is a lightweight formalism, providing only a minimal set of operations. This was a deliberate choice as it allowed to

define the language and the type system without unnecessary complication. Another issue that they intend to address, concerns the discovery of web services. At present, the web services that are used to define an experiment must be known in advance, and must be explicitly registered before enactment. Furthermore, the protocol must be defined to precisely match the WSDL definition of the web service. In order to reduce the restrictions, and allowing a more extensible kind of coordination, that allows for semi-automatic web service discovery and invocation. For this, they intend to semantically annotate web services, on which one can reason about the behavior of the services.

Regarding some of the advanced features discussed in the latter half of this paper, we note that several other research communities like in [4] are working on approach that supports the dynamic selection of services as a path toward autonomic computing where computational resources are self-managing and self-configuring. In addition, in [1] and [16] the authors discuss the use of process description languages for enacting business processes in the contemporary workplace. They note that strict adherence to prescribed workflows implies that systems are largely unable to adapt effectively to unforeseen circumstances. The work proposes that, workflow description languages be used to specify multi agent systems, specifically advancing the idea that the Business Process Execution Language for Web Services [2] can be used as a specification language for expressing the initial social order of a multi agent system, which can then intelligently adapt to changing environmental conditions.

Open Grid Services Architecture [13] is set of services provided by different organizations based on principles of service oriented computing. Open Grid Services Infrastructure (OGSI) specifications [1] have been re-factored as Web

Services Resource Framework (WSRF) [2] due to recent developments in Web Services. The Web Services Resource Framework is inspired by the work of the Global Grid Forum's Open Grid Services Infrastructure (OGSI) Working Group. Indeed, it can be viewed as a straightforward re-factoring of the concepts and interfaces developed in the OGSI version 1.0 specification, in a manner that exploits recent developments in Web services architecture (e.g., WS-Addressing) to express these concepts and interfaces in a manner that is fully aligned with current Web services directions [2]. The WSRF specifications have not been standardized till yet. That is why we have assumed Web Services (that would act as basis for Grid service in WSRF) as substitute to Grid Service in our design and implementation.

5.1 SERVICE REGISTRIES

In service oriented architecture, service registry is one of the key components and is used to register and search services. In this section, we describe service registries of both the technologies, i.e. Directory Facilitator (DF) for Multi Agent Systems and Universal Description Discovery and Integration (UDDI) for Web Services.

5.1.1 Universal Description Discovery and Integration (UDDI)

Universal Description, Discovery, and Integration (UDDI) provides the definition of a set of services supporting the description and discovery of businesses, organizations, and other Web Services providers, the Web Services they make available, and the technical interfaces which may be used to access those services. The idea is to discover organizations and the services that organizations offer, much like using a phone book or dialing information.

Figure 5.1: UDDI Schema

UDDI is based on a common set of industry standards, including HTTP, XML, XML Schema, and SOAP. It provides an infrastructure for a Web Services-based software environment for both publicly available services and services only exposed internally within an organization. The UDDI Business Registry system consists of three directories:

UDDI white pages: basic information such as a company name, address, and phone numbers, as well as other standard business identifiers like Dun & Bradstreet and tax numbers.

UDDI yellow pages: detailed business data, organized by relevant business classifications. The UDDI version of the yellow pages classifies businesses according to the newer NAICS (North American Industry Classification System) codes, as opposed to the SIC (Standard Industrial Classification) codes.

UDDI green pages: information about a company's key business processes, such as operating platform, supported programs, purchasing methods, shipping and billing requirements, and other higher-level business protocols.

Figure 2 describes schema of UDDI for storing information about services. In has Business Entity that contains information about different business organizations. Each business organization can have one Business Entity and multiple Business Services.

5.1.2 Directory Facilitator (DF)

Directory Facilitator (DF) is a core component of FIPA compliant Multi Agent System. It is responsible to provide 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. Agent is responsible to provide information related to service e.g. servie_type, service_name etc. Furthermore, an agent can also deregister or modify service

Any agent can interact with a DF in the following situations:

- To make its services public, to identify agents that provides a particular service through the yellow-pages
- 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.

5.2 SERVICE DESCRIPTION LANGUAGES

In service oriented architecture, service providers abstractly describe services which is used to be published in service registries and used by service consumers to search for the required services. This section provides concise information about service description languages of both technologies.

5.2.1 Web Services Description Language (WSDL)

WSDL is an XML based document for describing services provided by a Web Service as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information [26]. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints. WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate.

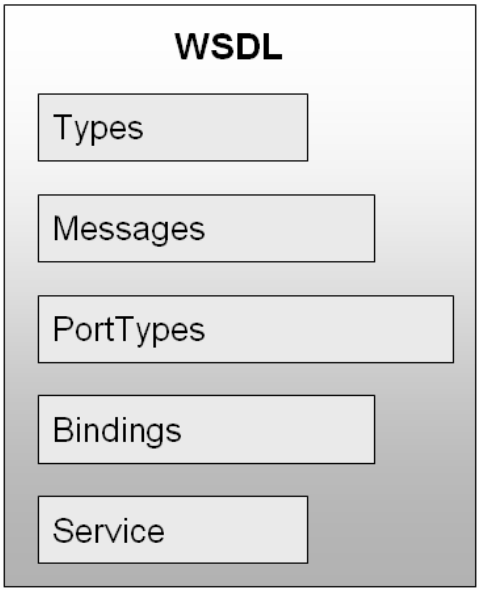

Figure 5.2: Web Services Description Language

WSDL defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitute a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Hence, a WSDL document uses the following elements in the definition of network services:

- Types - a container for data type definitions using some type system (such as XSD).
- Message - an abstract, typed definition of the data being communicated.
- Operation - an abstract description of an action supported by the service.
- Port Type - an abstract set of operations supported by one or more endpoints.
- Binding - a concrete protocol and data format specification for a particular port type.
- Port - a single endpoint defined as a combination of a binding and a network address.
- Service - a collection of related endpoints.

5.2.2 Directory Facilitator Agent Description (DF-Agent-Description)

A comprehensive support in Directory Facilitator is available in FIPA Multi Agent Systems to describe and search services provided by an agent. DFAgentDescription [27] is the type of object which is used to populate with

information of an agent and given to Directory Facilitator to search a match for it or to register it. Agent-Description is the section which contains information about agent exclusively. AgentID is the identified of agent through which it can be distinguished from that of other agents. It also contains the list of ontologies, interaction protocols for negotiation and content languages supported by an agent. Lease time is the time duration for which the description is valid. Information in scope variable deals with federation of different Directory Facilitators to provide required information.

Apart from describing details of agents, details of each service provided by the agent are also stored. It includes name and type of the service, list of ontologies, interaction protocols and content languages specific to the services. Since agents have capability to be social to form virtual organizations. It is possible that an agent describes some of the services in its Directory Facilitator Agent Description which are actually provided and owned by other agents, Ownership attribute mentions information in this regard.

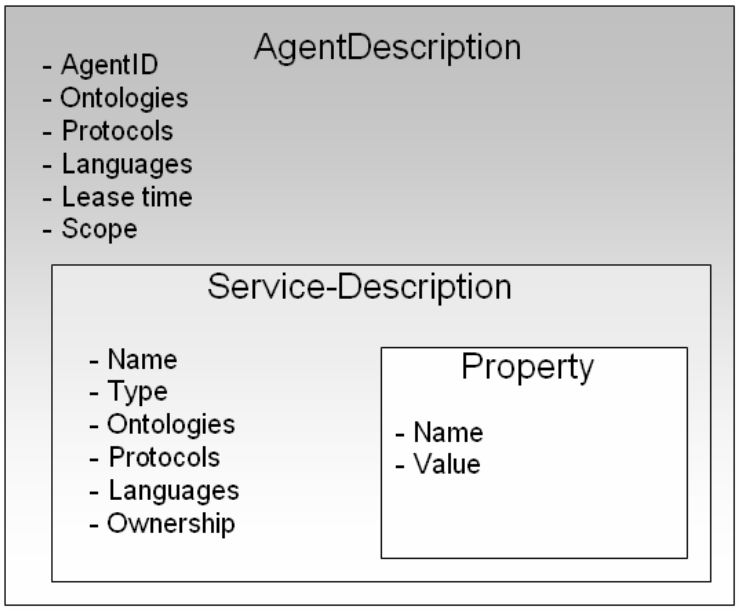

Figure 5.3: Directory Facilitator Agent Description

Each service has some properties which are used by agents to interact with. Properties in Service-Description of a service are quite similar to input arguments and return type. Each property has a name and term with it. Term is explained in the ontology of the agent. If the term of property belongs to AgentAction schema of ontology, it is similar to input arguments that are provided to an agent to perform some task. If term of property belongs to predicate schema of ontology, it is the return of the agent once it is required to perform some task.

5.3 COMMUNICATION PROTOCOLS

In service oriented architecture, service consumers acquire services provided by service providers using communication protocols. Typically it is called as service invocation. Communication Protocol represents what is sent during a given exchange, the combination of the data, the envelope and all other metadata necessary for the successful transmission of a message e.g. Message envelope, Message Exchange, Business Transactions, Digital Signature etc. This section provides concise information about communication protocols of both technologies i.e. Agent Communication Language (ACL) for Multi Agent Systems and Simple Object Access Protocol (SOAP) for Web Services.

5.3.1 Simple Object Access Protocol (SOAP)

SOAP is a simple XML based protocol to let applications exchange information over HTTP. SOAP is a communication protocol for accessing a Web Service. It stands for Simple Object Access Protocol. It is used for communication between applications and provides a format for sending messages via Internet. SOAP is platform independent, language independent as based on XML. It is simple and

extensible. It allows you to get around firewalls. SOAP will be developed as a W3C standard.

It is important for application development to allow Internet communication between programs. Today's applications communicate using Remote Procedure Calls (RPC) between objects like DCOM and CORBA, but HTTP was not designed for this. RPC represents a compatibility and security problem; firewalls and proxy servers will normally block this kind of traffic.

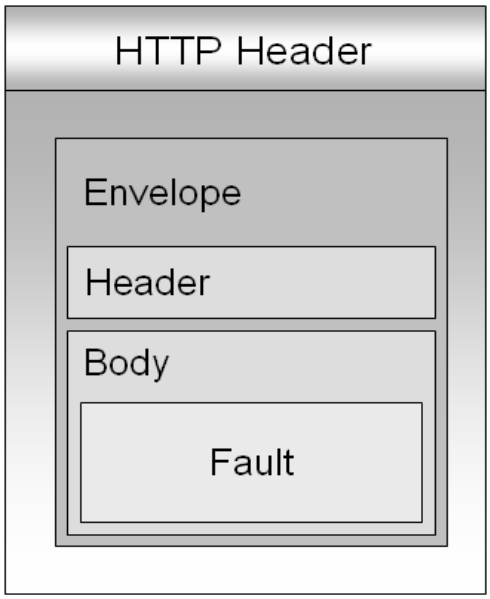

Figure 5.4: Simple Object Access Protocol

A better way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this. SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages. A SOAP message is an ordinary XML document containing the following elements:

- A required Envelope element that identifies the XML document as a SOAP message
- An optional Header element that contains header information
- A required Body element that contains call and response information
- An optional Fault element that provides information about errors that occurred while processing the message

5.3.2 Agent Communication Language (ACL)

Autonomous behavior of an agent depicts the fact that an agent has some inbuilt intelligence by using which it decides for itself its actions based upon what it senses from its environment. The other aspect of being autonomous is the fact that an agent's decisions can't be directly controlled by another party. This translates to the fact that an agents functions can't be invoked directly by a 3rd party, both theoretically. So, if an agent's functions can't be invoked by another agent/entity, then how can the two interact? How can be an agent influenced from, take input from another agent/entity or give its output to another agent/entity.

The solution to this problem is Agent Communication Language (ACL). The ACL package would be responsible for creation of a message that's 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 ACL Message.

Agent Communication Languages provides agents with a means of exchanging information and knowledge, which is really the essence of all forms of

interaction in multi-agent systems. The result of which, was the FIPA ACL. ACL is an outer language that specifies message format and includes 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 SL0 in 1997 and included basic semantics. After this SL1 and SL2 is published and is based on SL0 and provides rich semantics. Every agent has common semantics to talk each other that is known as shared ontology.

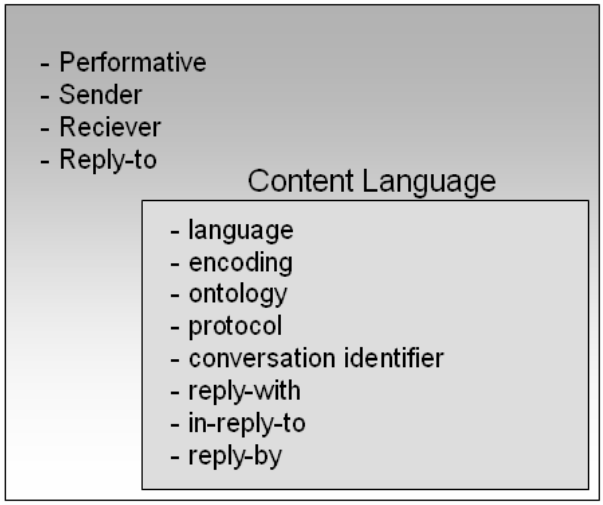

Figure 5.5: Agent Communication Language

Semantic analysis (Lexical analyzer) deals with semantic analysis and is actually responsible for the analyzing the raw message and identifying the tokens that are basically from the FIPA ACL Message format.

Message content creation of ACL Message deals with the content and ontology of the ACL Message, content field that is a part of the ACL Message format from FIPA Specification. This part will be responsible for the creation of the content based on the rules specified by Semantic language syntax by FIPA.

Message creation is responsible for the encapsulating of the different parts of the ACL message as mentioned in the FIPA- ACL Message format Specifications.

No communication can take place without a shared, unambiguous and negotiated vocabulary. There are multiple issues while considering vocabulary issues apart from the language barrier. By definition, ontology is a controlled, hierarchical vocabulary for describing a knowledge system with shared semantics. There will be a separate module for sharing a well negotiated and unambiguous ontology. There will be a separate module to deal with ontological issues. Conflicts like the word “hornet” having different meaning for an airlines person and a biologist will be removed.

Chapter 6

PROPOSED ARCITECTURE

Open systems like Grid are capable of dynamically changing and hence the resources are not known in advance, rather they can change over time and are highly heterogeneous [15]. It gives rise to the need of technologies that support negotiation or cooperation, in which Multi Agent Systems have proven to be. The entities in Grid environment should not just act as a dumb receptor of task descriptions, but should cooperate with the user and with other entities to achieve their goal. Hence there is need of agents acting as expert assistants or delegate with respect to some services, knowledgeable about both the Grid environment and the requirements of user in the form of an agenda of tasks and capable of negotiating with other agents (owner of different services), establishing contracts, performing service composition for achieve user's goals.

In Figure 1, proposed architecture is shown. The key idea is how autonomous software agents can help different entities like service registry, service provider and service consumer of existing service oriented principles based Grid computing environment in bringing semantic interoperability, negotiation and contracts. Another observable point of this architecture is that it is not changing any specifications and implementations of both the technologies. Grid computing is still focusing on hiding resource heterogeneity and providing a scalable robust infrastructure while software agents as autonomous entitles acts as owner to these services to semantically interoperate, negotiate with each other.

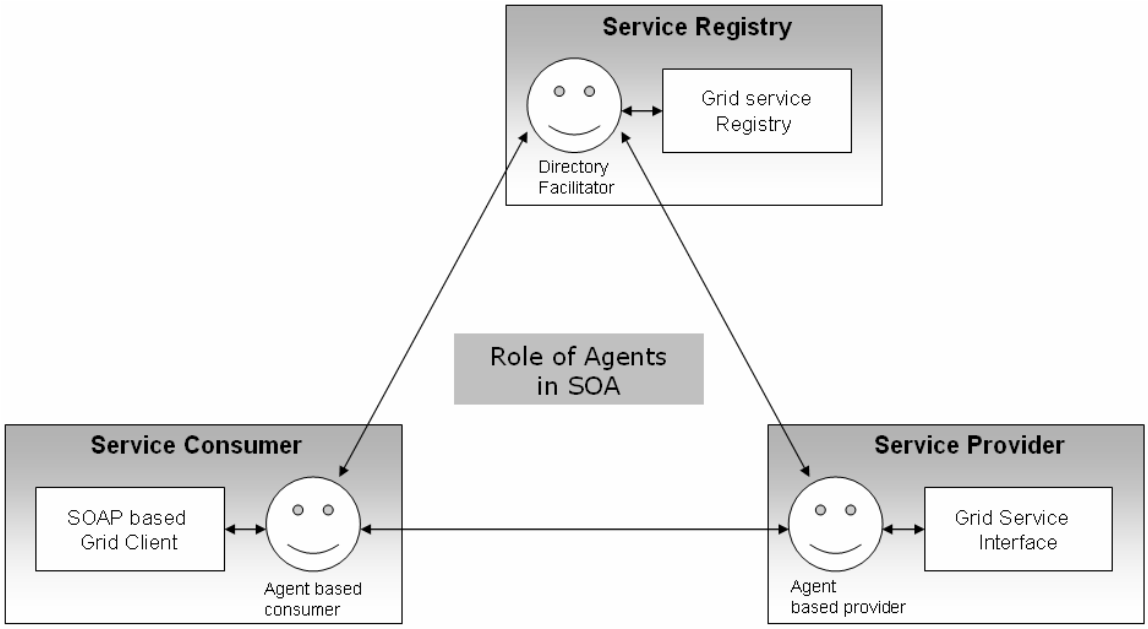

Figure 6.1: Role of Software Agents in Grid computing

Major issues in designing the proposed system are how agents can search for services in UDDI, how agents publish their services in UDDI, how agents can communicate with Grid services, how agents can understand a Grid service WSDL, how Grid clients can search for services in Agent platform (e.g. Directory Facilitator) and how Grid clients can communicate with agents to obtain services. All the issues are technically challenging because agents have their own mechanism of service description which is semantically rich with ontologies, negotiation languages and interaction protocols where as Grid services descriptions are based on Web Services Description Language (WSDL) which is not semantically rich, rather very simple. Secondly, both the Grid and Agents use different communication protocols for communication. Rest of the papers presents solutions to above mentioned challenges.

This section describes the detailed design of proposed system in which the most important technical challenges are solved, i.e. without changing any specification and implementation of Grid and Agents, enabling two way Service Discovery, Service Publishing (Service Description Transformation) and communication protocol conversion among Software Agents and Grid.

6.1 INITIAL PROXY BASED ARCHITECTURE

In the beginning of implementation phase, consuming a Web service by an agent client was the objective but there exists an inherent gap between the two. Web services use SOAP (Simple Object Application Protocol) as a communication protocol while agents send ACL (Agents Communication Language) messages for communication. Web services use WSDL (Web services Description Language) as service description language while, Agents use ontologies as service description. We followed the approach in which a wrapper in the form of Web service proxy agent is generated for a Web service. The wrapper handles all the protocols, parameter and service description transformations required. This proxy agent is generated by inspecting the WSDL document provided by the Web service. The ontologies of the proxy agent provide the same functionality as the one described in the WSDL files of the corresponding Web-service. Web service proxy agent can then run under JADE or any other agent platform to map ACL messages from client into SOAP calls to the invoked Web service.

We have developed a tool, which can generate a wrapper/ proxy agent (for JADE Platform) for an existing web service described by a Web Service Description Language (WSDL) file. Consequently it is then possible to call web services indirectly within an agent environment. This proxy agent accepts client agent

requests in ACL message, calls the actual web service by sending SOAP messages, and sends the results (returned by web service in form of SOAP) back to the client agents in form of ACL message.

The approach, which we followed to realize this solution, was to embed the Web Services client into Agent so that the Agent could be able to send and receive SOAP request as well. In order to generate the code of proxy agent, this tool first analyzes the WSDL file describing the web service (binding information, internet address, operations and input/output parameters), and then generates ontology codes (request, response classes containing operation names and parameters) and agent codes (Jade agent accepting requests, operation dispatching, web service call).

The tool we have developed to do all this has 2 components. First one is WSDL Parser that explores the description given in the WSDL file of a Web service to which client agent needs to communicate. The WSDL Parser component extracts information of operations of the Web service, operation name, parameters, return and address along with binding information.

All the information extracted by the Parser is then sent to another component named as Translator. The Translator gets all the information and maps it into an agent code. The agent code contains following details:

1. Agent code file, which contains SOAP client code in the basic class that is extended from Agent.
2. Ontology code, which helps Proxy agent understand the Web service operations.
3. Error Predicate, A common predicate ontology element which indicates web service invocation faults.

4. For each operation, a directory named same as operation name is created. Each directory further contains three files.

   i. Agent Action file, is agent action ontology element to invoke the corresponding operation of Web service.

   ii. Predicate, an ontology element which contains Web service corresponding operation results after the operation is invoked.

   iii. Ontology code file, having same name as corresponding operation name.

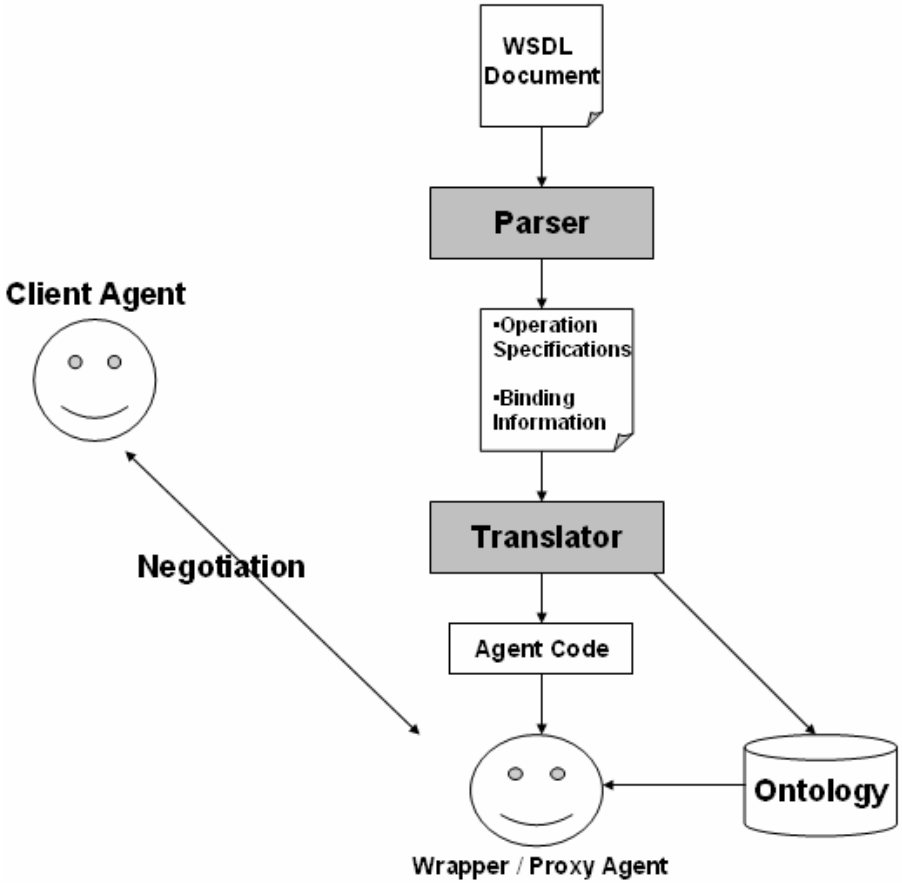

Figure 6.2: Detailed design of proposed solution

To generate the code of the wrapper agent from WSDL, the following steps are required:

1. The operation specification of Web service described in the WSDL file is to be explored. For the input and output messages of each operation the related AgentAction and Predicate classes must be created with fields corresponding to the input and output parameters.
2. The above mentioned classes must be registered by the agent as it's the ontology.
3. Agent code along with SOAP client must be generated to be able to receive ACL message and then dispatch it to appropriate operation in the form of a SOAP call by decoding input parameters, calling the Web Service, then encode the SOAP message response by Web service into an ACL message which is then sent back by the Proxy agent to Client Agent. The transformations from SOAP to ACL and ACL to SOAP have a template, the agent code and the ontology for all Web Service invocations can be generated from the WSDL automatically.

6.1.2 Test bed

We tested the tool using three machines with a real life scenario by invoking Web service, description is given as follows.

- On one machine, a Web service was deployed on Tomcat Web server having Apache Axis.
- On second machine, the above mentioned tool was deployed.
- On third machine, a client agent was deployed on JADE, which could send an ACL request message to the Proxy Agent.

6.1.3 Working

- The Client Agent sent an ACL message in which Web service name was mentioned.
- The tool as soon as received the ACL Message, extracted out the name of Web service and got WSDL file of the Web service was provided to WSDL Parser. The WSDL Parser which extracted all the necessary information of Web service. The extracted information was used by Translator which translated the information into a JADE agent code called as Proxy Agent.
- The Client Agent was notified of this Proxy Agent. The Proxy Agent then shared the ontology with client agent so that the client agent could be able to understand service provided by Proxy Agent and intern the Web Service.
- The client agent sent the ACL request message having parameters for the service to be used.
- Proxy agent after reviving the ACL message converted into SOAP request message and sent to the Web service. It then received a SOAP response message in return by the Web service.

The SOAP response message was then sent back by the Proxy agent to Client agent in the form of ACL message.

6.2 SOFTWARE AGENT INTERACTION WITH WEB SERVICE

In this section, the shown detailed design enables FIPA compliant Software Agents interact with W3C compliant SOAP based Grid computing environment entities including Grid services and Grid clients by performing Service Discovery in UDDI, understanding Grid Service and invoking a Grid Service.

6.2.1 Agent performing service discovery in UDDI

A middleware is designed that makes services visible to Software Agents. Whenever a Software Agent searches some service, it performs lookup for the Agent in Directory Facilitator (DF) of Multi Agent System. If DF does not have the required agent registered, it redirects its search to the middleware by sending an ACL (Agent Communication Language) message. As soon as the middleware input interface receives message, it passes it to protocol converter that extracts out the service name and passes it a component named as ACL2SOAP converter in the middleware. It performs transformation of the ACL based search query into SOAP based UDDI search query and forwards to UDDI where the required service is expected.

A search is performed in UDDI and if required service is found, a message is returned by the UDDI as SOAP based search query response to the middleware where the component SOAP2ACL converts that result into valid ACL based search response message and sends back to DF. The DF further forwards the message to Software Agent that requested for search. In this way, the middleware has helped a Software Agent to search for the services in UDDI with an illusion that it is searching Agent services in DF of Agent Platform. Whole description can be visualized from fig 2.

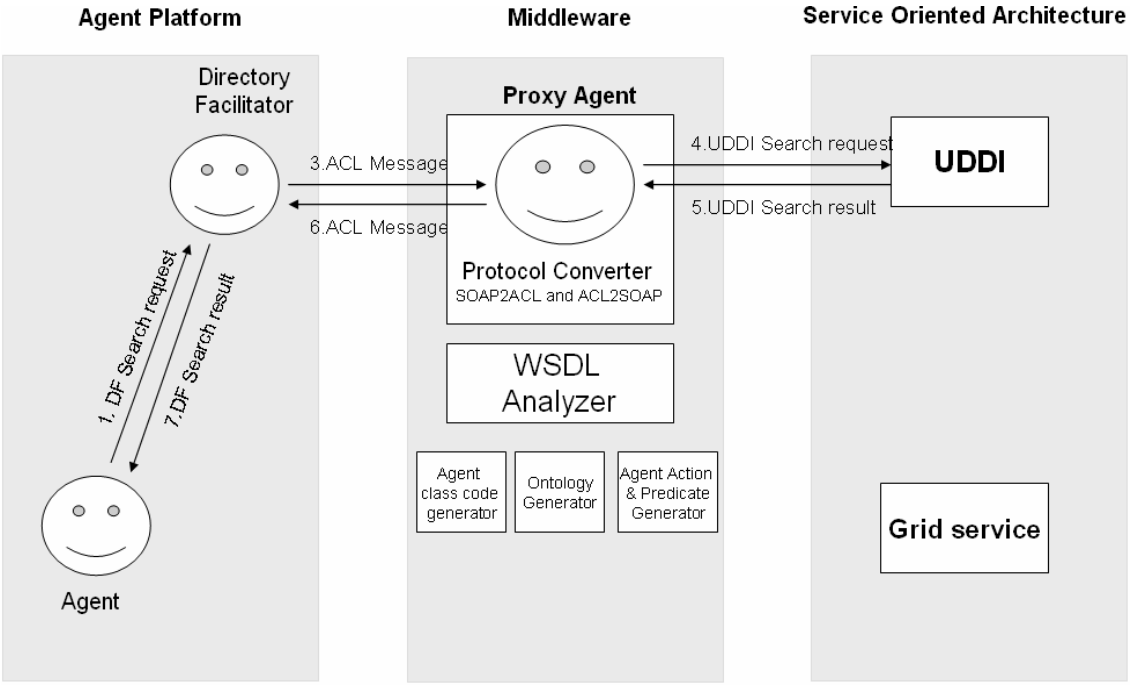

Figure 6.3: Software Agent searching for required service in UDDI

6.2.2 Agent understanding a Web Service

In previous section, the Software Agent has come to know about the existence and address of the required service and now the agent is required to consume the service. In order to consume the service, Software Agent is needed to know about the Ontology, AgentAction Schema, Predicate Schema and Concept Schema etc. On the other hand Middleware has the address for Services Description Language (WSDL) file.

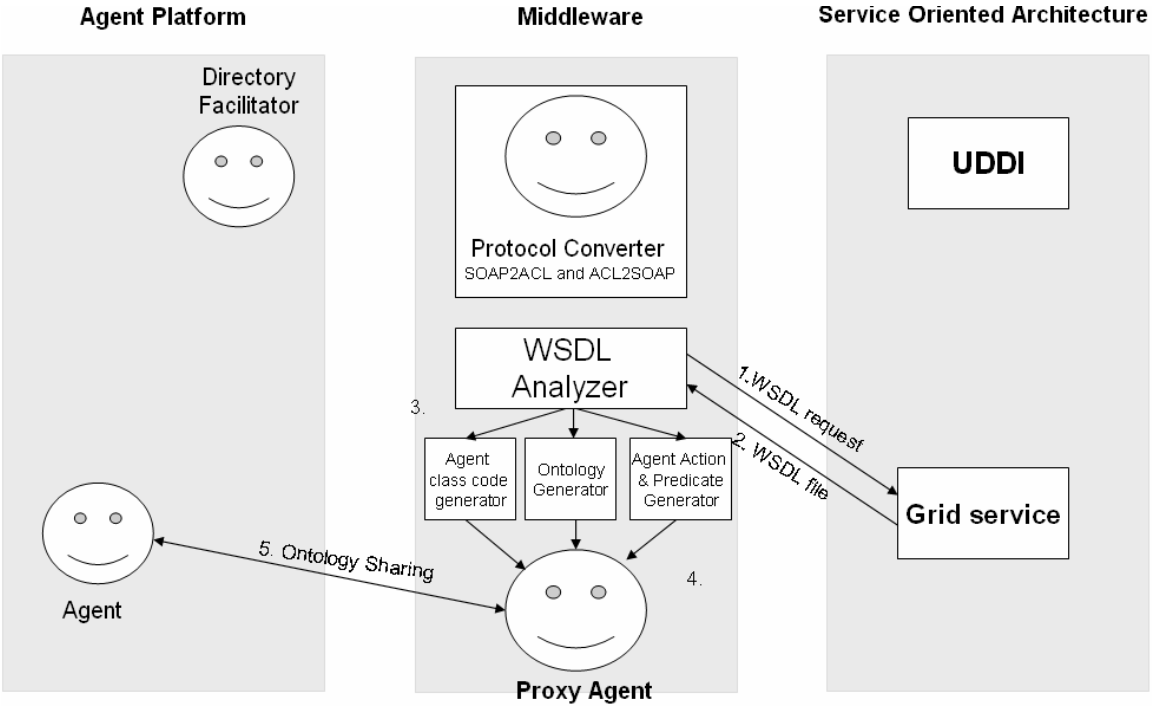

Figure 6.4: Agent understanding WSDL

WSDL Analyzer, a component as obvious from name gets the WSDL file of required Web Service and analyzes portTypes, SOAP bindings for service and extracts out useful information from it. This extracted information is then passed to three components which participate in generating the code for Proxy Agent. First of all, Agent class code is generated in contains necessary behaviors. Second component generates ontology file for Proxy Agent on the basis for portTypes of WSDL file of actual web service. The third component generates necessary AgentAction and Predicate Schema based on information from Web Service. In this way, complete for Proxy Agent along with its ontology is generated.

Translation of WSDL of services is performed into a form (Proxy Agent) that Software Agents can understand. Agent shares the newly generated ontology of Proxy Agent with itself with an illusion that the Proxy Agent is providing

the required service. In this way we have made the Web Service description published in Agent Platform in order to make it understandable for Software Agents. Fig 3 explains whole scenario.

6.2.3 Agent invoking a Web Service

In previous sections, middleware has helped the Software Agent to search and understands the services. Now the Software Agent is ready to consume the service. Software Agents shares ontology with Proxy Agent (which was generated in previous step) with an illusion that Proxy Agent is providing the required services. After sharing ontology, Software Agent sends an ACL request message (according to the shared ontology with Proxy Agent) having input parameters to middleware. Input interface receives the message and passes it to ACL2SOAP protocol converter. This converter extracts out the input parameters from ACL request message and creates an equivalent SOAP message.

The SOAP client at middleware is directed to send the generated SOAP request message is sent to the Web Service at remote Web Server providing required services. The Service after receiving SOAP request message processes the input parameters and then returns the output in the form of an SOAP response message to the SOAP client at middleware which upon receiving the SOAP response message passes it to SOAP2ACL protocol converter which extracts outputs from SOAP message and generates a ACL response message as shown in fig 4. The generated ACL message is then sent to the Software Agent. In this way, the middleware helps the Software Agent search, understand and consume Web Services.

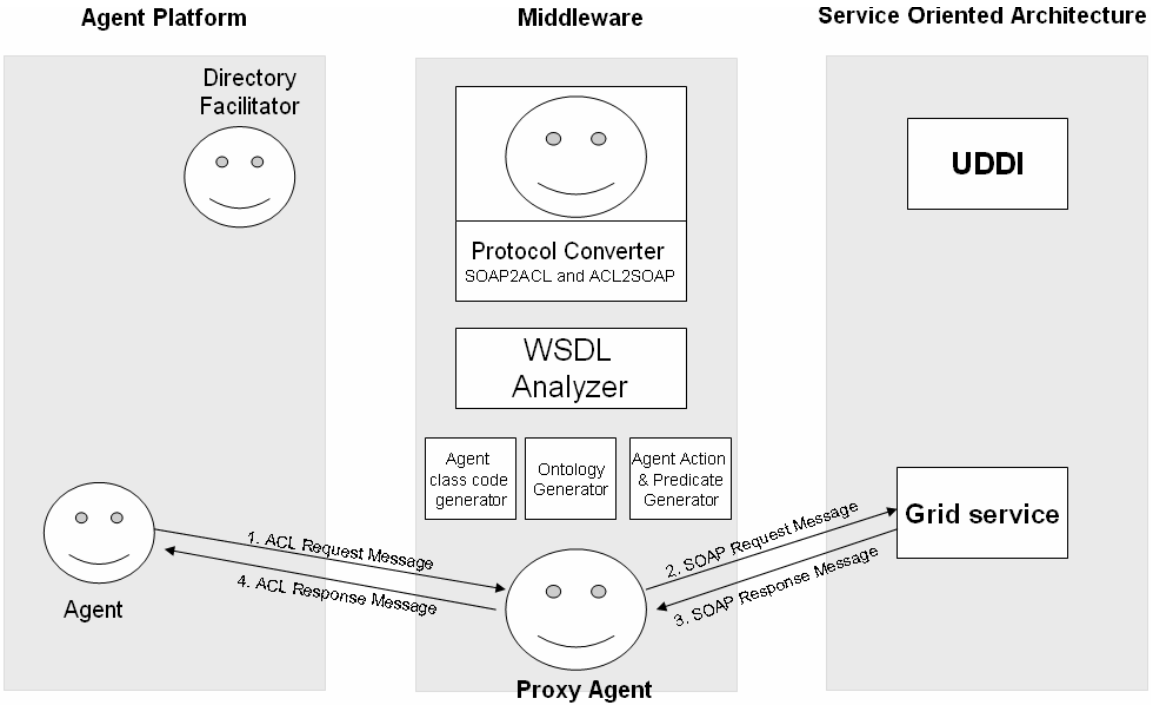

Figure 6.5: Software Agent invoking a service

6.3 WEB SERVICES INTERACTION WITH SOFTWARE AGENTS

In this section, the shown detailed design enables W3C compliant SOAP based Grid computing environment entities including Grid services and Grid clients interact with FIPA compliant Software Agents by performing Service Discovery in DF, understanding services provided by a Software Agent and getting services from Software Agents.

6.3.1 Web Service client performing service discovery in DF

Whenever SOAP based Grid client needs some service, it performs lookup for the service in UDDI, if the UDDI doesn't have the required service, it redirects its search to the middleware by sending a simple SOAP based UDDI search request message. As soon as the middleware input interface receives message, it passes it to protocol converter that extracts out the service name and passes it to Proxy Agent

present at middleware. This Proxy Agent creates and sends a valid ACL based DF search request message to remote agent platform from where the required service is expected.

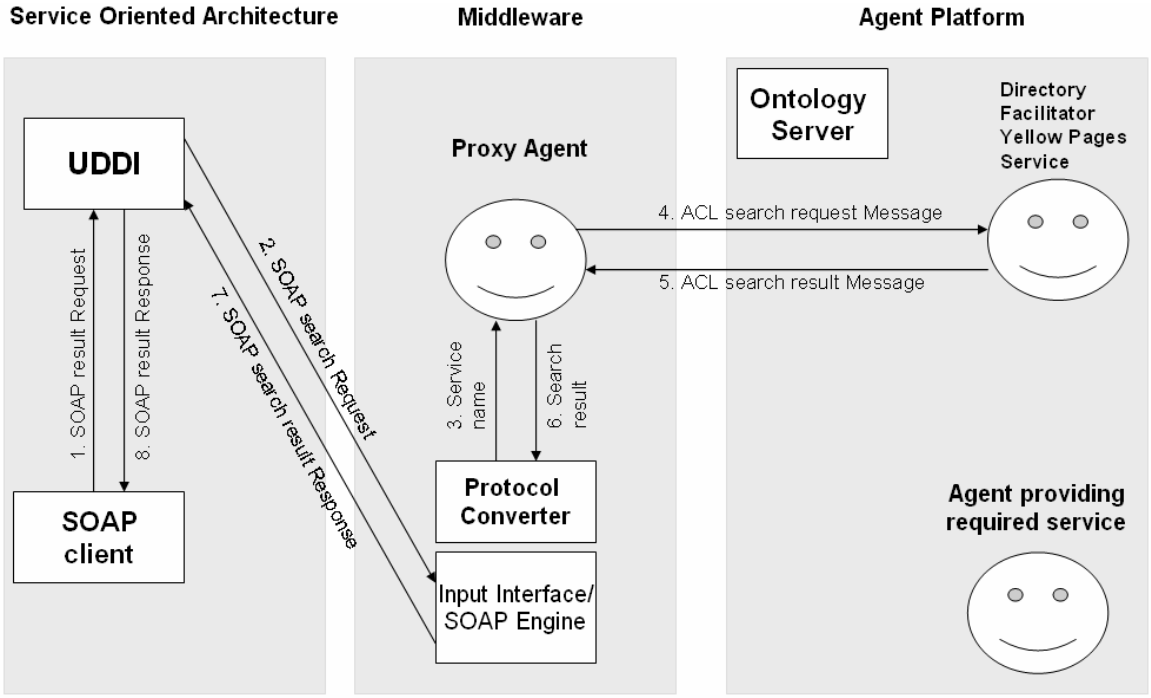

Figure 6.6: Grid client searching for required service in Agent Platform

Directory Facilitator of the remote Agent Platform performs a search. If required service is found, a message is returned by the DF of that remote Agent Platform to the agent at our middleware which is further passed to the ACL2SOAP protocol converter.

This converter transforms ACL based DF search response back to SOAP based UDDI search response and sends to the UDDI lookup service which further forwards message to the Web Service client requested for the search as shown in fig 5. In this way, the middleware helps the Web Service client to search for the services at Agent Platform. The SOAP response message contains the address of the middleware

which means that Web Service client is given an illusion that the required service is available as Grid Service at middleware.

6.3.2 Web Service client understanding service provided by a Software Agent

Up-till now, Grid client has come to know about the existence and address of the required agent and now the client is required to consume the service. In order to consume the service, the client is needed to know about the functions, input parameters and outputs of the service. Since the client has already got the address of the service from the previously received SOAP response message that is why, this time client send request message to middleware to get the Agent service published in UDDI. Middleware translates the Agent service into WSDL. How this translation will occur at middleware is given below:

The SOAP request message of Web Service client receives by the SOAP Engine. The service name for which the WSDL file is required is taken out and is passed to an Agent. This Agent then contacts to ontology server of the remote Agent Platform in order to share the ontology used by the Agent at remote platform providing the required services as shown in fig 6.

Agent at middleware gets the ontology of the Agent of remote platform shared by the ontology server and informs the ontology code generator which is present at the middleware. The ontology code generator gets instance of the shared ontology and using reflection, generates the equivalent java code of the ontology instance. This generated java code of ontology class contains information about methods, input parameters and outputs provided by the agent providing the required services in order to consume the service.

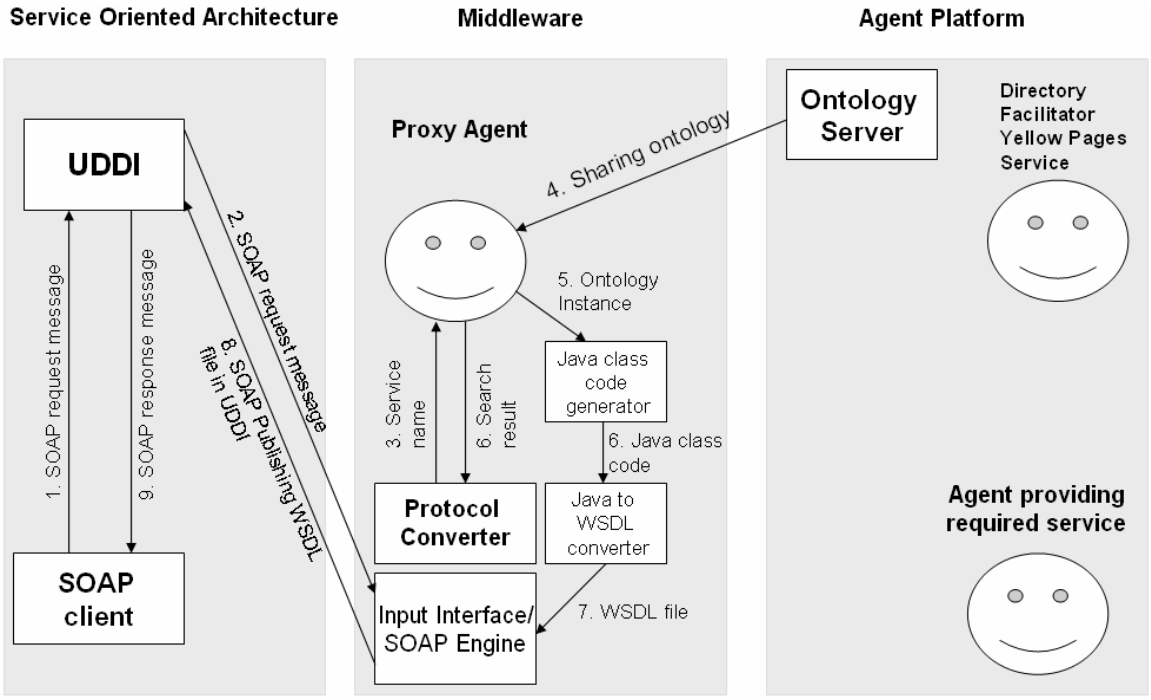

Figure 6.7: Agent publishing its services in UDDI to make it visible for Web Service clients

The java code is passed to Java to WSDL converter in order to translate the interface code into a WSDL to make it understandable for Web Service clients. WSDL file is generated and sent to Web Service client and may be used for preparation of SOAP requests. Same WSDL file can be published in UDDI which will make the Agent, publish it services in Web Services world to make it understandable for Web Service clients. Figure 6 explains whole scenario.

6.3.3 Web Service client accessing an Agent

Up till now, the middleware has helped the Web Service client to search and understand the services provided by Agents. Now the Web Service client is ready to consume the services provided by the Agent. This time Web Service client communicates with the middleware with an illusion that it is the required Web Service.

The client generates a SOAP request message (according to the service description which it got in WSDL) having input parameters.

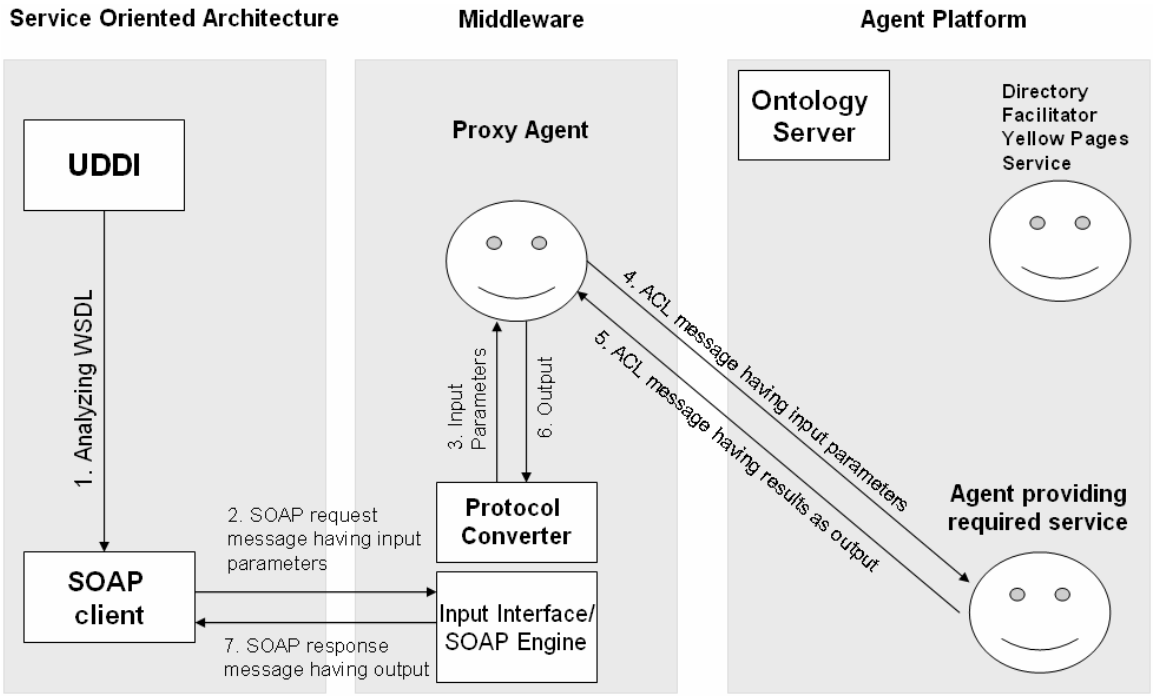

Figure 6.8: Web Service client consuming services provided by Agent

This SOAP request message is sent to the middleware. Input interface receives the message and passes it to SOAP2ACL protocol converter. This converter extracts out the input parameters from SOAP input message and creates an equivalent ACL message. The Agent at middleware is directed to send the generated ACL request message is sent to the Agent at remote platform providing required services.

The Agent after receiving ACL request message processes the input parameters and then returns the output in the form of an ACL response message to the Agent at middleware. The Agent at middleware upon receiving the ACL response message passes it to ACL2SOAP protocol converter which extracts outputs from ACL message and generates a SOAP response message. The SOAP response message is

finally sent to the Web Service client as shown in fig 7. In this way, the middleware helps the Web Service client search, understand and consume services provided by Software Agents.

Chapter 7

COMPARITIVE ANALYSIS OF TECHNOLOGIES

This section presents some low level details about how exactly conversions for service discovery, service description transformation and communication protocol conversion are takes place between Agents and Web Services Framework and vice versa. In order to understand this, one must understand what are the similarities and dissimilarities are among the specifications of both technologies. Here we present the comparisons of specifications of both the technologies Web Services Framework and Multi Agent Systems.

7.1 SERVICE REGISTRATION AND DISCOVERY

In Web Services Framework, clients perform service discovery in registry called UDDI whereas in Multi Agent Systems, Directory Facilitator is used as yellow pages for service discovery. In both the technologies there is a difference between the way information is stored in service registry and discovery is performed.

Table 7.1: Comparison of UDDI and Directory Facilitator

Universal Description Discovery and Integration (UDDI)	**Directory Facilitator (DF)**
BusinessEntity Name	AgentID
Contact	Ontology (Concept schema)
Contact_name	Ontology (Concept schema)
Contact_phone	Ontology (Concept schema)
Contact_email	Ontology (Concept schema)
Contact_address	Ontology (Concept schema)

BusinessService_Name	Service-Description_name
serviceCategory	Service-Description_type
serviceParameter	Service-Description_Property
Input	Property {Ontology (Action Schema)}
Output	Property {Ontology (Predicate Schema)}

UDDI stores service description language i.e. WSDL along with some other business related information. In Multi Agent Systems, Directory Facilitator stores only the service description language i.e. DFAgentDescription and no other information since it is quite comprehensive. The way of service discovery is also different in both cases. In Web Services Framework while service discovery, a web service client invokes some search related methods which are specific to WSDL related information or the other information stored. In case of Multi Agent Systems, and Agent for service discovery first fills the same object used for service description named as DFAgentDescription according to required service parameters. The search request is then received by DF where it compares the sent DFAgentDescription with the all that are already stored.

7.2 SERVICE DESCRIPTION LANGUAGES

Transformations of service description languages are explained here which take place from Agents to Web Services and vice versa. In FIPA compliant Multi Agent Systems, Agents describe services in DFAgentDescription. Where as Web Services are described in WSDL (Web Service Description Language).

Table 7.2: Comparison of WSDL and DFAgentDescription

WSDL	**DFAgentDescription**
Service-End point	AgentID
portTypes	Services
Complex Types	Ontology (Concept Schema)
Messages – input	Ontology (AgentAction schema)
Messages – output	Ontology (Predicate schema)
Binding	None
None	Interaction protocols
None	Content Languages
None	Lease time
None	Scope

If we precisely observe the above table, it compares the attributes in description language specifications of both technologies. In WSDL, Service-End point provides a direct URL to access where the web service is actually described where as in DFAgentDescription, AgentID is a unique identifier which is assigned to an Agent on its creating by Agent Management System (AMS) of the Agent Platform. Web Service supports one or more operations which are specified in portType section of WSDL, on the other hand DFAgentDescription has list of Service-Description which is just like operations provided by Web Services. If some operation of Web Service requires complex type or user defined object, the definition the user defined object is specified in 'Complex Type' section of WSDL where as in DFAgentDescription, there is attribute named ontology in Service-Description section. According to FIPA, Ontology is composed of three types of schemas named concept, agent-action and predicate

schema. Concept schema is used to define user defined objects in the ontology of an agent. In WSDL, each operation has some input message and may also have an output message. Information about this input and output messages would be used in 'Property' element of Service-Description. In case of WSDL, input and output messages are distinguished as parameters given to invoke a method and the result/return of the method respectively. In DFAgentDescription, the Property in case of input and output are distinguished from agent-action and predicate schema of the ontology specified in DFAgentDescription. Agent-Action schema contains the list of actions what the Agent can be asked to perform where as Predicate schema in the Agent's ontology contains list of outcomes/response of the Agent.

WSDL also contains the binding information for underline HTTP protocol for SOAP communication protocol for Web Services. In case of Agents, DFAgentDescription doesn't contain any binding information because an Agent requires AID of other Agent to communicate with. All the low level protocol details are dealt by another component named as Message Transport Service (MTS) of Agent Platform [25]. Negotiation among Agents in an Agent Platform is supported by Interaction Protocols. The attribute named protocol in DFAgentDescription contains the list of Interaction Protocols supported. In case of Web Services, a limited conversion is supported as Web Services Conversational Language (WSCL) but no information related to this is included in WSDL. DFAgentDescription also includes the set of content languages the Agent supports where as no such kind of information is supported in WSDL of Web Services. The validity of the service description of an Agent as DFAgentDescription in Directory Facilitator (DF) of an Agent Platform is

mentioned in ‘Lease time’ where as in case of WSDL no such kind of information is included. Finally, An agent can also restrict or allow its description to be explored by other agents by using ‘local’ or ‘global’ value in ‘scope’ in DFAgentDescription respectively.

7.3 COMMUNICATION PROTOCOLS

In this section, transformations of communication protocol are explained which take place from Agents to Web Services and from Web Services to Agents. In FIPA compliant Multi Agent Systems, Agents used Agent Communication Language (ACL) where as in Web Services; Simple Object Access Protocol (SOAP) is used as communication protocol.

Table 7.3: Comparison of SOAP and ACL

SOAP	**ACL**
Encoding style	Encoding
From (HTTP)	Sender
To (HTTP)	Receiver
Body	Content
Fault	Performative (Failure/Not-Understood)
None	Reply-to
None	Ontology
None	Protocol
None	conversation identifier
None	Reply-with
None	In-reply-to
None	reply-by

The table above provides a precise similarities and difference among communication protocol, Simple Object Access Protocol (SOAP) and Agent Communication Language (ACL) for Web Services Framework and Multi Agent Systems respectively.

SOAP includes the encoding details of message in it according to its specifications and same is the case with ACL. SOAP requires sender and received information in the underline HTTP protocol header instead of its own header where as in case of ACL, sender attribute contains AgentID of sender Agent of ACL and receiver attribute contains list of AgentID of recipients of the ACL message. The Body of SOAP message contains the actual content of the message i.e. which operation to invoke with the values of input parameters or the returned value/response after invocation of a Web Service. On the other hand, ACL is enriched with its content language e.g. FIPA SL 0/1/2/3. SOAP contains a section named 'Fault' in its body in inform the client about any errors/exception occurred at server side where as in case of ACL, it supports performative along with long list of communicative acts which not only can mention some failure but also many other functionalities as well. List of features of SOAP ends here but ACL doesn't. ACL is very comprehensive as compares 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. Ontology attribute contains name of the ontology which is used to give meanings to symbols used in message content. Protocol attribute indicates the name of Interaction Protocol for

negotiation being employed. There can be multiple negotiations going on among multiple Agents; conversation-identifier is used to identify individual conversation with multiple Agents. Responding Agent to reply this message uses reply-with attribute. In-reply-to parameter references an earlier action to which this message is a reply. Finally reply-by attribute specifies latest time by which the sending agent would like to receive a reply.

Chapter 8

DETAILED DESIGN

We have proposed AgentWeb Gateway for integration of Software Agents and Web Services. By integration we mean, enabling service discovery, service description transformation and service invocation among software agents and web services without disturbing the existing specifications of both. Major challenges are involved in this integration that both the technologies use different service registries, service description languages and communication protocols.

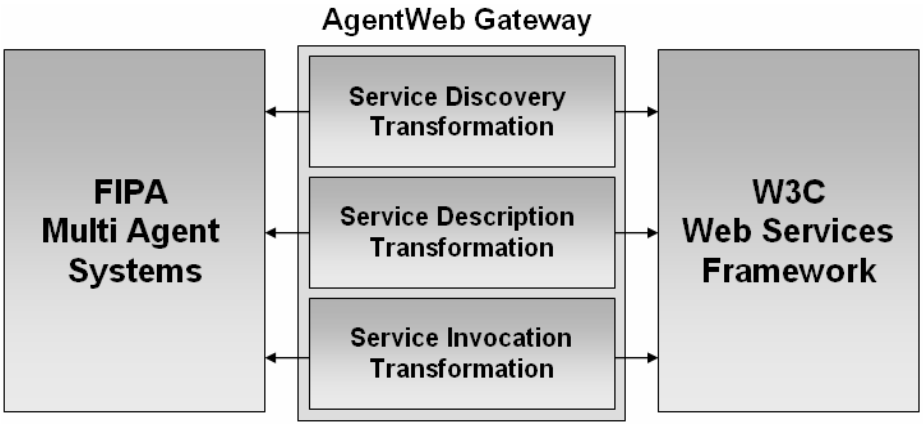

Figure 8.1: AgentWeb Gateway middleware

AgentWeb Gateway middleware provides solution for both the challenges by providing appropriate transformation mechanisms. The importance of this approach is that it enables integration of Software Agents and Web services without changing their existing specifications at the cost of time taken for translations which is negligible as compared to a transaction. In this paper we have presented a

detailed comparative analysis of service description languages of both. We also have proposed and implemented algorithm for required transformation.

According to ultimate Semantic Grid goals, Software Agents would be able to dynamically discover, compose, invoke and monitor web services. Software Agents and Multi Agent Systems specifications are governed by FIPA (Foundation of Intelligent Physical Agents) and specifications of Web Services are governed by W3C, hence there is a lot of difference among specifications of both technologies and hence Software Agents and Web Service cannot communicate with each other.

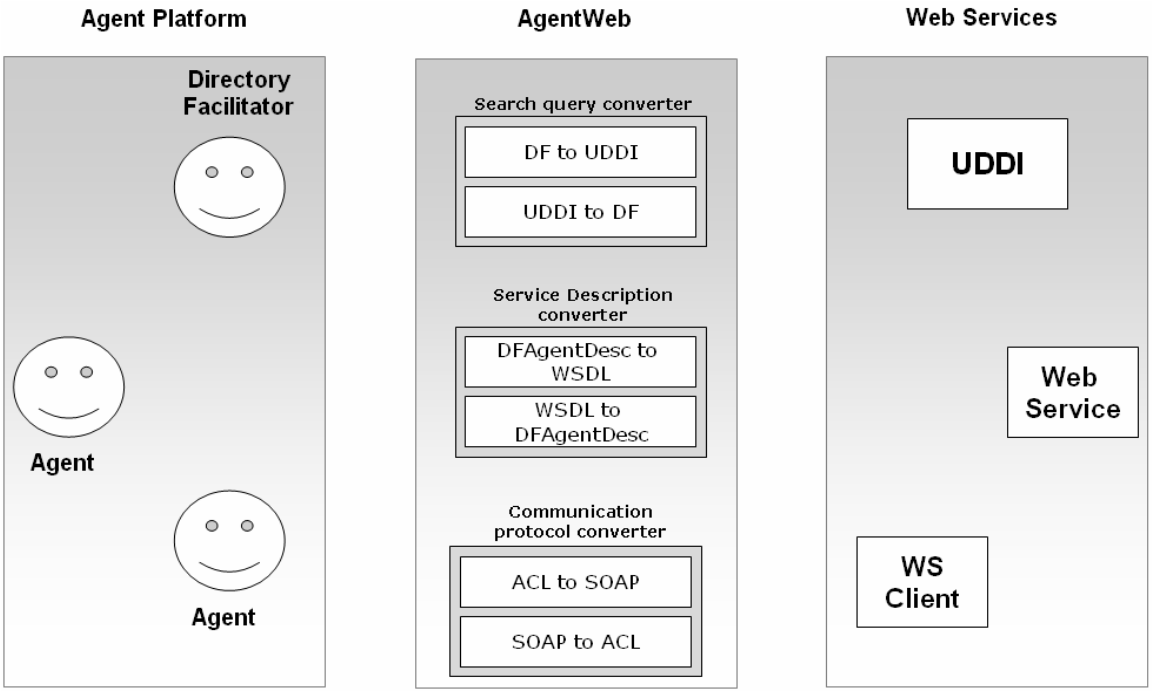

Figure 8.2: AgentWeb Gateway system architecture

We provide AgentWeb Gateway that acts as middleware between Multi Agent System and Web Services Framework and without changing existing specifications of both technologies. It provides Service Discovery transformation, Service Description transformation and Communication Protocol transformation.

Which means that using AgentWeb Gateway, without changing any specification of FIPA and W3C (agents and web services)

1. Software Agents can discover Web Services in Web Service registry (UDDI)
2. Software Agents can publish their services in Web Service registry (UDDI)
3. Software Agents can invoke Web Services
4. Web Service clients can discover Software Agents in Directory Facilitator (DF) of Agent Platform
5. Web Services can be published in Directory Facilitator (DF) of Agent Platform
6. Web Service clients can invoke Software Agents

This section describes the detailed design of proposed system in which the most important technical challenges are solved, i.e. without changing any specification and implementation of Grid and Agents by enabling two-way Service Discovery, Service Publishing and Service invocation among Software Agents and Web-Services.

8.1 SERVICE DISCOVERY CONVERTER

This section presents the details of first component of AgentWeb Gateway which is called as Service Discovery Converter. This component enables service discovery among Software Agents and Web services i.e. Software Agents can do service discovery in Web Services registry as Universal Description Discovery and Integration (UDDI) and Web Service clients can do service discovery in Multi Agent Systems service registry as Directory Facilitator (DF).

8.1.1 DF to UDDI search query conversion

Whenever a Software Agent searches some service, it performs lookup for the Agent in Directory Facilitator (DF) of Multi Agent System by sending required DF-Agent-Description. If DF does not have the required agent registered, it redirects its search to the middleware by sending an ACL (Agent Communication Language) message. As soon as the middleware input interface receives message, it passes it to DF-Agent-Description analyzer. It extracts out three major portions of information i.e. information about Agent that provides the required service, required service description and inputs and outputs of the services. The information about Agent providing required services is far waded to Business Entity builder where it is mapped to Business Entity for UDDI search query. Information about required service descriptions and its properties are forwarded to Business service builder where it is mapped to name and description of required service and its inputs and outputs. The generated business entity and business service is forwarded to UDDI search query builder.

A search is performed in UDDI and if required service is found, a message is returned by the UDDI as SOAP based search query response to the middleware which is converted back to a valid ACL based search response message and sends back to DF. The DF further forwards the message to Software Agent that requested for search. In this way, the middleware has helped a Software Agent to search for the services in UDDI with an illusion that it is searching Agent services in DF of Agent Platform. Whole description can be visualized from figure 7.

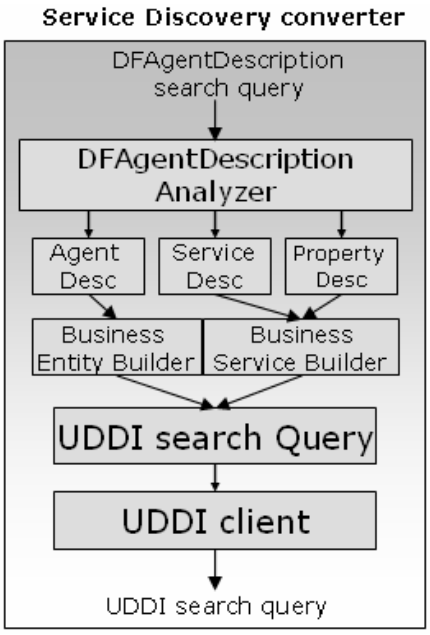

Figure 8.3: Software Agent searching for required service in UDDI

8.1.1.1 Algorithm for DF search query to UDDI search query conversion

get UDDI search query

generate Agent-Description

get Business-Entity

    get Business-Entity name and map name to AgentID

    get contact details to generate ontology (concept schema)

get Business–Service

    get Business-Service name and map to Service-Description name

get Category-Bag

    get serviceParameter and generate Property

    get input and map to Property (Ontology – Action schema)

    get output and map to Property (Ontology – Predicate schema)

add all generated Property objects into Service-Description

genereate DF-Agent-Description

    add Agent-Description and Service-Description into DF-Agent-Description

Enclose DF-Agent-Description into DF search query

Enclose DF search query into ACL message and send to DF

8.1.2 UDDI to DF search query conversion

Whenever SOAP based Web service client needs some service, it performs lookup for the service in UDDI, if the UDDI doesn't have the required service, it redirects its search to the middleware by sending a simple SOAP based UDDI search request message. As soon as the middleware input interface receives message, it passes it to UDDI search query analyzer. It extracts out information about business entity and business service. Information about business entity is sent to Agent description builder there business entity is mapped over information about Agent providing required service. Information about business service is forwarded to service description builder and property builder where service name and type is used for building service description and inputs and output parameters are used for building property. The generated Agent description, service description and property are forwarded to DF search query builder where DF-Agent-Description is generated and forwarded to DF of search. Directory Facilitator of the Agent Platform performs a search.

If required service is found, a message is returned by the DF of that remote Agent Platform to the agent at our middleware which transforms ACL based DF search response back to SOAP based UDDI search response and sends to the UDDI lookup service which further forwards message to the Web Service client requested for the search as shown in fig 5. In this way, the middleware helps the Web Service client to search for the services at Agent Platform. The SOAP response message contains the

address of the middleware which means that Web Service client is given an illusion that the required service is available as Web Service at middleware.

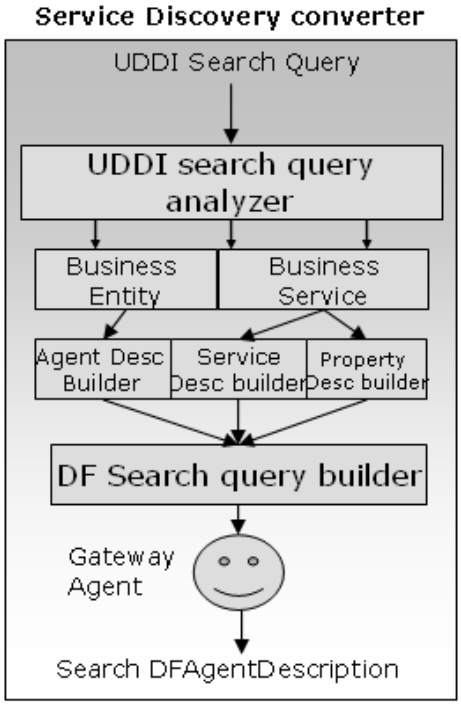

Figure 8.4: Grid client searching for required service in Agent Platform

8.1.2.1 Algorithm for DF search query to UDDI search query conversion

get DF search query

generate Business-Entity

get Agent-Description

    get AgentID and map to Business-Entity name

    get ontology and map to Business-Entity contact

get Service-Description

generate Business-Service

    get Service-Description name and map to Business-Service name

    get Property objects

    generate categoryBag

        map Property (ontology Action schema) to input

map Property (ontology Predicate schema) to output

add categoryBag into Business-Service

Add Business-Service into Business-Entity

Generate UDDI search query

Enclose Business-Entity into UDDI search query

Enclose UDDI search query into SOAP message and send to UDDI

8.2 SERVICE DESCRIPTION CONVERTER

This section presents the details of second component of AgentWeb Gateway which is called as Service Description Converter. This component enables service publishing among Software Agents and Web services i.e. Software Agents can publish services in Web Services registry as Universal Description Discovery and Integration (UDDI) and Web Services can be published in Multi Agent Systems service registry as Directory Facilitator (DF).

8.2.1 WSDL to DF-Agent-Description conversion

When a Software Agent comes to know about the existence and address of the required service and now the agent is required to consume the service. In order to consume the service, Software Agent is needed to know about the Ontology, AgentAction Schema, Predicate Schema and Concept Schema etc. On the other hand Middleware has the address for Services Description Language (WSDL) file. WSDL Analyzer, a component as obvious from name gets the WSDL file of required Web Service and analyzes portTypes, SOAP bindings for service and extracts out useful information from it. This extracted information is then passed further. For all complex types in the WSDL, Ontology (concept schema) is generated. Information about portType and binding is forwarded to DF-Agent-Description builder where the required

the final ‘Directory Facilitator Agent Description’ is generated and given to Gateway Agent which further send an ACL based publish request to Directory Facilitator.

Transformation is performed from WSDL of Web service into a form (DF-Agent-Description) that Software Agents can understand. In this way we have made the Web Service description published in Agent Platform in order to make it understandable for Software Agents. Figure 7 explains whole scenario.

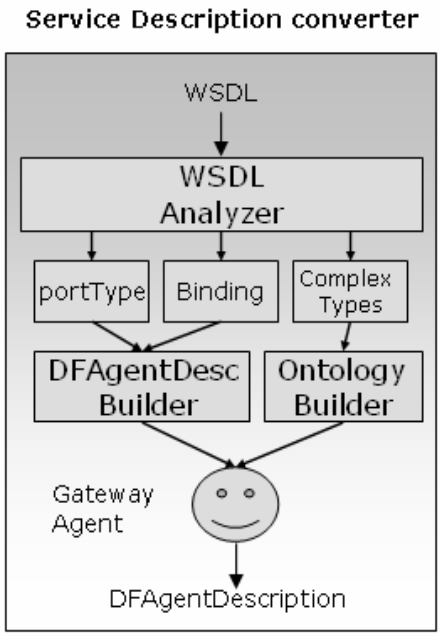

Figure 8.5: WSDL to DF-Agent-Description conversion: Agent understanding WSDL

8.2.1.1 Algorithm for WSDL to DF-Agent-Description conversion

In order to elaborate the design, given below is algorithm based on comparative analysis of WSDL and DF-Agent-Description for WSDL to DF-Agent-Description conversion in order to publish a Web Service in Directory Facilitator of Agent Platform.

get WSDL

get Service End point and map to Agent ID of AgentDescription

for all ComplexType, create Concept schema

each attribute of ComplexType as data member of Concept Schema class

for each operation in portType

get operation name and map to name of Service-Description

get names and types of inputs of operation to map to property and term mapped as AgentAction schema of ontology

if input element is a complex type then refer to corresponding concept schema

get name and type of output of operation to map to property and term mapped as Predicate schema of ontology

if output element is a complex type then refer to corresponding concept schema

indicate AgentAction and Predicate schema in Service-Description specific ontology

update ontology attribute in Service-Description with service specific ontology

add all properties in Service-Description

add all Concept, AgentAction and Predicate schemas in main ontology

update ontology variable of AgentDescription

add AgentDescription in DFAgentDescription

add all Service-Description in DFAgentDescription

return DFAgentDescription

8.2.2 DF-Agent-Description to WSDL conversion

This section explains that how a Software Agent publishes its services in Web Services registry UDDI. Information about services provided by an agent is stored as "Directory Facilitator Agent Description" (DF-Agnet-Description) ontology in Directory Facilitator which is transformed by Service Description converter into WSDL. The whole transformation process is given below:

First of all DF-Agent-Description ontology is analyzed and description about Agent i.e. AgentID is taken out and is mapped to Service-End-Point of WSDL to be built. There are one or more Service-Descriptions available in this ontology which

has information about the services of the agent. Name of each service is mapped to name of operation in portType of WSDL.

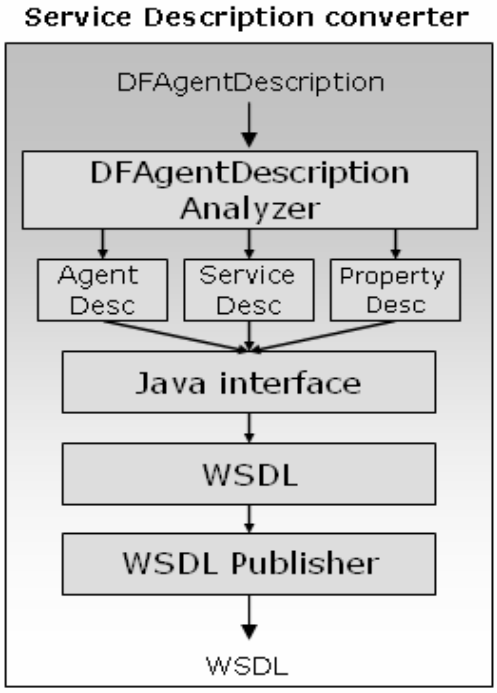

Figure 8.6: DF-Agent-Description to WSDL conversion: Agent publishing its services in UDDI

Each Service-Description of DF-Agent-Description has Property objects which indicate inputs and outputs of the corresponding services of the agent. Each Property object is checked. If it belongs to Predicate Schema (Predicate schema indicates propositions of an Agent) of Agent's ontology, it is treated as output of the corresponding operation of portType in WSDL. If the Property object belongs to Action schema of ontology (Action schema indicates the activities that can be carried out by an agent), it is then treated as input argument of the corresponding operation. After getting all the information about operation names, inputs and output, a java interface code is generated.

The java code is passed to Java to WSDL converter in order to translate the interface code into a WSDL to make it understandable for Web Service clients. WSDL file is generated and sent to Web Service client and may be used for preparation of SOAP requests. Same WSDL file can be published in UDDI which will make the Agent, publish it services in Web Services world to make it understandable for Web Service clients. In this way a Software Agents gets its services published in UDDI by transformation of its DF-Agent-Description ontology into WSDL by Service Description converter of Agent Web Gateway.

8.2.2.1 Algorithm for DF-Agent-Description to WSDL conversion

In order to elaborate the design, given below is algorithm based on comparative analysis of WSDL and DF-Agent-Description for DF-Agent-Description to WSDL conversion in order to publish an Agent based service in UDDI.

```
get DFAgentDescription
in AgentDescription
get Agent ID and map to Service End Point
for all concept schemas, generate Complex-Types
   indicate each data member of Concept Schema class as attribute of corresponding Complex-Type
for each Service-Description
   get service name and map to name of operation in portType
   get all properties
   for each property
      if property term belongs to AgentAction schema, map it to input arguments of operation
      if property term belongs to Predicate schema, map it to output of operation
add default SOAP binding information
generate and return WSDL
```

8.3 COMMUNICATION PROTOCOL CONVERTER

This section presents the details of third component of AgentWeb Gateway which is called as Communication Protocol Converter. This component enables service invocation among Software Agents and Web services i.e. Software Agents can invoke Web Services and Web Service clients can invoke Software Agents in Multi Agent Systems.

8.3.1 ACL to SOAP conversion

In previous sections, middleware has helped the Software Agent to search and understand services. Now the Software Agent is ready to consume the service. Software Agents gets the DF-Agent-Description ontology (which was generated in previous step) with an illusion that Gateway Agent is providing the required services. After getting ontology (DF-Agent-Description) from Agent, Software Agent sends an ACL request message having input parameters to the middleware.

Input interface receives the message and passes it to ACL2SOAP protocol converter. This converter extracts out the input parameters from ACL request message and creates an equivalent SOAP message. The SOAP client at middleware is directed to send the generated SOAP request message is sent to the Web Service at remote Web Server providing required services.

The Service after receiving SOAP request message processes the input parameters and then returns the output in the form of an SOAP response message to the SOAP client at middleware which upon receiving the SOAP response message passes it to SOAP2ACL protocol converter which extracts outputs from SOAP message and generates a ACL response message as shown in figure 8. The generated ACL message

is then sent to the Software Agent. In this way, the middleware helps the Software Agent search, understand and consume Web Services.

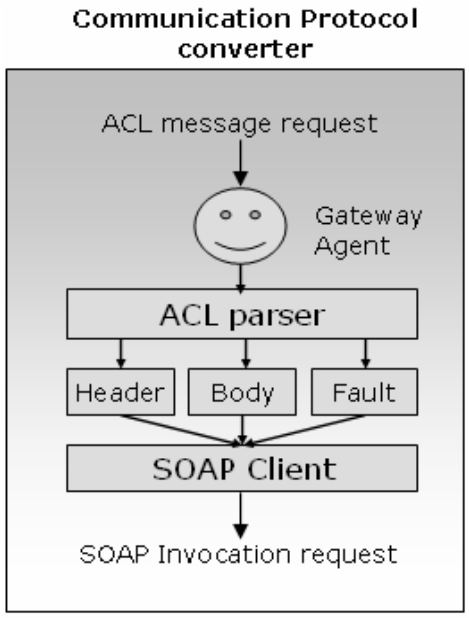

Figure 8.7: ACL to SOAP conversion: Software Agent invoking a service

8.3.1.1 Algorithm for ACL to SOAP conversion

In order to elaborate the design, given below is algorithm based on comparative analysis of SOAP and ACL for ACL to SOAP conversion in order for an Agent to invoke Web Service.

```
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
```

8.3.2 SOAP to ACL conversion

Up till now, the middleware has helped the Web Service client to search and understand the services provided by Agents. Now the Web Service client is ready to consume the services provided by the Agent. This time Web Service client communicates with the middleware with an illusion that it is the required Web Service.

The client generates a SOAP request message (according to the service description which it got in WSDL) having input parameters. This SOAP request message is sent to the middleware. Input interface receives the message and passes it to SOAP2ACL protocol converter. This converter extracts out the input parameters from SOAP input message and creates an equivalent ACL message. The Agent at middleware is directed to send the generated ACL request message is sent to the Agent at remote platform providing required services.

The Agent after receiving ACL request message processes the input parameters and then returns the output in the form of an ACL response message to the Agent at middleware. The Agent at middleware upon receiving the ACL response message passes it to ACL2SOAP protocol converter which extracts outputs from ACL message and generates a SOAP response message. The SOAP response message is finally sent to the Web Service client as shown in fig 10. In this way, the middleware helps the Web Service client search, understand and consume services provided by Software Agents.

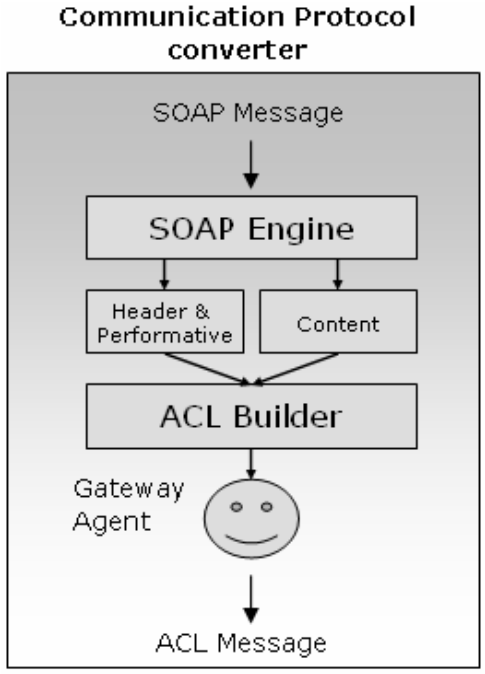

Figure 8.8: SOAP to ACL conversion: WS client consuming services provided by Agent

8.3.1.1 Algorithm for SOAP to ACL conversion

In order to elaborate the design, given below is algorithm based on comparative analysis of SOAP and ACL for SOAP to ACL conversion in order for a Web Service client to invoke an Agent.

```
get SOAP message
   get SOAP/HTTP Header
      map SOAPEndPoint to Sender attribute of ACL
      map HTTP Sender to GatewayAgent 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
```

Chapter 9

TESTING OF PROPOSED SYSTEM

This section presents some useful scenarios for evaluation of the algorithms presented in previous section.

9.1 EVALUATION OF SERVICE DISCOVERY TRANSFORMATION

This section presents a scenario for evaluation of the algorithms for service discovery transformation. Here we show how a Web Service client performs service discovery in DF of Agent Platform. The request initiated by Web service client is UDDI search query which is as follows:

```
 <businessEntity
  businessKey="677cfa1a-2717-4620-be39-6631bb74b6e1"
  operator="test " authorizedName=" Omair Shafiq: 86">
  <discoveryURLs>
    <discoveryURL useType="businessEntity">
http://uddi.rte.microsoft.com/discovery?businessKey=677cfa1a-2717-4620-be39-
6631bb74b6e1
    </discoveryURL>
  </discoveryURLs>
  <name xml:lang="en">CalculatorXmlWS</name>
<description xml:lang="en">Testing for AgentWeb Gateway by M. Omair Shafiq
</description>
  <businessServices>
    <businessService
    serviceKey="d8091de4-0a4a-4061-9979-5d19131aece5"
```

```
businessKey="677cfa1a-2717-4620-be39-6631bb74b6e1">
<name xml:lang="en">Math  Service</name>
<description xml:lang="en">
 Math Service
</description>
<bindingTemplates>
 <bindingTemplate
    bindingKey="942595d7-0311-48b7-9c65-995748a3a8af"
    serviceKey="d8091de4-0a4a-4061-9979-5d19131aece5">
   <accessPoint URLType="http">
http://202.83.166.177:8080/axis/Calculator.jws        </accessPoint>
    <tModelInstanceDetails>
     <tModelInstanceInfo
       tModelKey="uuid:42fab02f-300a-4315-aa4a-f97242ff6953">
      <instanceDetails>
       <overviewDoc>
         <overviewURL>
              http://202.83.166.177:8080/axis/Calculator.jws
          </overviewURL>
       </overviewDoc>
      </instanceDetails>
     </tModelInstanceInfo>
    </tModelInstanceDetails>
 </bindingTemplate>
</bindingTemplates>
</businessService>
</businessServices>
</businessEntity>
```

For the above mentioned generated UDDI search query, following DF search query was produced by service discovery converter of AgentWeb Gateway.

```
( REQUEST
:sender ( agent-identifier :name Creator:77166138202@cern1-7 )
:receiver  (set ( :agent-identifier DF:77166138202@cern1-7))
:content "( ( search-service ( :service-description : name Math Service) ))"
:ontology Directory-Facilitator )
```

The ACL message generated above is DF search query which is sent to DF for service discovery.

9.2 EVALUATION OF SERVICE DESCRIPTION TRANSFORMATION

This section presents a scenario for evaluation of the algorithms presented in previous section. We take a Web Services named ‘Calculator’ that contains one operation ‘add’ which requires two primitive integer types of arguments and has returns type of integer as well. Web Service Description Language (WSDL) (given below) of the web service is in plain text and is human readable.

```
<?xml version="1.0" encoding="UTF-8" ?>
   <wsdl:definitions targetNamespace="http://localhost:8080/axis/Calculator.jws"
<types>
    <xsd:schema
       targetNamespace="http://www.ecerami.com/schema"
       xmlns="http://www.w3.org/2001/XMLSchema">
       <xsd:complexType name="argument">
         <xsd:sequence>
           <xsd:element name="i1" type="xsd:int"/>
           <xsd:element name="i2" type="xsd:int"/>
```

```
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </types>

  <wsdl:message name="addResponse">
     <wsdl:part name="addReturn" type="xsd:int" />
  </wsdl:message>
  <wsdl:message name="addRequest">
     <wsdl:part name="i1" type="xsd:argument" />
  </wsdl:message>
  <wsdl:portType name="Calculator">
     <wsdl:operation name="add">
        <wsdl:input message="impl:addRequest" name="addRequest" />
        <wsdl:output message="impl:addResponse" name="addResponse" />
     </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="CalculatorSoapBinding" type="impl:Calculator">
     <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
      <wsdl:operation name="add">
        <wsdlsoap:operation soapAction="" />
           <wsdl:input name="addRequest">
               <wsdlsoap:body …/>
           </wsdl:input>
           <wsdl:output name="addResponse">
              <wsdlsoap:body …namespace="http…/axis/Calculator.jws" use="encoded" />
           </wsdl:output>
        </wsdl:operation>
     </wsdl:binding>
  <wsdl:service name="CalculatorService">
```

```
      <wsdl:port binding="impl:CalculatorSoapBinding" name="Calculator">
         <wsdlsoap:address location="http://localhost:8080/axis/Calculator.jws" />
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

In case of interoperability among Agents and Web Services, WSDL of calculator web services is to be published in Directory Facilitator of Agent Platform and hence WSDL is transformed service description transformation component of AgentWeb Gateway into Directory Facilitator Agent Description (DFAgentDescription). DFAgentDescription is serialized in binary format and is not human readable. Information about DFAgentDescription about object having values is shown below:

DFAgentDescription

- AgentID = 'CalculatorAgent:reverse-ip@machine-name'
- Ontologies = 'CAOntology'
- Protocols = ''
- Languages = ''
- Lease time= 'default'
- Scope = 'default'

Service-Description

- Name = 'add'
- Type = 'Math'
- Ontologies = 'addOntology'
- Protocols = ''
- Languages = ''
- Ownership = 'CalculatorAgent'

Property

- Name = ‘addRequest’

- Value = ‘AddRequestActionSchema’

- Name = ‘addReturn’

- Value = ‘AddResponsePredicateSchema’

addOntology has following information:

addOntology

Concept Schema

- Name = ‘argument’

AgentAction Schema

- Name = i1

- Schema = Concept (argument)

Predicate Schema

- Name = addReturn

- Schema = Primitive (Integer)

argument (concept schema)

- name = ‘i1’

- type = Primitive (Integer)

- name = ‘i2’

- type = Primitive (Integer)

In order to publish WSDL of Web Service in Directory Facilitator, it has been converted into Directory Facilitator Agent Description as given above according to algorithms in section 5.

9.3 EVALUATION OF COMMUNICATION PROTOCOL TRANSFORMATION

This section completes the above mentioned scenario, i.e. after service description transformation, communication protocol transformation is required for

service invocation. Consider a WS/SOAP client want to get services provided by an Agent that provides services of add, subtract etc. The WS/SOAP client would send request in according to its SOAP format as follows:

SOAP Request

```
POST /InStock HTTP/1.1
Host: http://202.83.166.177:8080/axis/Calculator.jws
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body xmlns:m="http://202.83.166.177:8080/axis/Calculator.jws">
   <m:add>
    <m:i1>2</m:i2>
    <m:i1>3</m:i2>
   </m:add>
 </soap:Body>
</soap:Envelope>
```

Using communication protocol converter of AgentWeb Gateway, the SOAP message will be transformed into ACL request according to the algorithm presented in section 8.1. The transformed ACL request is given below:

Transformed ACL Request

```
(request
   :sender (agent-identifier
```

```
    :name Gateway:78166138202@Cern1-7
    :addresses (sequence http://202.83.166.187:7776/acc))
  :receiver (set (agent-identifier
    :name MathAgent@78166138202@Cern1-7
    :addresses (sequence http://202.83.166.187:9999/acc)))
  :content
  "(action (addAgentAction
    :properties (set
      (property i1 2)
      (property i2 3)))")
```

The transformed ACL message will be forwarded to the actual agent (MathAgent) providing the required add service. The MathAgent would response accordingly in ACL which will be received by Gateway Agent. The ACL response is given below:

ACL Response

```
(request
  :sender (agent-identifier
    :name MathAgent:78166138202@Cern1-7
    :addresses (sequence http://202.83.166.187:7776/acc))
  :receiver (set (agent-identifier
    :name Gateway@78166138202@Cern1-7
    :addresses (sequence http://202.83.166.187:9999/acc)))
  :content
  "(action (addAgentAction
    :properties (set
      (property addResult 5)))")
```

Using communication protocol converter of AgentWeb Gateway, the ACL response message would be converted into SOAP response message according to the algorithm presented in section 8.2. The transformed SOAP response is given below:

SOAP Response

```
HTTP/1.1 200 OK
Content-Type: application/soap; charset=utf-8
Content-Length: nnn<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://202.83.166.177:8080/axis/Calculator.jws">
    <m:add>
      <m:addResult>5</m:addResult>
    </m:add>
  </soap:Body>
</soap:Envelope>
```

Chapter 10

PERFORMANCE ANALYSIS AND RESULTS

In this chapter, the performance evaluation of AgentWeb Gateway required transformations for integration is analyzed. It gives an estimate of an average delay imposed on normal transactions due to transformations for required integration.

10.1 SERVICE DISCOVERY CONVERTER

Both in DF search query and UDDI search query, more than one Service-Description and Business-Entity information can be used for search request in service registry respectively.

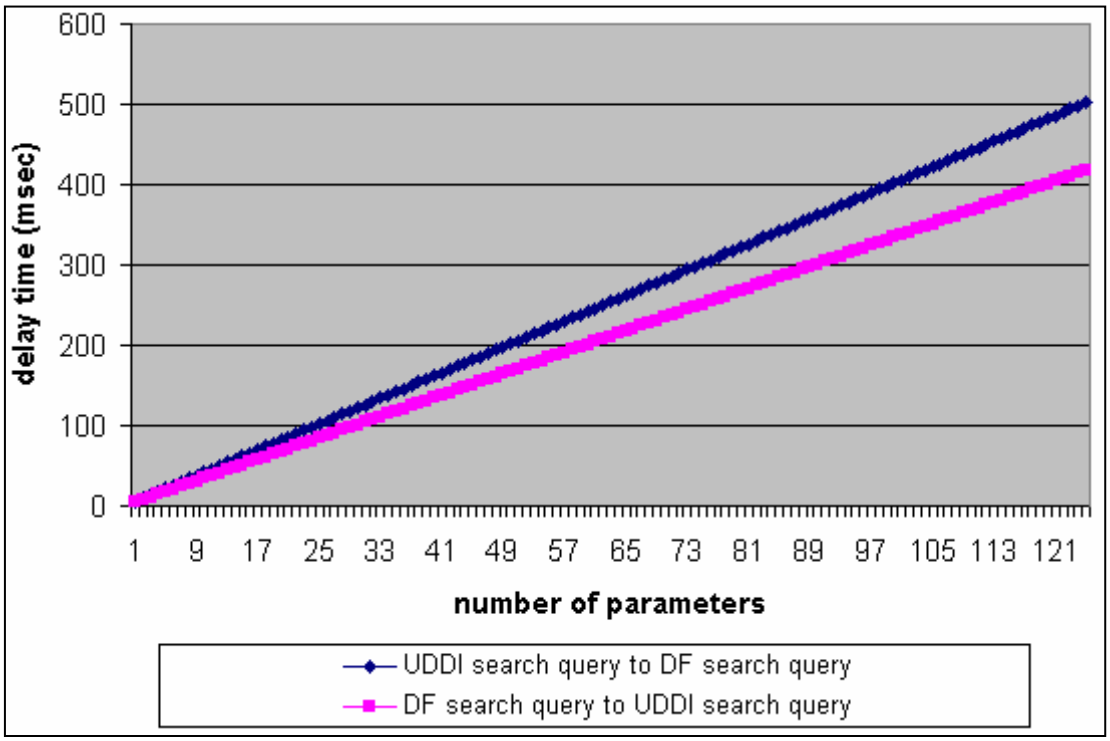

Figure 10.1: Performance analysis of service discovery transformation

The graph is shown in figure below. On x-axis, it is number of parameters in search query are shown i.e. Service-Description in DF search query and Business-Entity in UDDI search query. On y-axis, its time delay in milliseconds is shown. The increment of the required delay is linear in behavior as number of parameters is increased in the search query, both in UDDI search query and DF search query.

Another important thing to note here is that as number of parameters is increased, UDDI to DF search query conversion takes more time than that of DF to UDDI search query conversion. The reason to it is that in case of UDDI search query to DF search query conversion, UDDI search query is needed to be processed which is basically and XML based text file. Whereas in case of DF search query to UDDI search query conversion, DF search query is needed to be processed which is an object and based on binary information. The processing of binary information is faster than processing of XML based text information. So as we keep on increasing number of parameters the difference in the delay becomes more significant due to more time needed to be processed in search query conversion.

10.2 SERVICE DESCRIPTION CONVERTER

Accuracy depends on the provided information of Web Service in WSDL and Software Agent in DFAgentDescription. If it is completely valid then 100% results can be obtained. In case of a Software Agent publishing services in UDDI (DFAgentDescription to WSDL conversion), time required for transformation for Service description depends upon the complexity of ontology and number of service-description in DFAgentDescription.

In case of Web Services publishing its services in Directory Facilitator (WSDL to DFAgentDescription conversion), time required for transformation for service description depends on Complex-Types and number of operations in portType of WSDL.

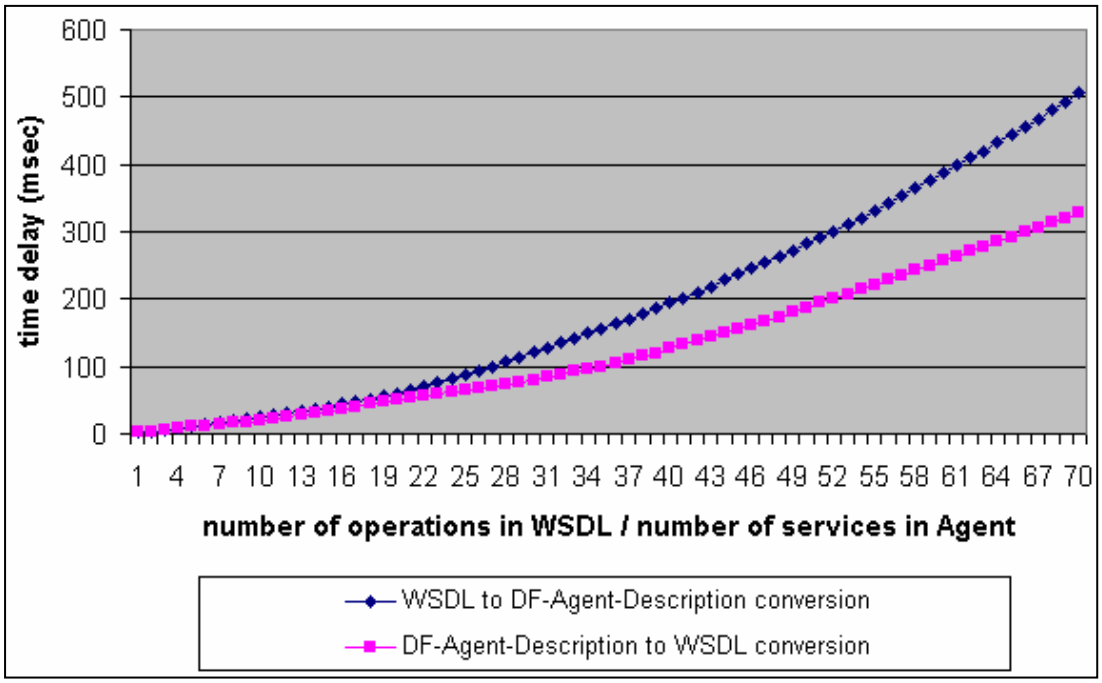

Figure 10.2: Performance analysis of service description transformation

A graph is shown above; x-axis shows the number of operations in portType of a WSDL (in case of WSDL to DF-Agent-Description conversion) or number of services in DF-Agent-Description ontology of Agent in case of (DF-Agent-Description to WSDL conversion). Y-axis gives the delay occurred in milliseconds for required transformation. We have analyzed the delay occurs in the process of transformation on either side. In case of DFAgentDescription to WSDL conversion for an Agent to publish its services in UDDI, It was observed that as number of services of

an agent increases the time taken for transformation also increases. Same is the case for WSDL to DFAgentDescription when a Web Service is to be published in Directory Facilitator, delay in transformation increases with increase in number of operations in portType of WSDL.

Next observation is that line of increment in delay of transformation shows exponential behavior. Reason for exponential behavior of increment is as there is increment of one operation in portType of WSDL, it would have some input message, output message, elements and complex types (optional). Same is the case in with DFAgentDescription.

Graph of WSDL to DFAgentDescription transformation has rapid increase than in case of DFAgentDescription to WSDL transformation. Reason for this is WSDL file requires more time in parsing as is based on file with plain text than that of DFAgentDescription which is in the form of object and binary based.

10.3 COMMUNICATION PROTOCOL CONVERTER

Both in SOAP request/response message and ACL message, only one operation or AgentAction/Predicate respectively can be targeted in a single call. Accuracy of SOAP to ACL and ACL to SOAP transformation depends upon the accuracy of message. A valid message would be transformed into it's vice versa with 100% accuracy.

In case of a Software Agent invokes a Web Service, time required for communication protocol transformation (ACL to SOAP conversion) depends upon the complexity of schema of Property objects in Service-Description of DF-Agent-Description of an Agent. If there are primitive schemas only, then transformation

process would take almost similar time as expected. In case of concept schema involved, additional time would be required for transformation of concept schema into complex type.

In case of a Web Service client invokes a Software Agent, time required for communication protocol transformation (SOAP to ACL conversion) depends upon the complexity of input and output parameters. If inputs and outputs are primitive data-types, transformation process would take almost similar time as expected. If there are complex data-types involved, additional time would be required for transformation of complex data-type into ontology (concept schema).

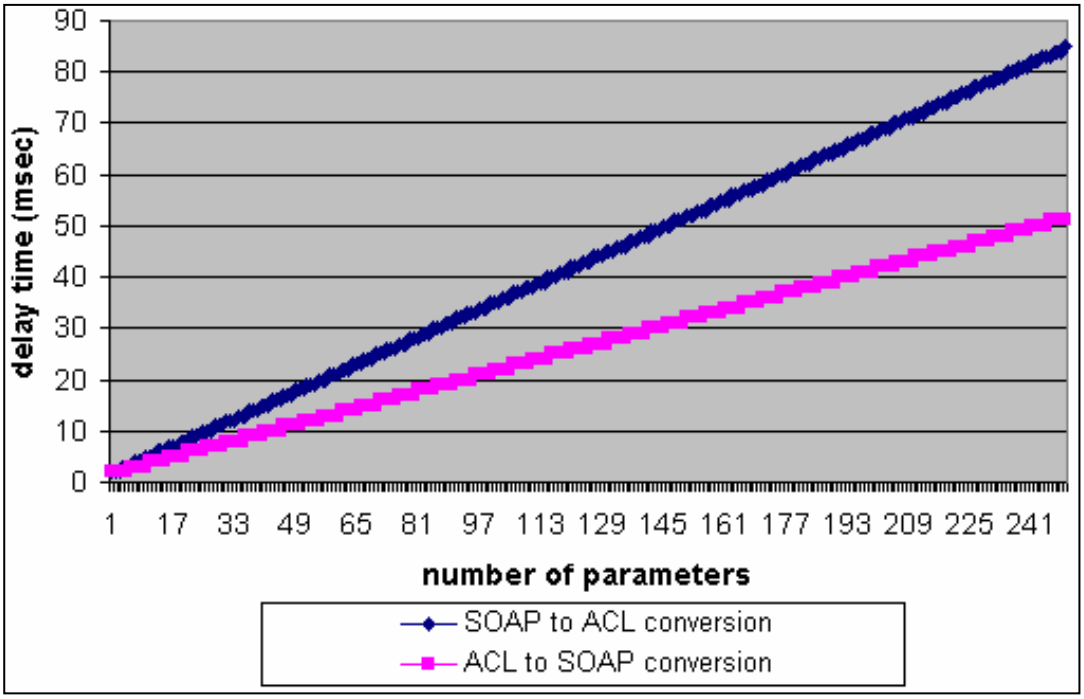

Figure 10.3: Performance analysis of communication protocol transformation

In following figure, we have analyzed the delay occurs in the process of transformation on either side. In case of ACL to SOAP conversion for an Agent to

invoke web service, it was observed that as number of parameters SOAP message increases, the time taken for transformation also increases. Same is the case for ACL to SOAP when a Web Service client is to invoke an Agent, delay in transformation increases with increase in number of Property elements of content of ACL message.

A graph is shown in figure 6. X-axis shows the number of parameters in SOAP (in case of SOAP to ACL conversion) or in ACL in case of (ACL to SOAP conversion). Y-axis gives the delay occurred in milliseconds for required transformation.

The line of increment in delay of transformation shows linear behavior. Exponential behavior can only be shown in case of complex data-types are used.

For a given number of parameters, ACL to SOAP conversion takes little less time as compared to SOAP to ACL conversion. Reason for this is SOAP requires more time parsing as it is based on plain text than that of ACL message which is in the form of object and binary based.

10.4 TIME DISTRIBUTION AMONG TRANSFORMATION

In this section, time required by different kinds of transformation is compared with each other. After performing a number of test cases on the implemented system, time distribution among three kinds of transformation is analyzed. As it has been discussed earlier that there is there kinds of transformation required i.e. service discovery transformation, services description transformation and communication protocol transformation. It was noticed that on the average, service discovery transformation takes 32% of time, service description transformation required 45% of time and communication protocol transformation requires 23% of time among whole

time required for integration of Agents and Web Services. Since most of the time is taken by service discovery and service description transformation.

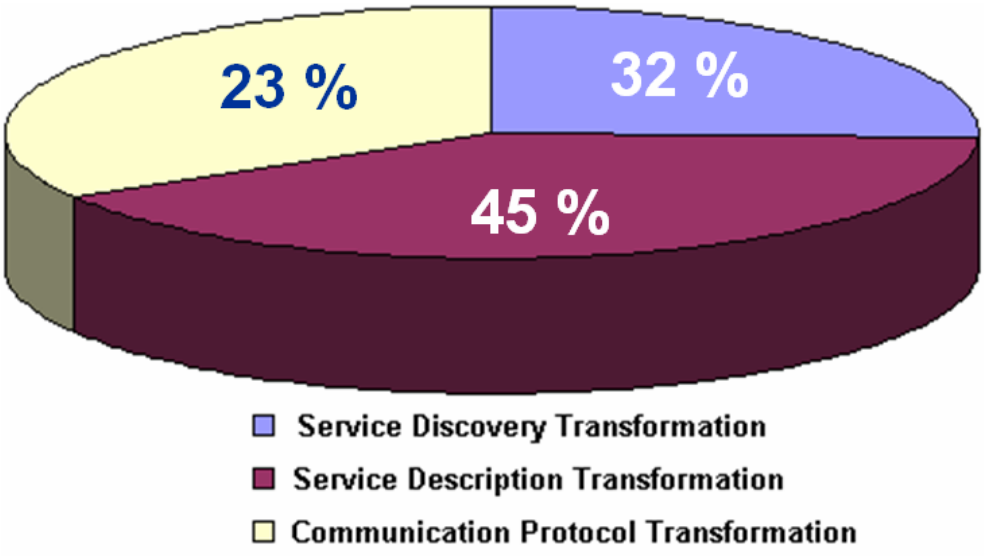

Figure 10.4: Time distribution among required transformations

In case of a real life scenario, for integration of a particular Agent with Web Service, only once service discovery and service description transformation will be required. Once a service is discovered and published, then it can be invoked several times.

10.5 TRANSFORMATION DELAY PER TRANSACTION

In this section, time taken by transformations is compared to time taken by transformation. The methodology for finding the average time taken for transformation to achieve the required integration is as follows:

- Average time taken by a web service client to discover, publish and invoke a web service was noted.

- Average time taken by an Agent to discover, publish and communicate with another was noted
- Average of time required from 1 and 2 was taken
- The same experiment was repeated with different number of input parameters (as discussed in chapter 9 and chapter 10, section 10.1, 10.2 and 10.3).

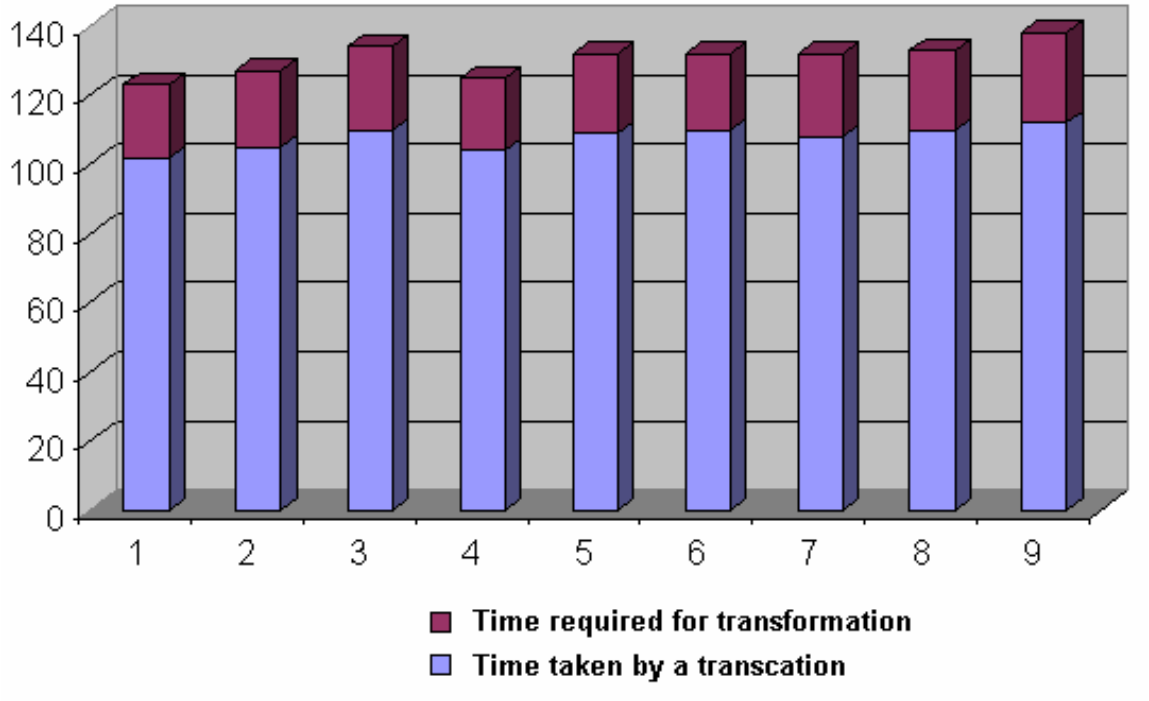

Figure 10.5: Time distribution among required transformations

After performing a number of experiments, it was noticed that, on the average 21% extra time is required by the transformation mechanism than among the normal transaction time individually in agents and web services, to achieve the required integration of agents and web services.

Chapter 11

APPLICATION OF AGENTWEB GATEWAY

A real life application has also been developed to show the significance of proposed system. It is called as "Distributed services based conference planner application using for software agents, web services and grid services". It shows collaboration of Agents, Web Services and Grid Services to do some real life activity. Here different agents communicate with each other and use information from Grid Service and Google Web Service to plan a conference.

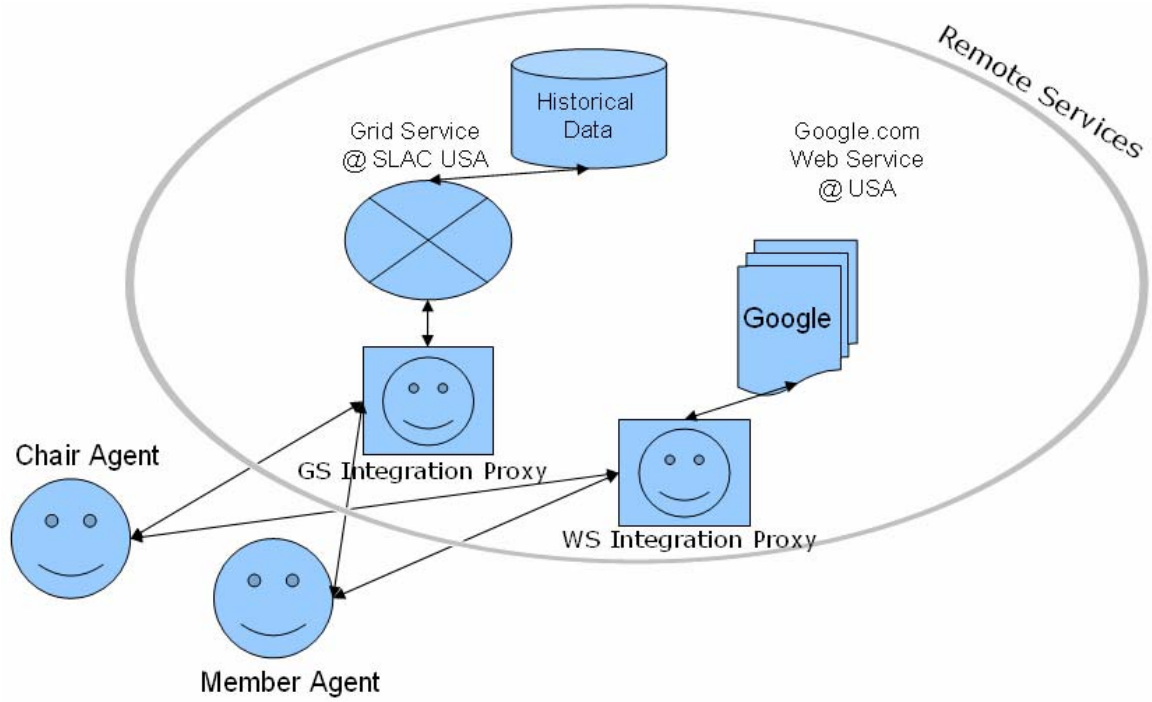

Figure 11.1: Conference planner application using AgentWeb Gateway

Conference Chair Agent is considered as Conference chair person who wants to initiate the planning of a conference. To work out the topics this agent selects

the members from the active member list (any number of members) and send them a request to suggest topics for the conference. Conference Chair Agent when receives reply from the members it compiles the results and generates the final list of topics. Along with the selection of the topics, the agent plans out the time and place for the conference along with topics by searching a huge database containing data of the past conferences. This search is carried out by utilizing the Grid services. Here the AgentWeb Gateway is seamlessly involved to make Agents and Grid Services communicate with each other.

Conference Member Agents receive requests from the Conference Chair Agent and select some topics which take form of their preference list provided by user. These Agents then search the Google web service, to get a look at related articles and papers. Here once again, the AgentWeb Gateway API acts seamlessly to make Agents and Google Web service communicate with each other. These Agents then send their list of topics along with related articles titles to the Conference Chair Agent.

11.1 GEOGRAPHICAL DISTRIBUTION OF SERVICES

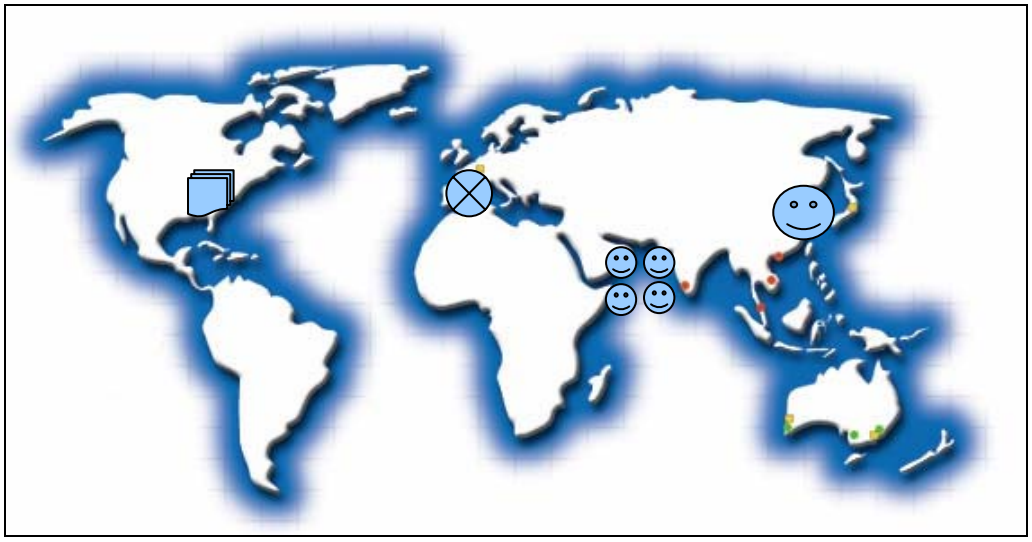

Figure11.2: Geographical monitoring service for Multi Agent Systems

11.2 ANALYSIS AND EVALUATION OF TIME DELAY IN COMMUNICATION WITH DIFFERENT SERVICES

11.2.1 Evaluation scenario

The performance analysis for the conference planner application was carried out in a comprehensive and systematic manner. Initially the evaluation of the web, Grid and the Agents were done individually and then an overall evaluation scenario was created. The purpose for this two phase testing was to realize the extent to which these features are contributing in the performance of the application. In addition to that the integrated testing was carried out to realize performance of the application with all the features integrated together.

11.2.2 Analysis of Agents interaction wit Google Web Services

Network delay analysis among distributed services was carried out between Comtec, Japan and Google USA. The results are shown in Figure 9.7.

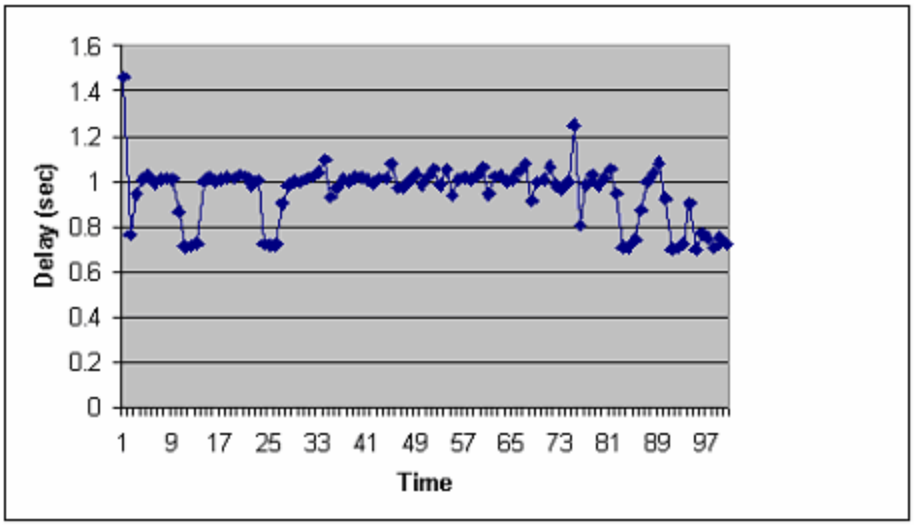

Figure 11.3: Network Delay Between Comtec Japan and Google USA

Average delay = 0.95373

Network delay analysis among distributed services was carried out between NUST, Pakistan and Google USA. The results are shown in Figure 9.8 where Average delay = 3.68548.

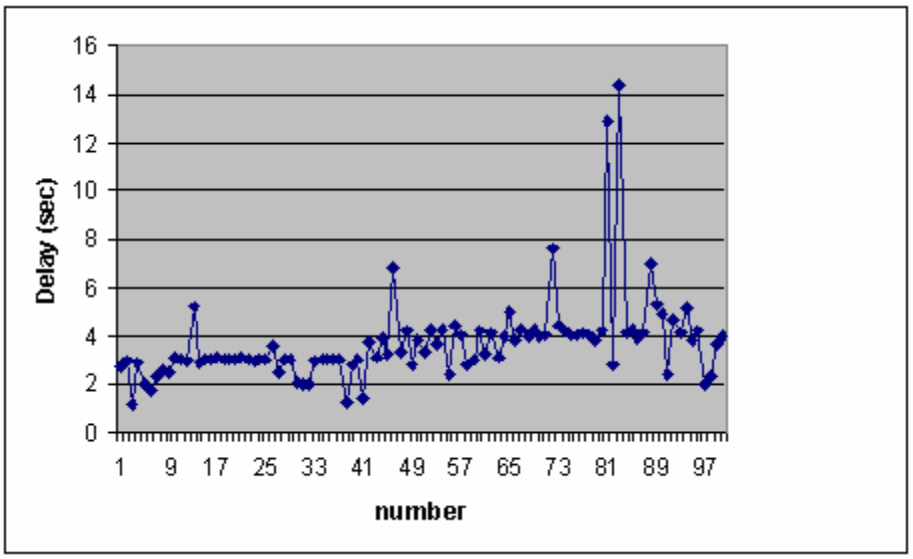

Figure 11.4: Between NUST Pakistan and Google USA

The results for the network delay clearly depicts that the application is bandwidth dependent. As the average delay when agents deployed at Comtec, Japan access the Google, USA is less then one second where as the average delay when the agents deployed at NUST, Pakistan access the Google, USA is more than three seconds. This significant difference in the delay is occurring due to different bandwidths available at Comtec and NUST. The results thus clearly depict that the performance of the agent accessing the we an web-service is bandwidth dependent and has an impact on the over all performance of the application.

11.2.3 Analysis of Agents communication with Grid Service

The Grid services (the Grid node) was deployed at NUST and were accessed from Japan. The results are shown in Figure 9.9.

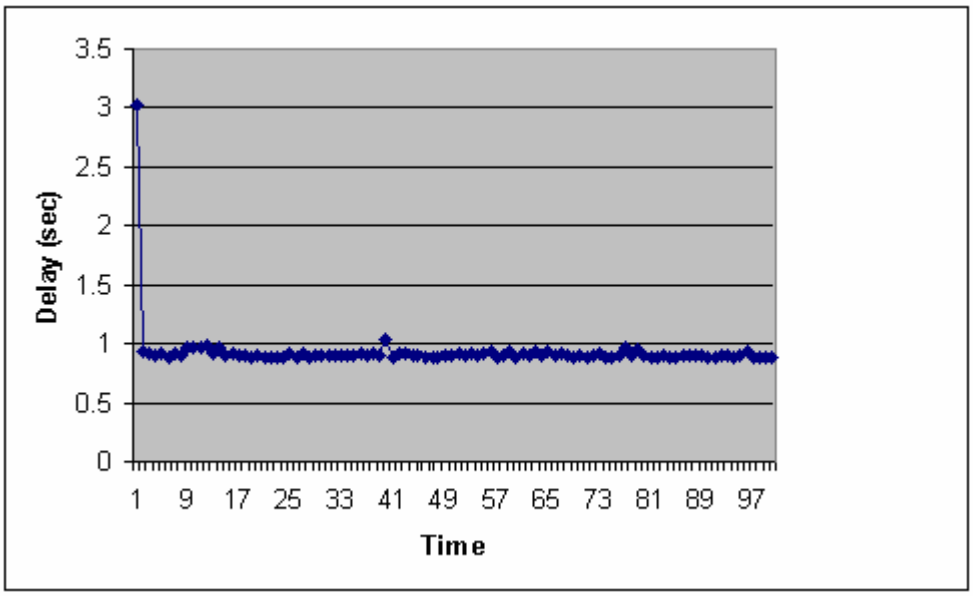

Figure 11.5: Between Comtec Japan and NUST Pakistan

Average delay = 0.92795

Chapter 12

FUTURE RESEARCH DIRECTIONS

In this chapter, several future research directions from this project have been discussed in an abstract manner.

12.1 SOC COMPLIANT MULTI AGENT SYSTEMS

We foresee Service Oriented Computing compliant Multi Agent Systems is a new emerging sub-domain in Multi Agent Systems. The key idea is to enhance the current Multi Agent Systems to comply with the principles of Service Oriented Computing. Following are the two research issues that have been identified in order to proceed to this direction.

12.1.1 Interoperability issues among different FIPA compliant Multi Agent Systems

Service-oriented architectures stress interoperability between supporting systems so that service entities could easily be modeled across heterogeneous platforms. Interoperability in Multi Agent Systems (MAS) is a key research issue. Standards bodies like FIPA have proposed abstract architectures for interoperable agent platforms. In spite of formulated standards, MAS developed by different vendors using same specifications are still not interoperable. We need to research on issues and devise solutions for developing interoperability in multi agent systems.

FIPA has proposed multiple Message Transport Protocols for interoperation among multi agent systems. IIOP (Internet Inter ORB Protocol) developed by OMG (Object Management Group) and HTTP are among the widely known protocols. We need to model complete platform interoperability with message

passing, agent description advertisement on remote platform, agent mobility and remote agent service utilization.

12.1.2 Dynamic ontology sharing support for FIPA compliant Multi Agent Systems

Currently the FIPA compliant Multi Agent Systems don't support dynamic ontology sharing. We discuss the problem associated with the two agents, who want share a common ontology for some domain. It ensures that the agents ascribe the same meaning to the symbols used in the message. For a given domain, designers may decide to use ontologies that are explicit, declaratively represented and stored somewhere or, alternatively, ontologies that are implicitly encoded with the actual software implementation of the agent themselves and thus are not formally published to an ontology service.

Solution is discussed in the form of ontology service which will be based on FIPA specs and consists of Ontology Agent (OA) and different kind of repositories to store the ontologies. Where ontology agent has the ability of tasks to be achieved like, discovery of public ontologies in order to access them, maintain (for example, register with the DF, upload, download, and modify) a set of public ontologies, translate expressions between different ontologies and/or different content languages, respond to query for relationships between terms or between ontologies, and facilitate the identification of a shared ontology for communication between two agents.

12.2 AUTONOMOUS SEMANTIC GRID SERVICE DESCRIPTION LANGUAGE

Another approach to proceed towards the ultimate goals of Autonomous Semantic Grid i.e. to achieve synergy of Agents, Web Services and Grid Services, is that current Web Services Framework should be enhanced. Key idea here is, while the enhancement of web services framework, the specifications of the three technologies should be managed in a layer manner in order to avoid the duplication of services in different technologies while integration. So we propose an Agent Services Description Language that is composed of different layers. Figure is shown below for more details.

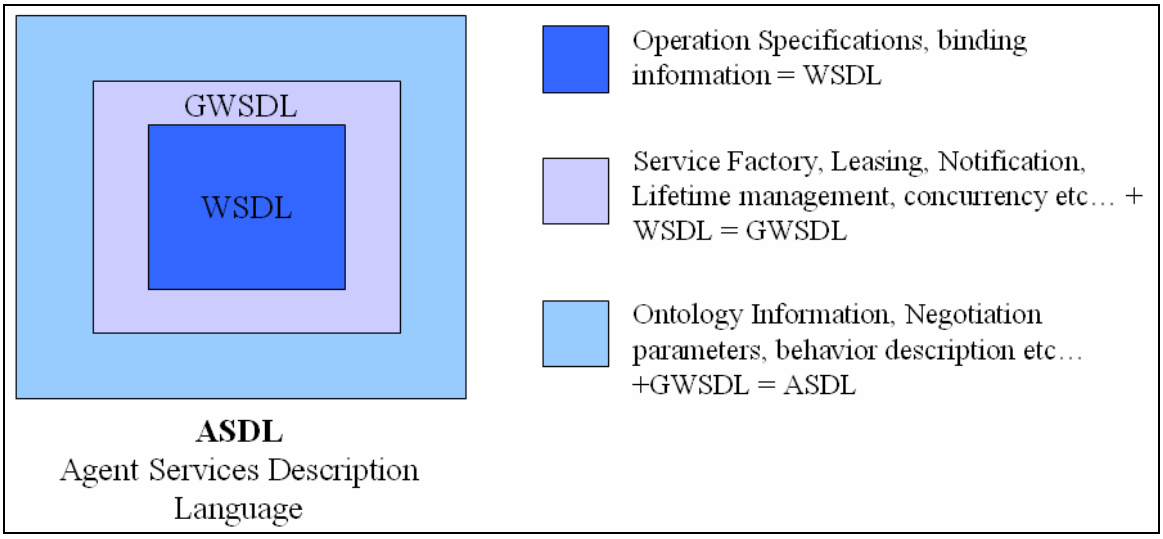

Figure 12.1: Evolution of Service Description Language for Autonomous Semantic Grid

12.3 ASYNCHRONOUS INVOCATION SUPPORT FOR WEB SERVICES

In case of SOC compliant Multi Agent Systems, using Web Services underline Agent Platform is a better choice than that of using CORBA. Currently Web Services have a drawback that while a web service client accesses a web service, it hangs up until it gets the response back. It can also be called as synchronous invocation.

We propose a layer for Web Services Framework that enables the asynchronous invocation support for current web services along with a caching mechanism. The addition layer should server as proxy to each client. When a web service client requests some web service, the proxy inside the additional layer should catch the query and release the web service client. After this the proxy should communicate to the web service locally, gets the result and return back to requestor web service client along with adding an entry in caching database so that if there are multiple requests to same web service with same input parameters, then every time the web service should not be invoked, rather the result should be obtained from cache.

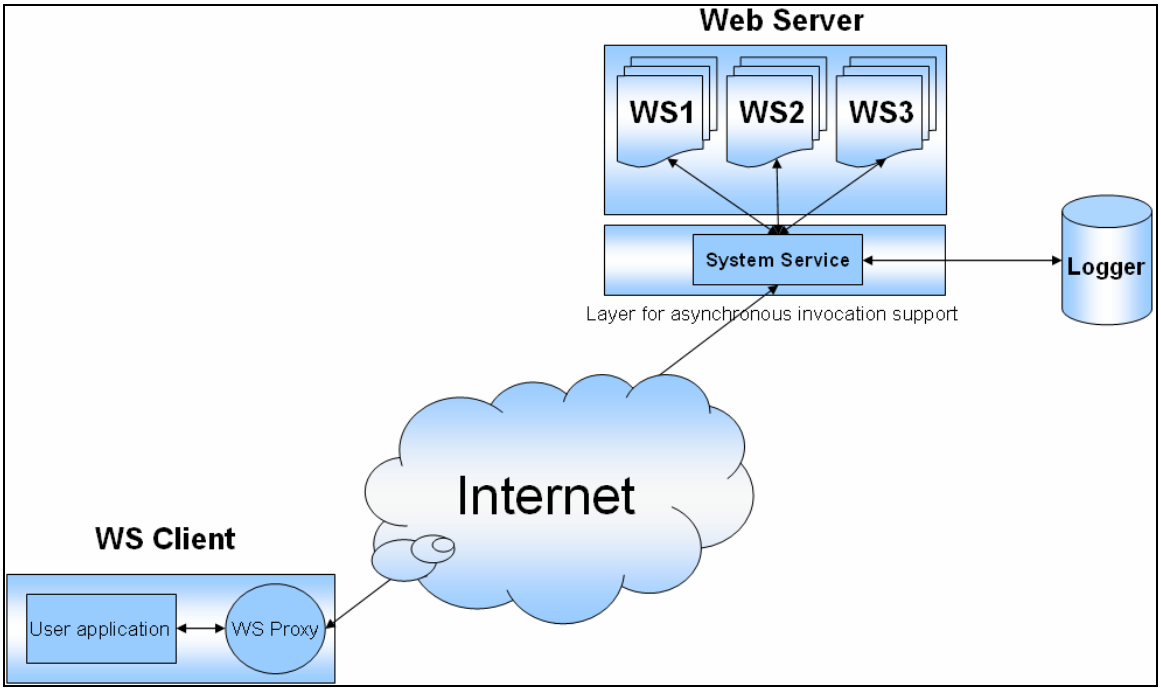

Figure 12.2: Asynchronous Invocation support for Web Services

12.5 GEOGRAPHICAL MONITORING SERVICE FOR MULTI AGENT SYSTEMS

Inspired from the work of CALTECH's project named Monitoring Agents using a Large Integrated Services Architecture (MONALISA) which have capability to monitor and manage the distributed services across the globe, we proposed a global geographical service for Multi Agent Systems. Multi Agent Systems technology is getting mature and will be used to solve in a number of real life problems.

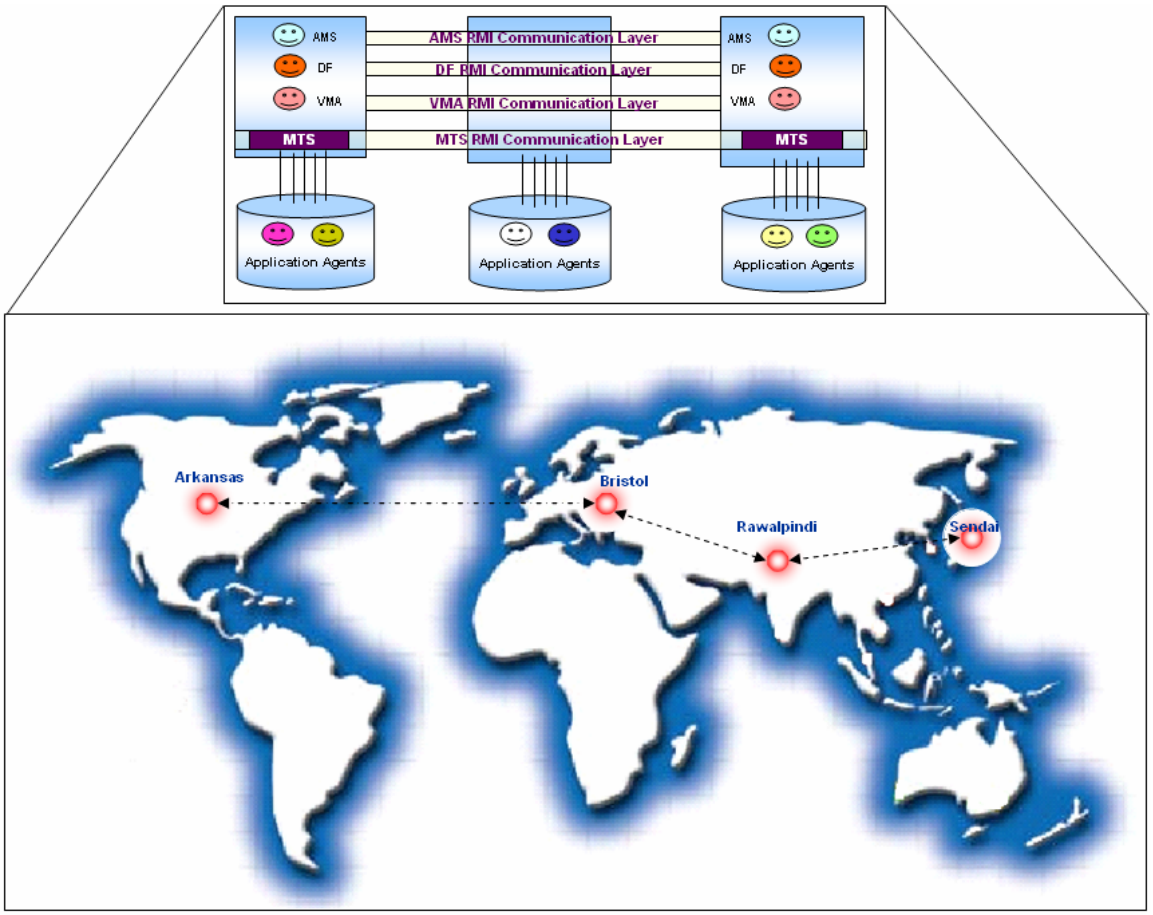

Figure 12.3: Global and geographical monitoring service for Multi Agent Systems

Along with this increasing usage, there is need to monitor and manage the different agents deployed on different containers of Multi Agent Systems across the globe. An Agent Platform consists of multiple containers deployed at different locations. There is need of a tool that could show the administrator about the state of

Multi Agent System over a world-map where all the communications among agents deployed widely across the globe could be seen.

Chapter 13

REFERENCES

[1] S. Tuecke, ANL; K. Czajkowski, I. Foster, ANL; J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, P. Vanderbilt, GWD-R (draft-ggf-ogsi-gridservice-33) "Open Grid Services Infrastructure (OGSI) Version 1.0", June 27, 2003
http://www.ggf.org/ogsi-wg

[2] Ian Foster, David Snelling, “Web Service Resource Framework – WSRF”, March 2004
http://www.globus.org/wsrf/faq.asp

[3] Ian Foster, Nicholas R. Jennings, Carl Kesselman, "Brain Meets Brawn: Why Grid and Agents Need Each Other", Proc. Autonomous Agents and Multi Agent Systems (AAMAS) July 2004, New York, USA

[4] The Web Services Agent Integration Project AgentCities.NET

[5] H. Kuno and A. Sahai, "My Agent Wants to Talk to Your Service: Personalizing Web Services through Agents" 1st International Workshop on "Challenges in Open Agent Systems, July 2002, Bologna, Italy

[6] Jonathan Dale, Steven Willmott and Bernard Burg, “Agentcities: Building a Global Next-Generation Service Environment”, Proceedings of OpenNet, June 2002, Geneva, Switzerland

[7] Michael Luck, Peter McBurney, Chris Preist, "Agent Technology: Enabling Next Generation Computing - A Roadmap for Agent Based Computing", January 2003, AgentLink II.

[8] Foster, I. and Kesselman, C. (eds.), “The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)”. Morgan Kaufmann, 2004.

[9] Wooldridge, M. Agent-based software engineering. IEE Proc. Software Engineering, 144. 26-37. 1997.

[10] Nicholas R. Jennings, Katia Sycra, Michael Wooldbridge, “A Roadmap of Agent Research and Development”, Autonomous Agents and Multi-Agent Systems, pp.275-306, March 1998, Kluwer Academic Publisher, Boston USA

[11] Mahesh S. Raisinghani, “Electronic Commerce at the Dawn of Third Millenium”, Idea Group Publishing, (2000).

[12] Global Grid Forum (GGF),
http://www.gridforum.org/

[13] Foster, I., Kesselman, C., Nick, J. and Tueske, S., “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration”, Globus,Project, July 2002
www.globus.org/research/papers/ogsa.pdf

[14] K. Mori, "Autonomous Decentralized Systems: Concept, Data Field Architecture and Future Trends", Proc. of the first IEEE International Symposium on ADS (ISADS), pp.28-34, May 1993, Kawasaki, Japan.

[15] Wolski, R., Brevik, J., Plank, J. and Bryan, T. Grid Resource Allocation and Control Using Computational Economies. Berman, F., Fox, G. and Hey, T. eds. Grid Computing: Making the Global Infrastructure a Reality, Wiley and Sons, 2003, 747-772.

[16] 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, August 2004, Valencia, Spain.

[17] Wooldridge, M. and Jennings, N.R. Software Engineering with Agents: Pitfalls and Pratfalls. IEEE Internet Computing, 3 (3). 20-27. 1999.

[18] H. Farooq Ahmad, Kashif Iqbal, Arshad Ali, Hiroki Suguri, "Autonomous Distributed Service System: Basic Concepts and Evaluation", Proc. The Second International Workshop on Grid and Cooperative Computing, GCC 2003, pp. 432-439, Shanghai, China.

[19] Goble, C.A., De Roure, D., Shadbolt, N.R. and Fernandes, A. Enhancing Services and Applications with Knowledge and Semantics. The Grid: Blueprint for a New Computing Infrastructure (2nd Edition), Morgan-Kaufmann.

[20] Levine, D. and Wirt, M. Interactivity with Scalability: Infrastructure for Multiplayer Games. The Grid: Blueprint for a New Computing Infrastructure (2nd Edition), rgan Kaufmann, 2004.

[21] H. Farooq Ahmad, K. Mori, "Autonomous Information Fading and Service-Guided Navigation Techniques for Mobile Agents", Proceeding of IEEE Computer Society, SMC99 conference pp. II-83-II-II-87, August 1999, Japan.

[22] Hiroyuki Tsunemitsu, H. Farooq Ahmad, Helene, Arfaoui, Kinji Mori, "Autonomous Decentralization Technology for Service Integration of Different Service Providers", 12th SICE Symposium on Decentralized Autonomous Systems, pp. 373-378, May 2000, Japan.

[23] H. Farooq Ahmad, Arshad Ali, Hiroki Suguri, Zaheer Abbas Khan, Mujahid ur Rehman, “Decentralized Multi Agent System: Basic Thoughts”, 11th Assurance System Symposium, Sendai, Japan.

[24] Abdul Ghafoor, Mujahid ur Rehman, Zaheer Abbas Khan, Arshad Ali, Hafiz Farooq Ahmad, Hiroki Suguri, “SAGE: Next Generation MAS” pp. 139-145, June 2004, Vol 1. Navada, USA.

[25] Salman Bashir, Mujahid ur Rehman, H Farooq Ahmad, Arshad Ali, Hiroki Suguri. “Distributed and Scalable Message Transport Service for High Performance Multiagent Systems”, pp. 152-157, INCC 2004 Pakistan.

[26] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, "W3C specifications Web Services Description Language (WSDL) 1.1"

[27] Foundation for Intelligent Physical Agents, FIPA Agent Management Specifications 2002, SC00023J, Geneva, Switzerland.

Appendix

RESEARCH CONTRIBUTION

A.1 INTERNATIONAL JOURNAL PUBLICATIONS

1. M. Omair Shafiq, H. Farooq Ahmad, Hiroki Suguri and Arshad Ali, "Autonomous Semantic Grid: Principles of Autonomous Decentralized Systems for Grid Computing", IEICE & IEEE Joint Journal, Special issue on Autonomous Decentralized Systems (ADS) December 2005. (Accepted)
2. M. Omair Shafiq, H. Farooq Ahmad, Hiroki Suguri and Arshad Ali, "Bridging Multi Agent Systems and Grid Computing: an initiative towards integration of Software Agents and Web Services", The International Journal of Web Services Research (JWSR), Special issue on Bridging Communities: Semantically Augmented Metadata for Services, Grids, and Software Engineering (Submitted)

A.2 INTERNATIONAL CONFERENCE PUBLICATIONS

1. H. Farooq Ahmad, Hiroki Suguri, M. Omair Shafiq and Arshad Ali, "Autonomous Distributed Service System", Proceedings of 13th Assurance System Symposium, Mori Labs, Tokyo Institute of Technology, September 2004, Tokyo Japan.
2. H. Farooq Ahmad, Hiroki Suguri, M. Omair Shafiq and Arshad Ali, "Autonomous Distributed Service System: Enabling Web Services Communication with Software Agents", Proceedings of 16th International

Conference on Computer Communication (ICCC), pp.1167-1173, September 2004, Beijing China.

3. Hiroki Suguri, H. Farooq Ahmad, M. Omair Shafiq and Arshad Ali, "AgentWeb Gateway: Enabling Service Discovery and Communication among Software Agents and Web Services", Proc. Third Joint Agent Workshops and Symposium (JAWS2004), pp. 212-218, October 2004, Karuizawa, Japan.
4. M. Omair Shafiq, Arshad Ali, Ejaz Ahmad, H. Farooq Ahmad and Hiroki Suguri, "Detection and Prevention of Distributed Denial of Services Attacks by Collaborative Effort of Software Agents, First prototype implementation", Proceedings of the 23rd IASTED International Multi Conference on Applied Informatics - Parallel and Distributed Computing and Networks (PDCN), pp 456-800, February 2005, Innsbruck Austria.
5. M. Omair Shafiq, Arshad Ali, H. Farooq Ahmad and Hiroki Suguri, "Mobile Network End Host Remote Monitoring Agent for a Mobile Agents Based Approach for Detection and Prevention of Distributed Denial of Services Attacks" in International Conference on Internet Computing ICOMP 05" as part of " The 2005 International MultiConference in Computer Science & Computer Engineering, June 27-30, 2005, Las Vegas, Nevada, USA".
6. M. Omair Shafiq, Arshad Ali, H. Farooq Ahmad and Hiroki Suguri, "A middleware based approach for integration of Software Agents and Web Services", Emerging Technologies for Next generation GRID (ETNGRID-2005) in 14th IEEE International Workshops on Enabling Technologies:

Infrastructures for Collaborative Enterprises (WETICE-2005), June 13-15, 2005, Linkoping University, Sweden.

7. M. Omair Shafiq, Arshad Ali, H. Farooq Ahmad and Hiroki Suguri, "Multi Agent Systems for Enhancement of Grid/Web Services Platforms" in International Symposium on Web Services and Applications" ISWS'05 as part of "The 2005 International MultiConference in Computer Science & Computer Engineering, June 27-30, 2005, Las Vegas, Nevada, USA".
8. M. Omair Shafiq, Arshad Ali, H. Farooq Ahmad and Hiroki Suguri, "Autonomous Semantic Grid as synergy of FIPA Multi Agent System and OGSA Grid Services Framework", IEEE International Conference on Services Computing (SCC-2005), 11-15 July 2005, Orlando Florida, USA. (Accepted)
9. M. Omair Shafiq, Arshad Ali, H. Farooq Ahmad and Hiroki Suguri, "Autonomous Semantic Grid - A Step towards integration of Software Agents and Web Services in Grid Computing", The International Conference on Internet Technologies and Applications (ITA 05), September 2005, Wrexham, North Wales, UK. (Accepted)
10. M. Omair Shafiq, Arshad Ali, H. Farooq Ahmad and Hiroki Suguri, "Service Description Transformation for Integration Software Agents and Web Services", IEEE International Conference on High Performance Computing (HiPC) December 2005, Goa India. (Submitted)
11. M. Omair Shafiq, Arshad Ali, H. Farooq Ahmad and Hiroki Suguri, "Communication Protocol Transformation for Integration Software Agents and

Web Services", 6th IEEE/ACM International Workshop on Grid Computing (Grid 2005) November 13-14, 2005, Seattle, Washington, USA. (Submitted)

A.3 APPLICATION DEMONSTRATIONS OF RESEARCH WORK IN INTERNATIONAL CONFERENCES

1. M. Omair Shafiq, Arshad Ali, Amina Tariq, Amna Basharat, H. Farooq Ahmad, Hiroki Suguri and Fawad Nazir “A Distributed Service Application using Software Agents, Grid Services and Web Services”,4th International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS), 25 - 29 July 2005, Utrecht University, The Netherlands.
2. H. Farooq Ahmad, Hiroki Suguri, Arshad Ali, Sarmad Malik, Muazzam Mugal, M. Omair Shafiq, Amina Tariq, Amna Basharat, "Scalable Fault Tolerant Agent Grooming Environment - SAGE Agent Platform", 4th International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS), 25 - 29 July 2005, Utrecht University, The Netherlands.

A.4 RESEARCH PROPOSALS

1. Architecture Framework for automating system management tasks using Agent based Automated Meta-data gathering (ABAM), submitted jointly by NUST Institute of Information Technology (NIIT) Pakistan and University of Arkansas at Little Rock (UALR) USA, accepted in first round for funding by Acxiom Research Labs USA.
2. Autonomous Distributed Services System, by PTCL (Research and Development section), Islamabad Pakistan.