WEB SERVICE ENABLED MOBILE AGENT SYSTEM

A THESIS SUBMITTED TO THE GRADUATE DIVISION OF THE UNIVERSITY OF HAWAI'I IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

DECEMEBER 2002

By
Hui Deng

Thesis Committee:

Anna Hac, Chairperson
Tep Dobry
Rahul Chattergy

Acknowledgments

I would like to thank to my thesis advisor Professor Anna Hac for her invaluable guidance, support and patience during these two years. Thanks also give to Paulo Marques at University of Coimbra, Portugal for his courtesy and generosity giving me the source code of Mobility component they had developed in their research project. The availability of source code allows me to gain deeper knowledge about the Mobility component and use it in my research implementation.

Thanks also go to my wonderful friends who have always been there whenever I need help.

Most important, thanks go to my family- my parents, my wife Hongbo and my daughter Ke – for their encouragement and support for many years. I owe them so much.

Thanks.

Abstract

Web Services and Mobile Agents are two most prominent technologies in current distributed computing world.

Built on top of existing Web protocols and based on open XML standards, Web Services are emerging to provide a systematic and extensible framework for application-to-application interaction. It is platform and language independent. Web Services are technologies that define the standardized mechanisms to describe, locate, and communicate services on Internet.

Mobile agent is considered an essential technology in the development of distributed software application because of its capability to move across distributed environments, interact with local resources and other mobile agents. Mobile agent allows for more flexible and dynamic structure than traditional systems based on the client-server paradigm.

This thesis discusses the necessity and benefit of incorporating the location and platform agnostic Web Services into mobile agent system. How Web Services technologies could be used to help to standardize the mobile agent system. A framework of Web Service Enabled Mobile Agent System has been presented and discussed along with the detail of the implementation. Evaluation and verification have been conducted to ensure that the experimented system performs as the way it is expected.

Table of Contents

ACKNOWLEDGMENTS ..... **III**

ABSTRACT ..... **IV**

TABLE OF CONTENTS ..... **V**

LIST OF FIGURES ..... **VII**

LIST OF ABBREVIATIONS ..... **IX**

CHAPTER 1. INTRODUCTION **1**

1.1 PROBLEM OVERVIEW 1
1.2 WEB SERVICES ENABLED MOBILE AGENT SYSTEM 2
1.3 THESIS STATEMENT 3
1.4 THESIS OUTLINE 4

CHAPTER 2. WEB SERVICES **6**

2.1 EVOLUTION OF DISTRIBUTED COMPUTING 6
2.2 WEB SERVICES 12
2.3 TECHNOLOGIES AND STANDARDS 16

CHAPTER 3. MOBILE AGENT SYSTEM **25**

3.1 INTRODUCTION 25
3.2 ADVANTAGES OF MOBILE AGENT PARADIGM AND ITS APPLICATION 29
3.3 COMMON MOBILE AGENT INFRASTRUCTURE 31
3.4 STANDARDIZATION 35
3.5 COMPONENT-BASED MOBILE AGENT SYSTEM 38
3.6 THE FUTURE 41

CHAPTER 4. WEB SERVICE ENABLED MOBILE AGENT SYSTEM **42**

4.1 SOAP AS COMMUNICATION MECHANISM 43
4.2 WSDL AND UDDI – SERVICE DIRECTORY 44
4.3 MOBILE WEB SERVICE 45
4.4 FRAMEWORK 46

CHAPTER 5. IMPLEMENTATION **53**

5.1 ARCHITECTURAL CONSIDERATION 53
5.2 ARCHITECTURE 54
5.3 IMPLEMENTATION 57
5.3.1 Programming Language 57
5.3.2 MobilityAxisServlet 58

5.3.3 Component-based Mobility 59
5.3.4 SOAP engine AXIS 64
5.3.5 Web Service Enabled Mobile Agent 67
5.3.6 Service Components 70
5.3.7 Web Service Publish 72
5.3.8 Agent Service Registry – a UDDI Registry 74
5.4 EVALUATION 74
CHAPTER 6. CONCLUSION 77
6.1 THESIS SUMMARY 77
6.2 THESIS CONTRIBUTIONS 77
6.3 FUTURE DIRECTIONS 78
APPENDIX A. SOAP MESSAGE 79
APPENDIX B. WSDL FILE FOR STOCKQUOTESERVICE 80
APPENDIX C. SERVICE INTERFACE DEFINITION 82
APPENDIX D. SERVICE IMPLEMENTATION DEFINITION 83
APPENDIX E. STOCKQUOTESERVICE CLASS 84
REFERENCES 88

List of Figures

Figure — Page

FIGURE 1. MIDDLEWARE IN DISTRIBUTED COMPUTING 7
FIGURE 2. COMPLEXITY OF ENTERPRISE SYSTEM 9
FIGURE 3. WEB SERVICES HELPS THE INTEGRATION 12
FIGURE 4. WEB SERVICES ROLES AND OPERATIONS 13
FIGURE 5. WEB SERVICES CONCEPTUAL STACK 15
FIGURE 6. THE SOAP MESSAGE 17
FIGURE 7. XML MESSAGING USING SOAP 18
FIGURE 8. UDDI DATA STRUCTURES 24
FIGURE 9. MOBILE CODE ABSTRACTION 26
FIGURE 10. FIPA AGENT REFERENCE MODEL 32
FIGURE 11. COMPARISON OF MOBILE AGENTS AND INTELLIGENT AGENTS 37
FIGURE 12. PLATFORM-BASED VS. APPLICATION-BASED MOBILE AGENT SYSTEMS 40
FIGURE 13. FIPA REFERENCE MODEL WITH WEB SERVICES COMPONENTS 48
FIGURE 14. UDDI DATA STRUCTURES IN AGENT CONTEXT 49
FIGURE 15. OVERVIEW OF THE IMPLEMENTATION 55
FIGURE 16. MOBILITYAXISSERVLET 58
FIGURE 17. INTERACTIONS BETWEEN THE MOBILITY COMPONENT AND APPLICATION 61
FIGURE 18. THE MOBILITY COMPONENT. 63
FIGURE 19. AXIS ENGINE ARCHITECTURE 65
FIGURE 20. STOCKQUOTESERVICE DEPLOYMENT DESCRIPTOR 66
FIGURE 21. STCOKQUOTESERVICE UNDPELOYMENT DESCRIPTOR 67
FIGURE 22. WEB SERVICES ENABLED AGENT INTERFACE 68
FIGURE 23. AGENT CALLBACK METHODS 69
FIGURE 24. AGENT SERVICE DEPLOYMENT AND PUBLISHING 70
FIGURE 25. AGENTLIFCYCLELISTENSER INTERFACE 71
FIGURE 26. WSDL TO UDDI MAPPING 73
FIGURE 28. SOAP REQUEST FOR SERVICE 79

FIGURE 29. SOAP RESPONSE 79
FIGURE 30. STOCKQUOTESERVICE WSDL, PART 1 80
FIGURE 31. STOCKQUOTESERVICE WSDL, PART 2 81
FIGURE 32. STOCKQUOTESERVICEINTERFACE 82
FIGURE 33. STOCKQUOTESERVICE IMPLEMENTATION DEFINITION 83

List of Abbreviations

CORBA	Common Object Request Broker Architecture
MASIF	Mobile Agent System Interoperability Facility
FIPA	Foundation for Intelligent Physical Agents
API	Application Programming Interfaces
RPC	Remote Procedure Call
MOM	Message-Oriented Middleware
ORB	Object Request Broker
RMI	Remote Method Invocation
EAI	Enterprise Application Integration
XML	Extensible Makeup Language
SOAP	Simple Object Access Protocol
WSDL	Web Services Description Language
UDDI	Universal Description, Discovery and Integration
ACL	Agent Communication Language
COM	Component Object Model
DCOM	Distributed Component Object Model
SOA	Service-Oriented Architecture

Chapter 1 . Introduction

1.1 Problem Overview

With the widespread of deployment of distributed systems, the management, interoperability and integration of these systems have become challenging problems. People research and develop new technologies to cope with these problems. One of the fruits of the continuous evolution of *distributed computing* in last decade is the **Web Services**. The other is the **Mobile/Multi Agent system (MAS)**, which provides alternative computing paradigm vs. traditional client-server paradigm.

Web Services offer an evolution of the internet-standards based distributed computing model, an evolution in the way of architecting, designing, implementing, and deploying e-business and integration solution. Web Services foster a trend from tightly coupled, rigid, and static solutions that focus on implementation technologies, to loosely coupled, flexible and dynamic solutions focus on system interoperability, dynamic business models, enabling dynamic integration for both new and existing applications [15]. It enables the services (programs or applications) to be location and platform agnostic.

On the other hand, Mobile Agent is considered as a very promising and essential technology in the development of distributed software application because of its mobility to move across network, directly interact with local resource, and other mobile agents on behalf of the remote user and authority. Mobile agent allows for

more flexible and dynamic structure than traditional systems based on the client-server paradigm. It cherishes several distinguish advantages over client-server paradigm, but hasn't become as a mainstream technology due to its immaturity [4]. Particularly, interoperability and standardization of mobile agent system are one of those. Even though, it has been a very active research area and a lot have been done in last decade, it still needs more work before it is widely deployed in industrials.

There are incentives to incorporate Web Services technologies into mobile/agent systems. Is it possible to combine the mobility and intelligence provided by mobile agent with the location and platform agnostic features of Web Services? How can the technologies to build the Web Services be used to architect and standardize (mobile) agent system? How could the mobile agent expose itself as a web service? What is the implication when the mobile agent system is capable of deploying the incoming agents as web services?

1.2 Web Services Enabled Mobile Agent System

So far, to my best knowledge, there is no research work has ever been done relating the (Mobile) agent systems with the Web Services. The research in these two areas has been carried out respectively for years without crossing. Our proposed ***Web Services Enabled Mobile Agent System (WS-MAS)*** is a framework of mobile agent system that encapsulates the emerging Web Services technologies. The Web Services technologies are used to help architect (mobile) agent system functional modules and components and provide the potential to standardize the (mobile) agent system.

Particularly, SOAP could help to standardize the communication and collaboration among agents, agent systems and external systems; WSDL and UDDI together can be used for agent service description and directory service. As adopting Web Services technologies, the mobile agent systems would benefit the great flexibility and interoperability that Web Services endorse.

On the other hand, the mobile intelligent agent systems are Web Services ready so they can consume web services provided by other service providers. Furthermore, the proposed system allow agent to expose itself as a web service to provide service for others, or to aid in dynamical deployment of new web service by taking advantage of its mobility.

1.3 Thesis Statement

1. Argue the necessity and the benefit of incorporating Web Services technologies into mobile agent system. The emerging technologies of Web Services would be good candidates used to help architect and standardize the (mobile) agent systems.

2. Propose a framework of Web Service Enabled Mobile Agent System. Describe the architecture and its modular design. The rationale and justification are given.

3. Present the detail implementation of the proposed system. The demo applications evaluate and verify the system design and its implementation.

Statement 1 conclusion is based on reviewing the state of art of current mobile agent and Web Services technologies. Statement 2 addresses the issues that rise while incorporating the Web Services technologies into mobile agent system.

How Web Service technologies could be used to standardize the mobile agent system is addressed. Statement 3 focuses on the development of some enabling components we have developed. Details how the proposed system is constructed and built through the development works and integration of various subsystems from different vendors.

1.4 Thesis Outline

The rest of the thesis, Chapter two and Chapter three give the state of art of Web Services and mobile/agent technology respectively.

Chapter two discusses the evolution of distributed computing, the emerging of Web Services, and the historical imperatives driving us towards it. Then discusses what is the Web Services and the advantages claimed for this technology. Finally, underneath technologies, the architecture and role models that IBM uses to depict the relationship of Web Services components have been reviewed.

Chapter three introduces the Mobile/Agent system, a new distributed computing paradigm and its advantages over traditional client-server paradigm. The common architecture of Mobile Agent System and its necessary components have been reviewed. The standardization effort has been lengthily discussed. Some modern Mobile Agent Systems, its implementation and applications have also been given.

With the foundation and better understanding built in previous chapters, Chapter four argues the necessity and benefit of embracing Web Service technologies into mobile agent system, argues Web Services technologies would be good

candidates used to help architect and standardize the (mobile) agent systems. A framework of Web Service Enabled Mobile Agent System is presented and discussed. The system architecture, its indispensable components as well as the rationality and implication are also given.

As the prove-of-concept, Chapter five presents an implementation of the proposed system. This chapter details the architecture of the materialized system, the integration of software components and applications from various venders, and the technical implementation – major enabling components, supporting modules we have developed. Finally, one demo application is present to evaluate and verify the system design and implementation.

Chapter six summarizes our research work and contributions, proposes some directions for future work.

Chapter 2 . Web Services

Web Service is the result of continuous evolution of distributed computing. Web Services is an evolution not a revolution in term of technologies underneath it. To fully understand the promising nature of Web Services technologies, we must first understand the driving forces and issues that influenced its evolution.

2.1 Evolution of Distributed Computing

The root of Web Services could trace back to the *Component Architecture* that came into scene in 1980s. It tried to solve one problem: improve software engineering productivity, reusability, and flexibility. The component architecture allows the code to be broken down into independently compiled units (components) which communicate via the infrastructure provided by the underneath operating system, middleware or propriety system. This opened up the possibility to make use of components that written by other developers in more efficient way without exposing the source code. It also allows people to quickly pull the pieces from several sources to build up new application, dramatically cut the development cycle [30]. This technology spreads to other areas soon.

With the advent of computer network (further with the advent of Internet), we make it possible that programs execute on separate machines to interact. With compatible component infrastructure, components not only communicate with each other on the same computer, but also with other components on remote

machines across networks. Now these components are not only used in applications, but also widely used to handle communication between client programs and databases, or between web servers and business applications. Distributed component architecture enabled the rapid development of complex, distributed applications. The software that manages the communication in these systems is what we call *component middleware*.

Middleware services are sets of distributed software that exist between the application and the operating system and the network services on a system node in the network (see Figure 1).

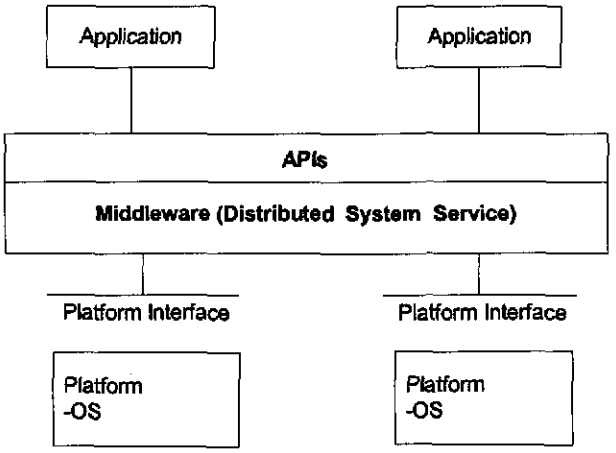

Figure 1. Middleware in Distributed Computing

Middleware services provide a more functional set of Application Programming Interfaces (API) than the operating system and network services to allow an application to [31]:

. locate transparently across the network, providing interaction with another application or service

. be independent from network services .

be reliable and available and extensible

. scale up in capacity without losing functions

The services that Middleware components provide may take different forms. The common seen forms are:

Remote Procedure Call (RPC), which enable the logic of an application to be distributed across the network. Program logics on remote systems can be executed as simply as calling a local routine.

Message-Oriented Middleware (MOM), which is a much more disconnected architecture because it allows the disconnection of software not simply across space, but also over time. It provides program-to-program data exchange, enabling the creation of distributed applications. Message queuing, as the name implies, provides a mechanism for messages from one program to be queued, and then handled by another program at a later time. MOM is analogous to email in the sense it is asynchronous and requires the recipients of messages to interpret their meaning and to take appropriate action.

Object Request Brokers (ORB), which enable the objects that comprise an application to be distributed and shared across heterogeneous networks.

As the distributed computing technologies evolved during 1990s, to facilitate the development and compatibility of the distributed components, the Object Management Group (OMG) developed the specification of CORBA (the Common Object Request Broker Architecture) in 1991, and Microsoft developed the distributed version their COM (Component Object Module) component architecture, DCOM, in 1995. The counterpart introduced by Sun in 1995 for Java platform is RMI (Remote Method Invocation).

However, middleware is not a panacea. It still has its limits. The technologies mentioned above are tumbling to meet the demands of today's business. With the emergence of Internet and ubiquitous computing, what we need today are technologies that are featured as location and platform agnostic, technologies that won't depend on platform and language and technologies provide great flexibility and interoperability. Web Services inherits from Component technology yet evolves to meet more these demands from information technology world. Web Services promises to promote system *flexibility*, *integration* and *interoperability*. As an example, Web Services provides superb solution for Enterprise Application Integration (EAI) and Enterprise-to-Enterprise (or called B-2-B) integration [19]. Let us look at some business scenarios that may help us understand the problems that our business is facing today.

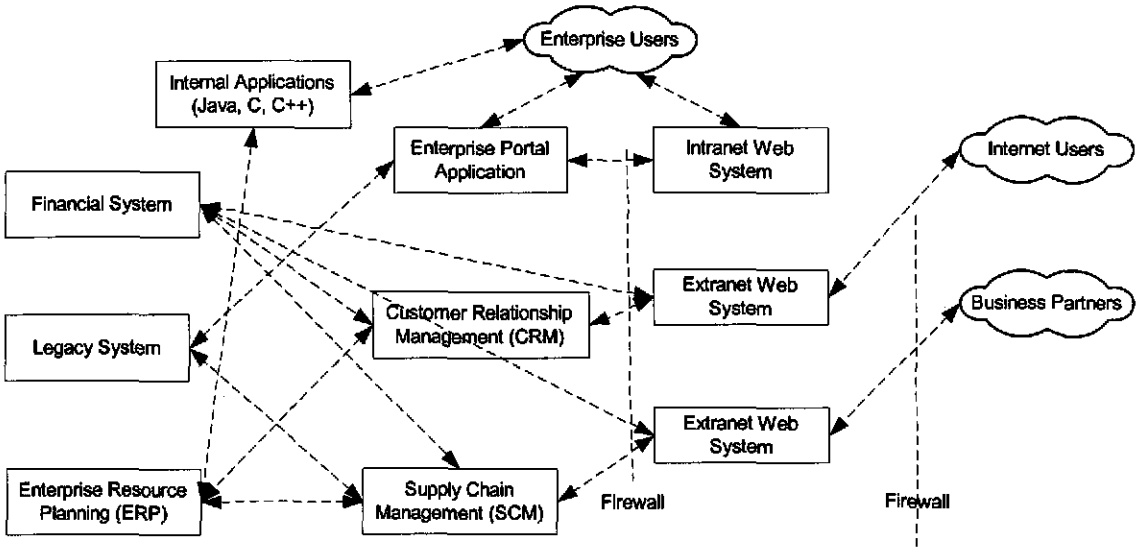

Figure 2. Complexity of Enterprise System

In today's business world where merge and acquisition happen frequently, most companies have an environment of disparate legacy systems, applications and

data sources which typically interact by a maze of interconnection (see Figure 2). Some of the nodes (applications or systems) provide services independently as designed without consideration of interconnection with other sources or are self-contained and may be deployed in different departments. The system and application integration here is necessary, the data exchange and application-to-application communication play an important role to achieve the business efficiency.

In the Net economy era, companies need web presence where people with Internet access can search, investigate and order their products. This requires the real time information tied to inventory, billing, and vendors (see Figure 2), so that they can know what is available, when, and for how much. These put tight on companies, requiring their efforts to put all these pieces together that usually are not. The backend legacy systems like billing and inventory are not really well hooked up with Web systems.

Integrating with a business partner is proven more difficult as it is a time-consuming and expensive undertaking that companies can not switch partners as they desire when business changes. This usually involves the program-to-program or application-to-application integration in which interoperability and flexibility could be big hurdle to overcome because of disparate systems among partners.

The middleware services aforementioned DCOM, CORBA etc, could not adequately solve the problem because of their limitation or provide solutions with overwhelming expense. Middleware product implementations are unique to the

vendors. They either have language, or platform preference or both. DCOM works only for communicating between systems running Windows. CORBA is cross platform, but still requires compatible ORBs on both endpoints and it's very expensive to use because of its complexity. RMI works only between applications developed in Java Technology with coordinated libraries. In [20], authors exploit the reasons why middleware technologies fail to meet today's challenge and not be adopted in Web Services.

Interoperability and integration need to be flexible, and that can only be accomplished by way of standards for defining and exchanging data independent of the implementation of the end points. In the recent decade, we have seen the evolution of new standards designed to improve integration among cooperating companies, partners, and customers. TCP/IP and HTTP define the protocols enable for open and public Internet. XML is used to enable the platform-independent universal messaging and exchanging of business data. Java Technology allows the creation of platform-independent software components, modules and applications, It allows code realize "write once, run anywhere".

These standards enable software/system design based on loose coupling which reduces restriction and eliminates similarity requirement between cooperating systems, and avoids problems that occur as a result of inevitable changes to the software. Although these standards form a solid foundation for building loose-coupling system, they only partially solve the difficult puzzles: how shall we move XML data between systems; how shall we know and agree on what message to send; how should we find the business partners; how to coordinate the work flow

in a solution that involves multiple vendors; how can we reduce the integration time from years, months to weeks or even days. Is it possible to find a solution to ease the communication for application-to-application and program-to-program?

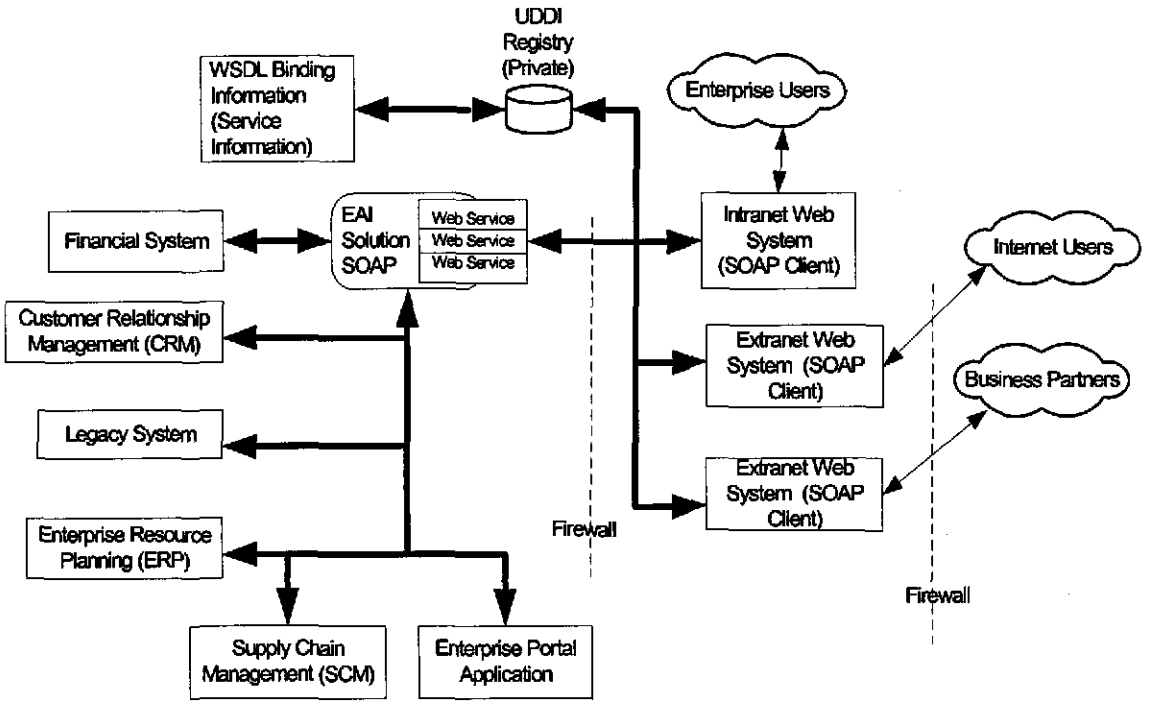

Figure 3. Web Services Helps the Integration

So that applications, systems from different vendors can gracefully be integrated, but not touch their underlying implementation. Web Services technologies (see Figure 3) are intended to provide the answers.

In next section, we discuss what's all about Web Services, how and why Web Services is able to solve the problems we discuss above.

2.2 Web Services

A *Web Service* is an interface that describes a collection of operations that are network-accessible through standardized XML messaging [15]. A Web Service is described using a standard, formal XML notion, called its *service description*. It

covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and location. The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written. This allows and encourages Web Services-based applications to be loosely coupled, component-oriented, cross-technology implementations. They can be used alone or with other Web Services to carry out a complex aggregation or a business transaction.

IBM describes Web Services key architectural principle as the *service-oriented architecture* (SOA) [15]. We brief the SOA here to help us understand the high level architecture of Web Service.

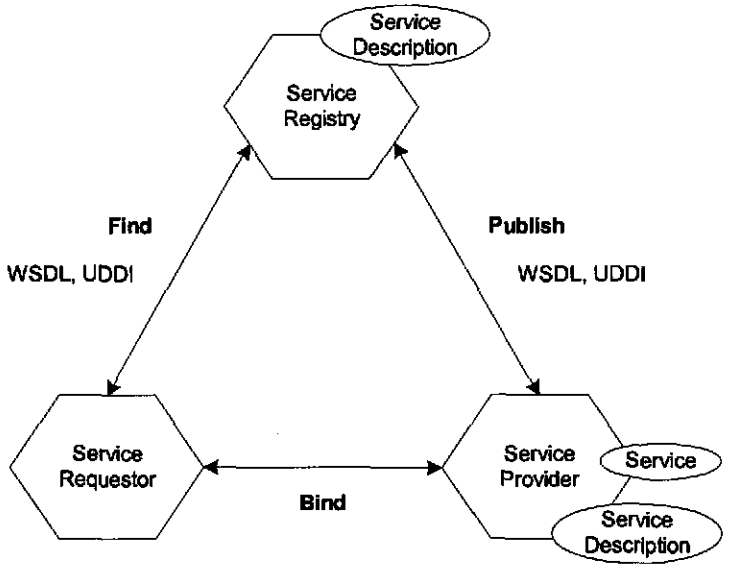

Figure 4. Web Services Roles and Operations

The SOA focuses on the message-based integration of components in a network rather than the details of individual component implementation. The SOA (Figure 4) relates three component roles, service provider, service registry, service requestor and three operations between the roles in support of dynamic, automated discovery and use of services. The operations involve publish, find and bind.

Service registry is a searchable registry of service descriptions where *services provider* can publish their service descriptions. *Service requestors* find services and obtain the binding information, which is the service descriptions for services and finally invoke the service.

Service Provide. A service provider hosts a network-accessible Web service software module. The service provider defines a service description for the Web service and publishes it to a service requestor or service registry.

Service Requestor. The service requestor uses a find operation to retrieve the service description locally or from the service registry and uses the service description to bind with the service provider and invoke or interact with the Web service implementation.

Service Registry. The service registry is a searchable registry of service descriptions where services providers can publish their service descriptions. For statically bound service requestors, the service registry is an optional role in SOA, as a service provider can send the description directly to service requestors. Besides a service registry, service requestors can obtain a service description from other sources, such as a local file, FTP site, Web site etc.

Service. Where a Web Service is an interface described by a service description, its implementation is the service. A service is a software module that is deployed on network-accessible platform provided by the service provider. It exists to be interacted with or invoked by a service requestor. It also can act with dual roles, function as a requestor if it needs to use other Web services in its implementation.

Service Description. The service description contains the details of the interface and implementation of the service. This includes its data types, operations, binding information, and service (network) location. To facilitate the discovery by service requestor, it may also include categorization and other metadata. The service description might be made available to service requestors by publishing to a service registry or FTP site, Web site.

In the next part, we examine Web Service stack and what are the technologies underneath it.

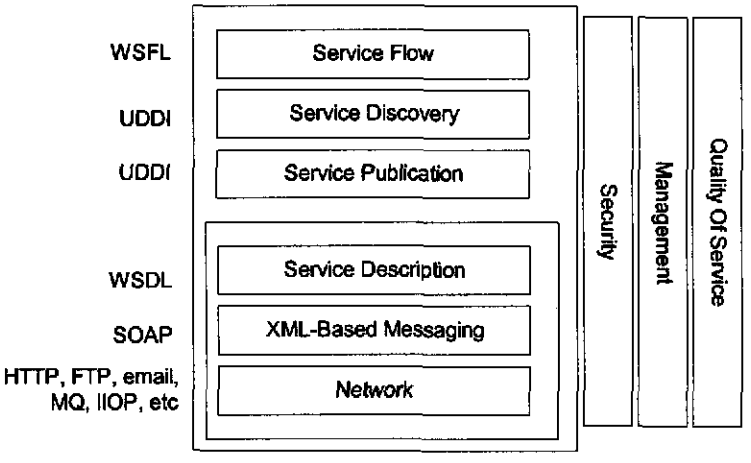

Figure 5. Web Services Conceptual Stack

2.3 Technologies and Standards

A variety of technologies are deployed in Web Services. Figure 5 depicts the Web Service architectural layers and the technologies used to build respective layer. Like network architectural protocol stack, Web Services could be presented as stack that embraces standards at each level [15]. The upper layers build upon the capabilities provided by the lower layers. The vertical towers represent requirements that must be addressed at very level of the layer. The text at the left represents the standard technologies that apply at that layer.

The bottom layer is the network, which provides the basis for communicating between Web Service and service requestor. This layer can represent any number of network protocols: HTTP, SMTP, FTP, Message Queuing (MQ), Remote Method Invocation (RMI), e-mail, and so on. Because of its ubiquity, HTTP is the de facto standard network protocol for Internet-accessible Web Services. For Web Services being deployed and consumed within an Intranet, the alternative network technologies may be used upon the agreement. The network technologies can be chosen based on other factors, such as security, availability, reliability and performance. Specially, one particular may be chosen because of trying to capitalize on existing infrastructure.

The second layer is the XML-based messaging, which provides the mechanism for exchanging the messages between Web Services entities. It's the core layer of Web Services. Given the Web's intrinsically distributed and heterogeneous nature, communication mechanism must be platform-independent, international, secure, and as lightweight as possible. XML (Extensible Message Language) is

no doubt established as standard for information and data encoding for platform independence. The current industry standard for XML messaging is **SOAP (Simple Object Access Protocol).**

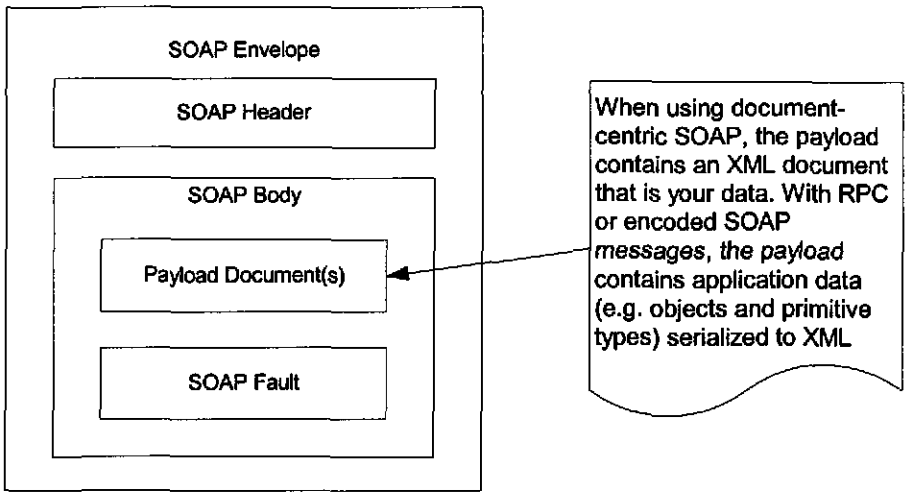

Figure 6. The SOAP Message

SOAP is a simple and lightweight XML-based mechanism for exchanging structure data between network applications. SOAP can be used in combination with or re-enveloped by a variety of network protocols such as HTTP, SMTP, FTP, ACL etc. It can either be used as document-centric messaging or be used as remote procedure call. SOAP consists of three parts (see Figure 6.): an envelope that defines a framework for describing what is in a message, a set of encoding rules for expressing instances of application-defined data type, and a convention for representing remote procedure calls (RPCs) and responses. This structure message may include the information that indicates how recipients should process SOAP message, also may include actors, which indicate a series of intermediaries that process the message parts meant for them and pass on the rest.

Figure 7 illustrates how the network layer, XML-messaging layer (SOAP) and *application form the Web Services architecture.*

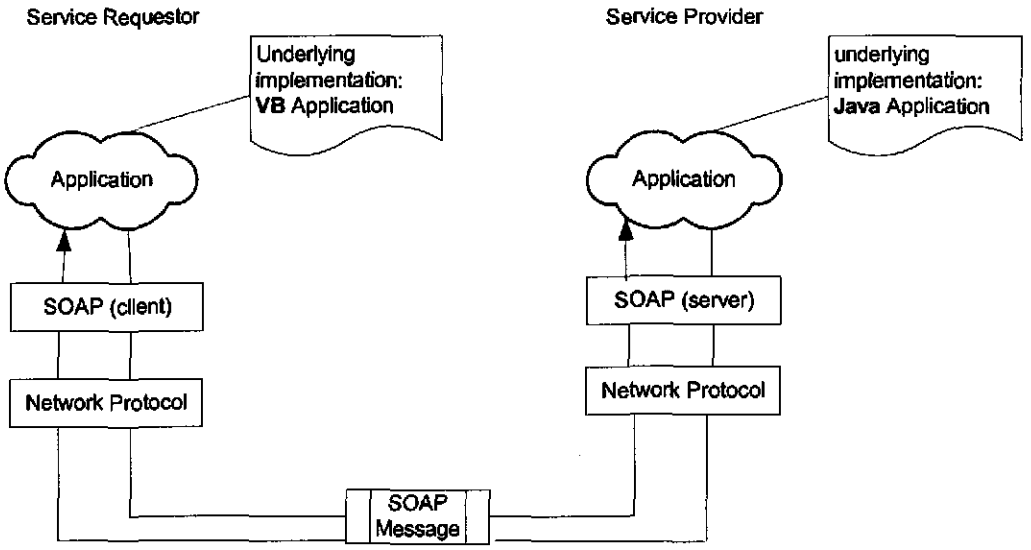

Figure 7. XML Messaging Using SOAP.

In Web Services, the requestor and the provider are required to be able to build, parse the SOAP message and have the ability to communicate over a network. As we will see later, typically a SOAP server runs in a Web application server or an application performs these functions. Alternatively, the application may use a programming language-specific runtime library API to perform these functions. Figure 7 illustrates the interactions between the server requestor and the service provider.

1. A service requestor's application creates a SOAP message. This message is the request that will invoke the intended Web service operation provided by the service provider. The XML document in the body of the message can be either a SOAP RPC request or a document-centric message as indicated in the service description. The service requestor presents this message together

with other necessary information such as the IP address of the service provider to the SOAP infrastructure (a SOAP client runtime or SOAP server). Then this infrastructure interacts with underlying network protocol (for example, Http or SMTP) to send the SOAP message out over network. See Appendix A for an example of request message.

2. The infrastructure delivers the SOAP message to the service provider's SOAP runtime (for example, a SOAP server). The SOAP server routes the request message to the service provider's Web service. The SOAP runtime is responsible for converting the XML message into programming language-specific objects if required by the application. This conversion is governed by the encoding schemas found within the message.
3. The Web services responsible for processing the request message and formulating the response if it has one. The response is also a SOAP message and presents to the SOAP runtime. The SOAP runtime sends the response to the service requestor over the network. Go to Appendix A for an example of response message.
4. The response SOAP message is received by the network infrastructure on the service requestor side. The message is routed through the SOAP infrastructure. Before the response message presents to the requestor application, it may be converted into objects in a target programming language.

The use of SOAP simplifies the development for both the requestor and provider because SOAP effectively hides the details of the implementation of the end

points. The web services implementation could be a legacy COBOL application, a new EJB system, or components written in VB, or C++ as long as it can interpret a request message and return an appropriate response. Similarly, the service provider knows nothing about the implementation of the requestor. What SOAP achieves is that neither the service requestor nor the service provider knows or cares about anything besides the format and content of request and response message.

The request / response exchange can be synchronous (RPC) or asynchronous (document-centric). When the SOAP is used as a remote procedure call where the message contains a method and the arguments, it extends the power of object-oriented programming to web-based remote objects (through HTTP). It can also be used for exchanging documents containing any kind of XML data. It enables complete reuse of code, from systems of any type, both inside your company and among business partners. This makes SOAP one of ideal foundations of Web Services.

The use of SOAP allows software processes to change without requiring changes to the other party. A web service implementation can change with no impact on the users of the service, as long as the request and response messages do not change. The first implementation of the service provider could be a thin wrapper around legacy code, allowing a planned replacement to be developed over time. When the new code is ready, the change is invisible to the users of the service.

The third layer is the Service Description. It is through the service description that the service provider communicates all the specifications for invoking the Web service to the service requestor. It's all about telling service requestor what's the service about, and how to invoke it.

WSDL (Web Service Description Language) is an XML document for describing Web Services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented (PRC) messages. A *complete* WSDL *service description* provides two pieces of information: an application-level service description, or abstract interface, and the specific protocol-dependent details that requestors must follow to access the service at *concrete service end point. This separation accounts for the fact that similar* application-level service functionality is often deployed at different end points with possible different access protocol details. For the detail of WSDL specification, see [21]. For an example of WSDL file, see Appendix B.

Because a Web service is defined as being network-accessible via SOAP and represented by a service description, the first three layers are required to provide or use any Web service. Fourth and fifth layer are about the Service Publication and Service Discovery, which may not be necessary element to use or provide Web service.

Service requestor needs the service description (WSDL document) to invoke the intended service. The simplest way to make it available is the service provider sends a WSDL document directly to a service requestor. This is called direct publication, which is useful for statically bound applications. Alternatively, the

service provider can publish the WSDL document describing the service to a host, which is a local WSDL registry, a private UDDI registry or an UDDI operate node.

The discovery of Web Services includes the acquiring of the service descriptions and the consuming of the descriptions. Acquiring can use a variety of mechanisms. Like publishing Web service descriptions, acquiring Web service descriptions will very depend on how the service description is published and how dynamic the Web Service application is meant to be.

With the direct publishing approach, the service requestor caches the service description at design time for use at runtime. The service description can be statically represented in the program logic, stored in a file or in a simple, local service description repository. Otherwise, service requestor can retrieve a service description at design time or runtime from a service description repository.

As to the present, the **UDDI (Universal Description, Discovery, and Integration)** specification is the wide accepted standard for the publication, discovery, and description of Web service. The UDDI offers a unified and systematic way to find service providers and services through a centralized registry of Web Services that is analogy of a phone directory (yellow page). For detail of the specification, see [23].

An UDDI-complaint registry provides an information framework for describing services exposed by an entity or a business. Using this framework, the description of a service managed by an UDDI registry is information about the

service itself. In order to promote cross-platform service description, the description is rendered in cross-platform XML. UDDI provides two basic specifications that define a service registry's structure and operation: A definition of the formation to provide about each service, and how to encode it. A query and update APIs for the registry that describes how this information can be accessed and updated. For a list of UDDI implementation, see http://www.uddi.org.

In UDDI, the information that makes up a registration consists of four data structure types: businessEntity, businessService, bindingTemplate, and tModel. This division by information type provides simple partitions to assist in the rapid location and understanding of the different information that makes up a registration. Figure 8 shows the four core data types and their *containment* relationship.

These four structure types make up the complete amount of information provided within the UDDI service description framework. Each of these XML structures contains a number of data fields that serve business or technical descriptive purpose [23].

The businessEntity structure represents all known information about a business or entity that publishes descriptive information about entity as well as the services that if offers. Service descriptions and technical information are expressed within a businessEntity by a containment relationship. Services are represented in UDDI by the businessService data structure, and the details of

how and where the service is accessed are provided by one or more nested bindingTemplate structures. A bindingTemplate specifies a network endpoint.

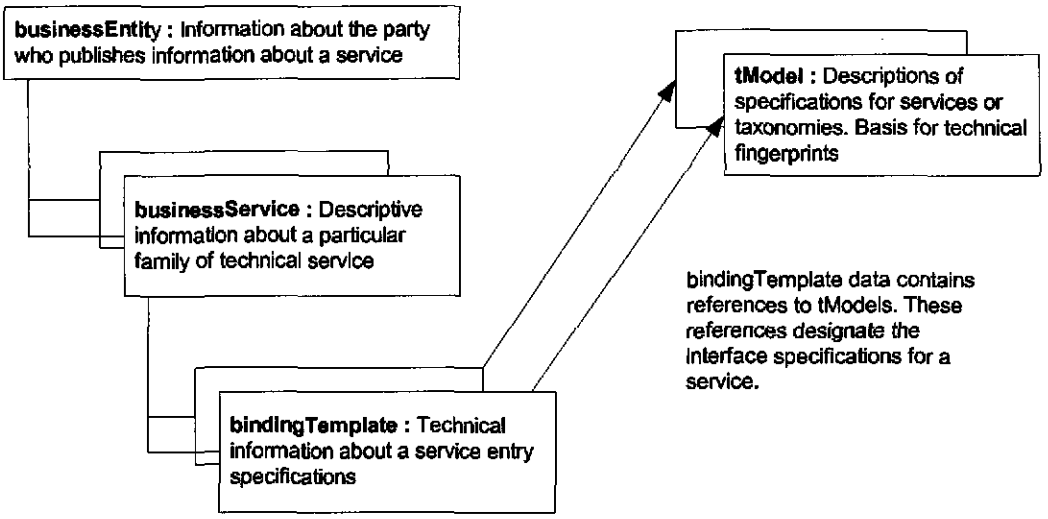

Figure 8. UDDI Data Structures

address and a stack of tModels describing the technical specification of the service.

Registry access is accomplished using a set of standard SOAP APIs for both querying and updating.

Chapter 3 . Mobile Agent System

Far before the advent of Web Services, an alternative paradigm in distributed computing emerged and has been evolved since last decade. Over last few years we have observed the proliferation of Mobile Agent System. Mobile Agent System promises to cope more efficiently and elegantly with a dynamic, heterogeneous, open environment. Contrary to the traditional client-server distributed computing we discussed in previous chapter that is built on the location transparency abstraction, Mobile agent system is characterized as an enabling distributed system by supporting local interaction, mobile logic and data. The Mobile Agent System offers some unique advantages over the client-server model. It allows for much more flexible and dynamic structure than the traditional systems based on client-server paradigm. To fully understand this new paradigm, we review the Mobile/Agent technology and examine the differences between client-server and mobile agent based distributed systems.

3.1 Introduction

Today, most of the deployments of distributed computing systems follow the client-server paradigm. The *server* is defined as a computational entity that provides the services. The *client* requests the execution of theses services by interacting with the server. After the service is executed the result is delivered back to the client. The server therefore provides the knowledge of how to handle the request as well as the necessary resources. Besides client-server, there are

several other paradigms in distributed computing: Remote Evaluation, Code on Demand and Mobile Agent (see Figure 9.).

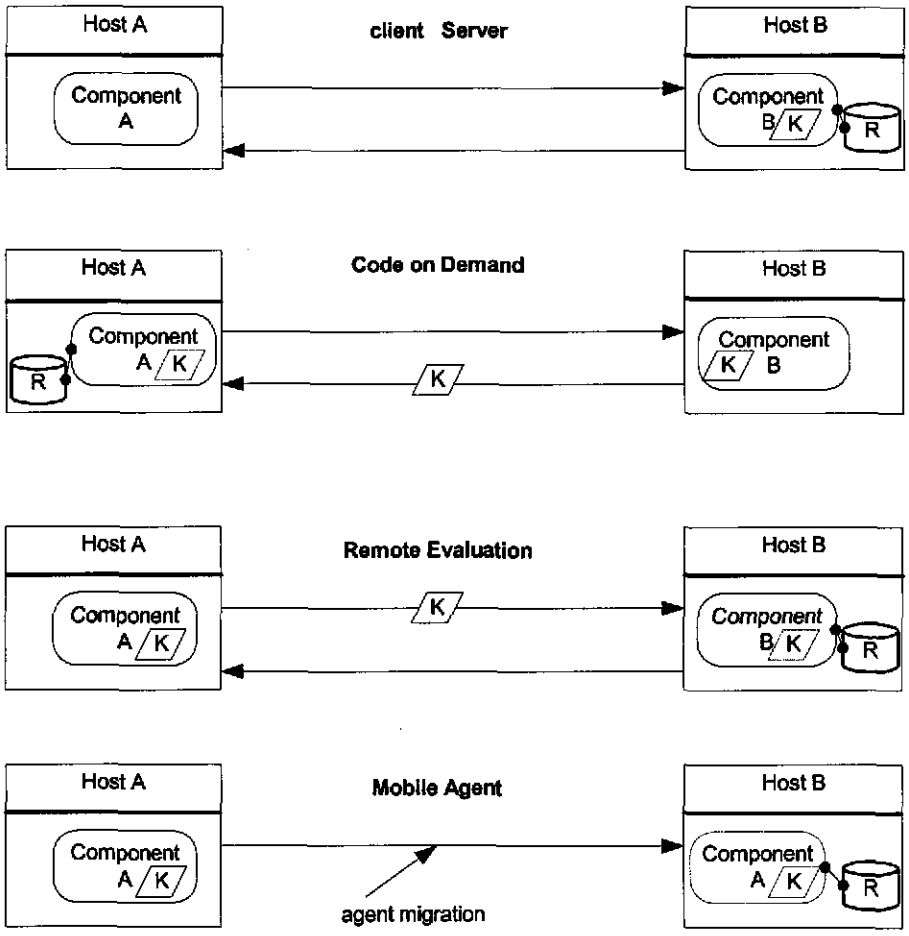

Figure 9. Mobile Code Abstraction

. **Remote Evaluation** (RE): In this paradigm, component A sends instructions specifying how to perform a service to component B. B then executes the request using its resources.

. **Code on Demand** (CoD): In this paradigm, the component A has the resources collocated with itself, but lack of knowledge of how to access and process its resources to perform the task. A sends a request to component B, asking for B to

forward the knowledge. Upon the receipt, A is then able to perform the task. Java Applets fall under this paradigm.

. **Mobile Agent** (MA): In this paradigm, component A has the knowledge to perform its tasks, but has no access to resources that locate in remote place. Instead of forwarding/requesting to another component, A itself migrates to the place and interacts locally with resources. The migration involves the mobility of the entire computational entity, along with its code, logic and state.

The idea of sending programs to and executing them at remote host has been explored for a long time. In 1990, general Magic launched the first commercially available mobile agent platform called Telescript, In 1994 the notion of Mobile Agent was established with the release of white paper by White [29] that described Telescript. Before we give a more formal definition to mobile agent, let's look at the agent definition first.

An agent is an encapsulated software entity with its own states, behavior, thread of control, and an ability to interact and communicate with other entities, including people, other agents, and legacy systems [36].

Typically, agents have been categorized as two kinds. One is so called the intelligent agent, which exhibits intelligent behaviors and has the capability to autonomously conduct works based on its state and information, or shows adaptive behaviors of neural network or other heuristic techniques. This type of agent is usually static, not mobile as with the intelligent capability, it certainly is bigger and less desirable to move around. Agents may be associated and

collaborated to work together to solve a certain problem. The agents with the supporting platform and system constitute the *Multi-Agent Systems* (MAS),

Another type of agent, called Mobile Agent (MA), which is an executing program that can migrate from machine to machine in either heterogeneous or homogeneous network while retaining its state information. On its itinerary, it interacts with other stationary service agents or resource to accomplish its task. The mobile agent characterizes as autonomous, self-adjustable and self-conscious. Mobile agents and its supporting platform constitute the *Mobile Agent System*. Contrary to traditional client-server distributed system or more modern DCOM and CORBA diagram which embody "location transparency", the mobile agent diagram shifts to "local interaction" which brings resource closer to the user in a sense.

Both the mobile agent and intelligent agent technologies are targeting adaptive and flexible co-operation, particularly interoperability between or within distributed systems in a dynamically changing environment. There are a lot of similarities between them, but they are different in their origins, emphases, implementation and the interests of the supporting communities. There are two organizations attempting to standardize the agent technology. One is the Object Management Group (OMG), which represents the interest of Mobile Agent community. The other is the Foundation for Intelligent Physical Agents (FIPA), which represents the interest of Intelligent Agent community. Both of them have developed specifications to promote and support interoperability among agents and agent platforms. As the agent technology gets mature, we have observed in

recent years a lot of efforts trying to integrate these two specifications, consolidate the standardization on agent technology. Some agent systems developed in recent years have accommodated specifications from both organizations. We will discuss more on this later in section 4.

Mobile Agent System is our primary interest and we will focus on it in our research.

3.2 Advantages of Mobile Agent Paradigm and its Application

Generally, there are consensuses that MA exhibits the advantages over traditional client-server computing paradigm even though there haven't produced enough quantitative assessments. In his lengthy PH.D. dissertation, Todd philosophically argued that the contemporary distributed systems built with the location transparency abstraction are fundamentally flawed and that we require new abstraction for distributed computation. The local interaction is better suited to the underlying hardware substrate upon which distributed systems are built [7].

The ability to move the data computation to data source and continue locally is one of the biggest advantages of mobile agents. Another true benefit is really the computing paradigm from which the type of software architecture that can be built. Mobile agent technology allows us to build a much more flexible and dynamic system/application structure than one traditional client-server system allows.

The use of mobile agents for distributed applications has several potential benefits. The following is the summary of those benefits [2][3][28]:

1. Conservation of bandwidth and latency: If the communication between two interacting entities involves a considerable amount of data, it may be beneficial to move one of them closer to the other instead of moving the data between them. The locality of their interaction would decrease the latency and save bandwidth used in communication. It's clear that the use of mobile agent paradigm is only justified that the gain in latency and bandwidth must overcome the cost of sending the mobile agent code.

2. Disconnected operation and mobile computing: Mobile agent can be delegated to perform certain tasks even if the delegating entity does not remain active. So it helps to develop applications for Personal Digital Assistants (PDA) or laptops. Instead of being online for a long period of time, a mobile user may send an agent out to work on his/her behalf while being disconnected, and receive back the agent with the result at some later time.

3. Dynamic deployment and extensibility of service: Mobile agents can be used to dynamically extend capabilities of applications, provide additional services whenever appropriate. The mobility of agent (code) makes the dynamic deployment possible, easy and allows for building the systems that are extremely flexible.

4. Dynamic adaptation and Load balancing: Mobile agents have the ability to adapt dynamically to the change in their environment. They can, for instance, react autonomously to balance the load in the network, or migrate to problematic node as a replica of the service.

Though none of these above strengths are unique to mobile agents, no competing techniques share all these advantages [2]. Traditional client-server technology may accomplish those aforementioned functionalities, but with complexity and overwhelming cost. A mobile agent system provides a single general framework in which a wide range of distributed applications can be implemented efficiently and easily.

3.3 Common Mobile Agent Infrastructure

Agent technology is currently one of the most vibrant and active research areas. According to [1][32], there are over 70 (mobile) agent systems that have been developed over last decade. Applications have been developed in all kind of domains. Even though these systems are implemented in different languages and platforms, they have similarities in architecture and functionalities.

In an agent-based computing scenario, the host needs to provide the infrastructure for (mobile) agents. It acts as a local environment called as agent platform, therefore it not only provides the execution environment for receiving and migrating agent, but also provides other infrastructure such as tracking, location management, security and communication. As an example, Figure 10 represents the FIPA 98 agent reference model, which provides the normative framework within which FIPA Agents exist and operate. The entities contained in the reference model are logic capability sets (that is, services) and do not imply any physical configuration [9]. Additionally, the implementation detail of individual agent platform and agents are the design choices of the individual agent system developers.

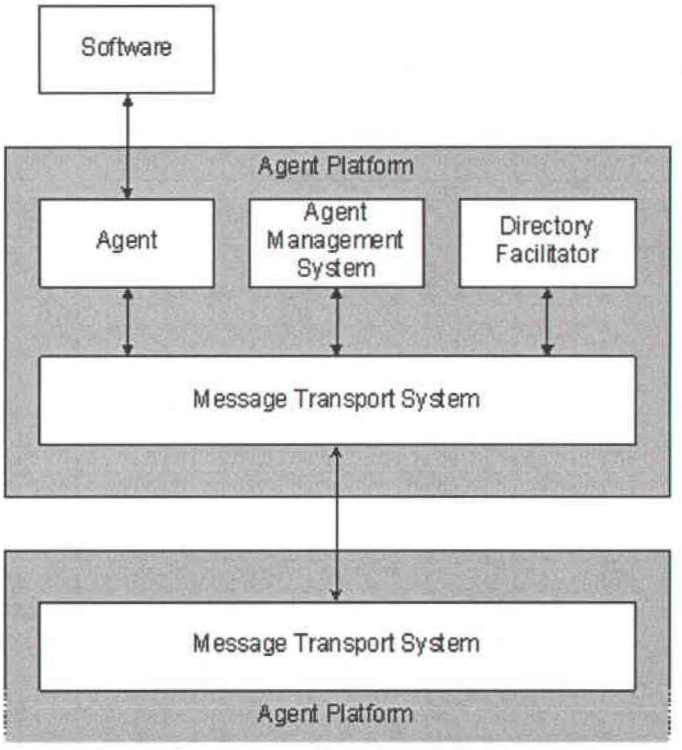

Figure 10. FIPA Agent Reference Model

Generally, to be a mature Mobile Agent System, it has to have the infrastructure to provide following supports:

Mobility support:

It needs to provide an environment for accepting incoming agent, installing and registering the agent and provide an execution environment where the incoming can continue its execution. Also, upon the request or as will of the agent, it packs the state of agent along with the code together, send it over the network to the destination where the agent supposes to go.

Resource management:

The infrastructure communicates with incoming agent to check its authorization, quotas, to prevent it from abusing the system or excessive use of system resource.

Execution support:

The agent platform either supports the creation of agent or provides the environment for the incoming agent to continue its execution upon arrival. To continue the execution to accomplish its tasks, the agent may need the access to necessary runtime libraries and other services provided by platform.

Agent Communication support:

Agent should be able to communicate with other locally residing agents, and also with remote agents and with its owner or creator. For that, the agent environment should support standard communication mechanism and protocol. It is essential for agent to be able collaborate and interact with other agents. The communication has to take place through some mechanisms such as RPC or messaging. It can be well-standardized XML documents or proprietary messages, or Agent Communication Language (ACL), which is widely deployed in Multi-Agent System [10].

Directory and information service:

Agents must be able to check the availability of services either provided by the system or by other agents. They should be able to learn about the local or remote presence of other agents and their services.

Location Management:

As agent moves around from machine to machine to accomplish its mission, the owner or creator needs to retrieve its location. Also, as for communication purpose, other agents need to know where to find, locate the one they are

looking for. The agent platform should have a central registry where it can register and update the location as the agent changes its location.

Security support:

An agent platform must ensure the privacy and integrity of agents and its own infrastructure. For that, it needs means for encryption and decryption of agent code, and it must provide authentication, authorization and access control mechanisms.

Support for Fault Tolerance:

Correct and reliable execution of agents should be guaranteed even when partial failures occur. A graceful exit and remedy should be implemented.

In order to use the existing distributed system infrastructure, it would be desirable to have interoperability mechanisms that connect agent platforms to middleware concepts like CORBA or other emerging infrastructures.

As some applications building on agent platforms may require further, more specific services, the agent platform should have a robust framework that allows additional service components can be easily integrated into platform.

Besides these mainly local tasks of an agent environment, there are tasks that require cooperation among several distributed agent platforms and hence necessitate standard protocols and interfaces. Also as whole of agent societies, there is need for some global services such as location of agents, message forwarding, broke service.

3.4 Standardization

Mobile agent is a relatively new technology, but we have observed proliferation of implementations [1], such as AgentTCL, MOA, Grasshopper [33]. These systems differ widely in implementation. That thereby impedes interoperability, rapid proliferation of agent technology, and the growth of the industry. To promote interoperability and system diversity, some aspects of agent technology must be standardized.

There are currently two important agent standardization efforts which are attempting to support interoperability between agents on different types of agent platform: OMG's MASIF and FIPA.

Object Management Group (OMG) is an international organization founded in 1989, which currently has over 800 members including information system vendors, software developers and users. The OMG promotes the theory and practice of object-oriented technology in software development. It was formed to help the establishment of industry guidelines and detailed object management specifications to provide a common framework for application development. Primary goals are the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments. One of its outstanding achievements is the CORBA (Common Object Request Broker Architecture) [34]. In 1995, OMG started working on a standard, called Mobile Agent Facility (MAF) in order to promote interoperability among agent platforms. The current version is called MASIF (Mobile Agent System Interoperability Facility) that came out in 2000 [37].

The Foundation for Intelligent Physical Agents (FIPA) [10] is an international organization formed in 1996 that is dedicated to promoting the industry of intelligent agents rather than with mobile ones by openly developing specifications supporting interoperability among agents and agent-based application. The current specification is FIPA 2000, which covers all aspects of agent and agent based application from application, abstract architecture, and agent management to agent message transport, especially it covers detail of agent communication language where MASIF left out.

The major difference between mobile agents and intelligent agents and the corresponding OMG MASIF and FIPA specifications is that a mobile agent usually uses a low level programming language, while the intelligent agent typically has a speech act alike communication language and a predicate logic based content language.

OMG MASIF aims at enabling mobile agents to migrate between agent systems of the same profile (language, agent system type, authentication type and serialization methods) via standardized CORBA IDL interfaces, and achieves a certain degree of interoperability between mobile agent platforms of different manufactures. Language interoperability for active objects that carry "continuations" around is very difficult, and it is not addressed by MASIF. In order to address interoperability concerns, the interfaces have been defined only at agent system rather that at the agent level. A MASIF-compliant agent platform can be accessed via two standardized interfaces that are specified by means of the OMG's Interface Definition Language (IDL): *MAFAgentSystem* and

MAFFinder. These interfaces provide fundamental operations for agent management, agent tracking, and agent transport. MASIF adopts a mobile agent paradigm that is more appropriate in situations where dynamic and autonomous swapping, replacement, modification, and updating of application components are required.

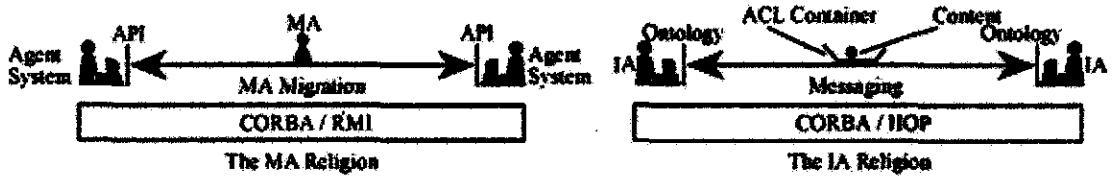

Figure 11. Comparison of Mobile Agents and Intelligent Agents

FIPA works on enabling the intelligent agent interoperability via standardized agent communication and content languages. Besides the generic communication framework, FIPA also specifies ontology and negotiation protocol to support interoperability in specific application areas (travel assistance, multimedia entertainment, network service provisioning, manufacturing etc.). FIPA adopts an agent communication paradigm, which can better express the nature of cooperation and is more suitable for integration with other AI technologies. Within an agent communication paradigm, co-operation is realized via the Agent Communication Language (ACL), the content language and the ontology that identifies the set of basic concepts (taxonomy) used in the message content for co-operations. ACL is at a high level of abstraction that the classic representation of APIs in distributed object technologies such as CORBA, RMI, DCOM rely heavily on exact syntactical matching of service requests to service provider interfaces. However, ACL messages cover a lot of information

that may not be necessary for specific interaction scenarios. For instance, by providing some "specialized" operations for accessing fundamental capabilities of an agent platform, e.g. for requesting the state of a registered agent, this information overhead could be avoided. The invocation of an IDL method getState(agentID) can be performed much easier and faster than the generation of a corresponding ACL request, including a content message. Of course, these specialized operations limit the flexibility of an agent interface, compared to the generic method message as specified in FIPA. A compromise of both solutions seems to be desirable.

As agent technology grows mature, there is effort that brings these two standards together. The liaison has been established [35]. Particularly, FIPA proposed Agent Management Support for Mobility Specification in 2000 [11], thus built a bridge between the intelligent and mobile agent technology. On OMG side, OMG Agent Platform Special Interest Group is working on new mobile agent standards that should deal with an integration of the current MASIF submission and the FIPA standards [36]. One benefit of this co-operation could be ACL support in MASIF-compliant mobile agent environments. On the other hand, the FIPA standards could benefit from the mobility aspect handled by MASIF.

3.5 Component-based Mobile Agent System

The ubiquitous availability of agent environments is necessary for any successful usage of the mobile agent paradigm. Many mobile agent systems have been developed, but few commercial mobile agent systems exist. One of the hurdles

that harm the spread and acceptance of this technology is the monolithic approach to build the mobile agent systems and applications.

Most of the existing Mobile Agent Systems are built around the mobile-agent distributed platform as an extension of the host's operating system, focusing too much on mobile agents and associated issues, such as mobility, agent lifecycle, security, and coordination etc [4]. It's very difficult to develop an application that uses mobile agents. This monolithic approach forces the development to be centered on the agents, many times requiring the application itself to be a special type of agent – stationary agent, or have to provide some special interface agents (service agents) to bridge the application and the incoming agents. These agents have to know how to communicate with the mobile agents and with the application. This diagram of platform-based mobile agent system makes it very difficult to be deployed in ordinary distributed applications, which developed by traditional object-oriented, component-based along other technologies and methodologies.

Object-oriented, component-based software development methodology is mature and has been proven to build agile, flexible and reliable distributed applications. It would allow more flexibility for developers to build mobile agent system that constituted by a set of mobility components.

In [4][5], the researchers at University of Coimbra give a new approach that is called **Application Centric Mobile Agent Systems (ACMAS)**, which is that there are no agent platforms. Instead, agents arrive and leave from the application that they are part of, not from agent platforms. The application

becomes agent-enabled by incorporating well-defined software components into it. Theses components give the application the capability of sending, receiving and interacting with mobile agents (see figure 12). The application knows the interface of agents and the agents know how to interact with the application. The application itself is developed using the current industry best-practice software methods and becomes agent-enabled by integrating the mobility components.

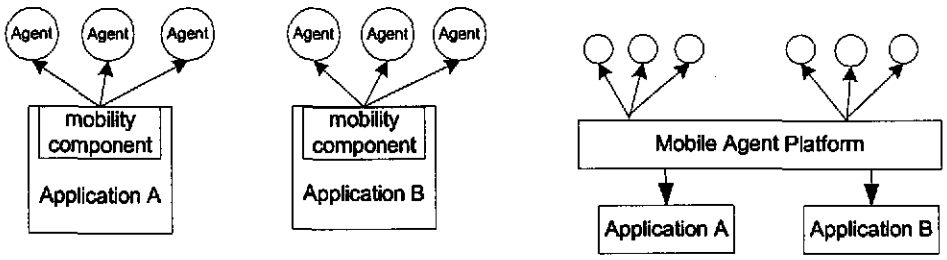

Figure 12. Platform-based vs. Application-based Mobile Agent Systems

In ACMAS, the application is developed using different kinds of components. Mobile-agent supporting components provide the infrastructure of mobile agent. These components provide the functionalities typically found in agent platforms: mobility support, agent tracking, inter-agent communication mechanisms, security and others. We will give more detail later in Chapter 5. Third party off-the-shelves are components that are commercial available from software vendors and can be used for all kinds of system. There are vast varieties of components available, so may save your quite lot of time to develop them yourself. Domain specific components are modules that you may have to write, in the context of the application domain be considered.

3.6 The future

There have been a lot Mobile Agent Systems exist in both academic and industrial areas. Applications have been developed in all kinds of domains [1][8][12][13][14][28]. One of widely applied areas is the telecommunications and network management (NM) where see the potential of mobile agents. MA-based applications have been developed to use mobile agents to perform management tasks that deal with very large amounts of data, distributed over the nodes of GSM networks [26][27].

Even though many mobile agent systems have been used in academic and a few in industrials, mobile agent technology still has not caught in mainstream. It has both technical and non-technical reasons [2]. First, it's not imperative to deploy this technology in distributed environment. All problems so far could be solved in traditional client-server model with endeavors. Second, a lot more works need to be done in some areas before it's acceptable to commercially deploy in industrials. The areas include security, interoperability, reliability, transaction support, etc. Third, as we discuss in previous section, the monolithic approach to build MA applications prevents it from being accepted by developers. One last, would be the standardization and the integration with other non-agent based applications, systems, and middleware.

Nevertheless, the advantage of MA technology is obvious. It's predictable that this technology will be more deployed when it comes to mature. If not dominates over the traditional distributed systems, it complements the existing distributed computing systems.

Chapter 4 . Web Service Enabled Mobile Agent System

Web Services is a quite new distributed computing technology with very short history comparing with others. But there are incentive and momentum that led it quickly accepted by industrials. Web Services technologies are used to promote the service integration and interoperability among heterogeneous program-to-program or application-to-application scenarios. It is natural to see Web Services fit into agent systems where the service invocation and interaction between agents are among heterogeneous environment. So far, there is very little research about incorporating Web Services into agent systems, even less about how mobile agent could help the dynamical deployment of Web Services, and the potential to build a very adaptive Web Services infrastructure via mobile agent technology.

Incorporating Web Services technologies into agent systems is imperative as we see more and more deployment of agent systems or agent applications in enterprise environment. If agent systems and applications are going to be widely accepted and deployed in industrials, they have to get out of current state of practice that agent systems are often closed systems that agents only communicate with other agents in their own societies, but not with other middleware or non-agent systems. Consequentially, the more agent systems deployed in enterprise environment, telecommunication industrials, the more the agent systems will involve with integration with other legacy systems, web

systems. It's evitable that agent systems will adopt Web Services technologies to survive.

Web Services technologies may be used to help building agent system infrastructure, such as agent communication, directory service, service description, service publication and discovery.

On the other hand, if mobile agent can be deployed as a Web Service in enterprise environment, it may add rich functionalities to Web Service applications where the deployment of web service becomes more flexible and adaptive. The combination of Web Services, intelligence and mobility of agents allows the systems and applications have more flexible and responsive service infrastructure.

In this chapter, we mainly come to:

1. Discuss the Web Services technologies that could be used in agent technology.
2. Discuss the benefit of incorporating Web Services into mobile agent systems.
3. Propose a framework of Web Services Enabled Mobile Agent System and discuss its architecture and major components.

4.1 SOAP as Communication Mechanism

We discussed in previous chapter that the mobile agent or the stationary agent might need to interact with other agents on a different system. The agent typically uses some communication transport mechanisms such as Remote Procedure Call (RPC), messaging with some high level languages like ACL to achieve that. The RPC could be one of forms of CORBA, DCOM, and RMI. CORBA was

chosen in both MAFIF and FIPA because of its language and platform independent characteristics that promote interoperability. CORBA relies on a generic, language-independent Interface Definition Language (IDL) to define the service interfaces. All implementation specific aspects are hidden and not handled by CORBA. Also, CORBA defines a platform-independent protocol, Internet Inter-ORB Protocol (IIOP) for remote interaction between distributed objects. As we discussed in chapter 2, SOAP has all of those characteristics that promote interoperability with even greater flexibility and extensibility. It's far superior to CORBA and more suitable for Internet computing which is now the dominant distributed computing paradigm [20].

4.2 WSDL and UDDI – Service Directory

Agent platforms (APs) need to provide the directory and information service (see section 3.3 common mobile agent infrastructure) to advertise the services that available to agents or other service entities. For example, In FIPA a Directory Facilitator (DF) is a mandatory component of an AP that provides a yellow page directory service to agents. Agents that wish other agents to use its service may register (publish) their services with the DF. Agents can query the DF to find out what services are offered by other agents.

The services registered to DF may be implemented as agents that are accessed via method invocation, using programming interfaces such as those provided in Java, C++, or IDL [9]. To be interoperable across platforms, the services usually are defined by CORBA's IDL, which hides the implementation details. Web Services SOAP is superior alternative to CORBA. Web Services' WSDL is very

good alternatives for service description. SOAP and WSDL together provide a very good mechanism to allow service requestor to dynamically invoke services. UDDI is good candidate for directory service.

WSDL is an XML format document for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate [21]. It recognizes the need for rich type system for describing message formats, and supports the XML Schemas specification (XSD) as its canonical type system, allows using other type definition languages via extensibility. See Appendix B for an example of WSDL describing an agent service:

The UDDI data structures provide a framework for the description of basic business and service information, and architect an extensible mechanism to provide detailed service information using any standard description language [22]. The extensibility of such data structure provides very flexible way to allow almost any kind of industry specific domain to use it.

4.3 Mobile Web Service

There are two main benefits to incorporate Web Service into agent systems.

One is that the widely accepted and established Web Service technologies may facilitate the standardization of agent and agent system.

The other is that with Web Service enabled, agent and agent system will act like any other Web Service enabled entities, enjoy the great flexibility and

interoperability Web Service endorsed as we discussed in Chapter 2. The agent in host can expose itself as a service provider on the network, providing the service. It publishes its service to a registry where other parties can discovery and therefore invoke it. In order to accomplish its tasks, with the help of its own intelligence and directory service like UDDI, the mobile agent can deal with unanticipated requests and it can spontaneously recruit the help of third parties when they need to. It can act as Web Service client, requests and consumes the service provided by other providers on the network.

With a Web Service enabled agent platform, the mobility of the mobile agent may have implications in Web Services applications, such as dynamic deployment of web services, remote monitor service.

4.4 Framework

Incorporating Web Services into an agent system won't change the fundamental architecture of agent system. But it does change some of agent system landscape. In this section, we are going to propose and discuss a framework of a Web Services enable mobile agent system. The presented framework is trying to encapsulate the emerging distributed computing technology-Web Service into mobile agent system, but as the scope of a master thesis, we are not able to cover the detail of every aspect of changes Web Services may bring into mobile agent system.

Like any other mobile agent system, the framework should provide all basic common modules and functionalities we discussed in chapter 3. To be a Web Services enabled mobile agent system, it should not only provide the Web

Services infrastructure or subsystem that enables the Web Service, but also, probably most important, the modules or subsystem that bridge the agent system with the Web Services infrastructure (subsystem).

These enabling modules in the framework provide the infrastructure that enables the incoming mobile agent (or agent) exposed as a web service. The web service provided by mobile agent (or agent) has no difference with any others. It can be accessed and invoked by service requestor (client). The agent service can be published to a UDDI compliant registry, either a private one or public registry to allow requestors to search and query. In the other hand, the agent system or agents should be able to act as service requestor (client), to search, query, and invoke other web services on Internet via SOAP.

Modern mobile agent systems are usually built following today's software development practice that advocates modular, component-based, object-oriented methodology. Systems built in this way usually have flexibility to expand to incorporate new elements or subsystems. The implementation of various Web Services systems is a good example of that. Most of the available Web Services packages from different vendors can be easily integrated into existing applications or systems. For instance, SOAP engine developed by Apache Software Foundation is implemented in Java following the best practice of object-oriented and modularized software development. It can be easily integrated into applications by instantiating a server or client engine class or deploying it as a servlet in any web server that is compliant with Servlet standardization. So, it is reasonable to expect that we may integrate some Web Services components or

modules to existing agent systems as its subsystems through certain mechanisms.

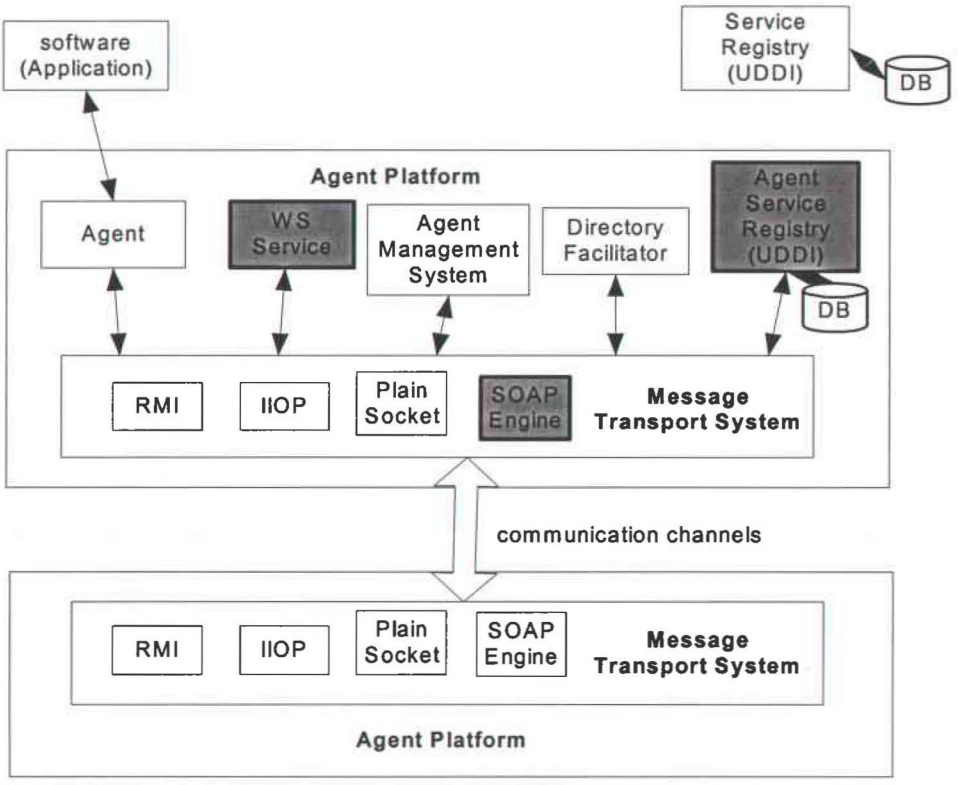

Figure 13. FIPA Reference Model with Web Services Components

To illustrate our framework, we still use FIPA reference model as the base. Besides the basic modules that constitute the AP, the Web Services related modules and components have been added (see Figure 13) and highlighted: SOAP Engine, Agent Service Registry, WS service. In the following section, we will discuss each of these new modules.

Agent Service Registry. It is a UDDI complaint private or public registry. It serves as Web Service registry as any Web Service registries on the Internet, providing a mechanism to allow service be published, discovered and invoked. When the agent is deployed as a web service, the service information will be

registered to Agent Service Registry. In the agent context (see Figure 14.), this registry should also server as the information repository of the agent that provides the service and the agent systems that host the agents.

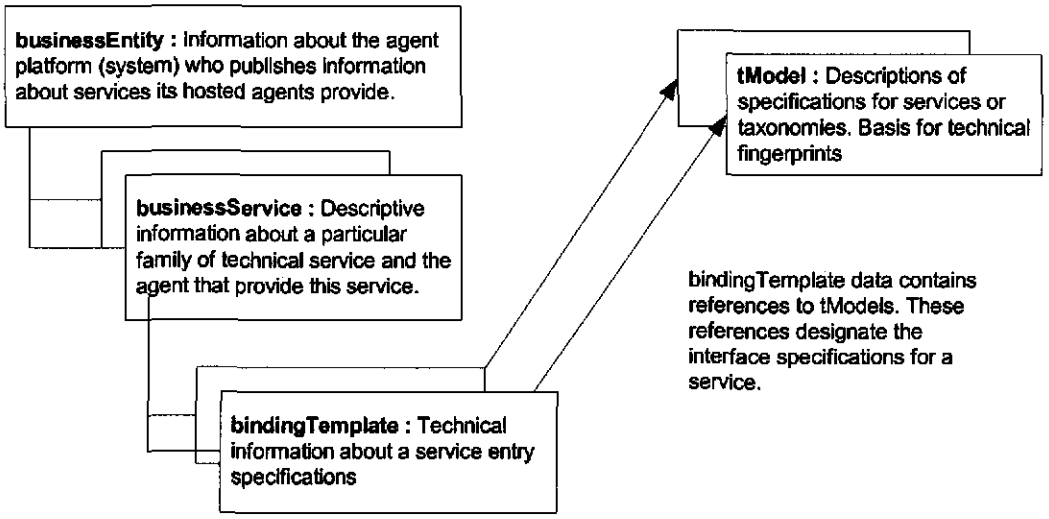

Figure 14. UDDI Data Structures in Agent Context

UDDI provides many levels and dimensions of flexibility such as arbitrary categorization schemes, open-ended discoveryURLs, etc. Many requirements can be satisfied by some application of the specification without introducing any extension. The mapping of agent system and agent attributes to businessEntity, businessService data structures is relatively straightforward. The businessEntity's fields authorizedName, operator, name, description, contacts could be used to capture agent system information. In case of capturing more comprehensive agent system information, fields discoveryURLs, indentifierBag, categoryBag could be used. The discoveryURLs field is a list of Uniform Resource Locator (URL) that point to alternate, file based service discovery mechanisms. The file can be XML format descriptive agent or agent system information compliant to standard. The indentifierBag and categoryBag are list of

name-value pairs that are used to record identification and tag with specific taxonomy information respectively [3]. These fields could be used to hold identification and category information.

Besides these pre-defined UDDI data structures to contain and organize service information, UDDI also supports extension through a derivation mechanism provided by XML Schema to enable access to additional functionality using extended UDDI API and data structures. XML Schema restrictions are explicitly prohibited to prevent the UDDI core functionalities from being arbitrarily limited [23]. So, conceptually and theoretically, UDDI should be perfect platform to serve as agent service registry.

For interoperability, format and content of the agent and agent platform (system) information should comply with agent standards such as FIPA or MAFIF.

Web Service (SOAP) Engine. It is the module that provides infrastructure to parse, process and consume SOAP message. The engine here is a broad term and it's dependent on the implementation of your choice. It should include at least two basic components. One is the client runtime. The other is server runtime. It also includes some utility tools.

The server runtime is to pre- and post-process messages that coming from the service requestor (client) to a web service and invoke that service and any appropriate utility if needed. Another core role of it, it is for deploying and exposing services. Here in agent context, the deployed service is actually the instantiation of an agent, but the SOAP engine should know nothing about it. It

should treat the agent service no difference with any other non-agent web service deployed.

The client runtime is to equip agents or other subsystems to communicate with UDDI registry, providing them with support for the Save, Delete, Find and Get among other operations specified in UDDI specification and access to UDDI registry via SOAP protocol. The client runtime also provides API and tools to allow agents access Web Services via SOAP.

Deployment Services. This module consists of all kind of Web Services related services and functionalities for the framework. Some of its services and functionalities are the cornerstones of the framework. For instance, there is an indispensable service to bridge the subsystem that supporting mobility with SOAP engine subsystem, so that the incoming mobile agent could be deployed as a web service. The services may vary depending on the implementation. The module may include the component to support additional Web Service security, the component to deploy the coming agent as a Web Service, the module to register the coming agent service to an UDDI registry, and more.

The incoming mobile agent may come with service information, in the form of a WSDL file (or generated at runtime) that used to describe the capability it has, or the services it provides to others. When mobile agent arrives at mobile agent system (host), based on the agent information, the agent host decides whether or not to deploy the agent to Web Service engine as a Web Service. Two processes involve in this Web Service deployment. One deploys the agent service to the Web Service engine, at meantime, the other publishes the service information to

a local or central Agent Service Registry (UDDI registry) if the system knows one. When mobile agent departures the agent host or be terminated, the service will be undeployed from the Web Services engine. The service registration entry in UDDI will also be removed.

As adding these new modules to an existing mobile agent system, the complexity of integration is really dependent on the architecture of mobile agent system and the implementation of Web Services infrastructure. The more modular, and loose coupling of both systems is, the easier to pull pieces together and make up such a system.

Chapter 5 . Implementation

In the previous chapters, we discussed the benefit and advantage of incorporating Web Services technologies into agent systems. A framework of such system has been presented. In this chapter, we will present a concrete architecture and give the detail of our implementation.

To build a complete Web Services enabled mobile system from ground is way beyond the scope of a Master thesis. What we implemented here is a prove-of-concept type of system, which can demonstrate that such system is in the reach of our hand, and most important, to demonstrate the benefit of mobile Web Services and its implication. Our approach to build such system is through the development and integration of some core subsystems, components. The implementation is to build the Web Services infrastructure for the framework, so that it is capable of deploying the incoming mobile agent as a web service. It is still a challenging work, since this is a new research area and lack of knowledge and materials on this topic.

Starting from the following section, we will discuss our approach to implement such system, the choice we made in choosing building blocks and the detail implementation.

5.1 Architectural Consideration

With coming up our concrete architecture, we keep several objectives and issues in our minds:

System agility. As both mobile agent system and Web Services technologies are still evolving, we hope the architecture of our system design is agile to response to changes.

Component-based and loose coupling. Component-based approach allows us to reuse components and quickly assemble them into a new system, easily to pull some pieces to build a new component. Loosely coupling allows the component to be dismantled or replaced without breaking the system. It allows the system be very flexible to adapt to changes.

Internet based. Web Services is all about application-to-application or program-to-program service call via Internet protocol (SOAP over HTTP). By this nature, most of the implementations of Web Services are deployed in Web application servers even though it's not necessary. As the ubiquity of Internet presence in distributed computing world, it is desirable to integrate Mobile Agent System into Web Server.

Availability of source code. As choosing the building block of subsystems, we particularly care about the availability of the source code. Our approach is to integrate and extend some existing components and subsystem, so the source code would help us understand the in-and-out and it is extremely helpful to the integration work.

5.2 Architecture

With these concerns in our minds and through a lot of research and investigation, we designed and implemented a prototype. The architecture of the prototype is depicted in Figure 15.

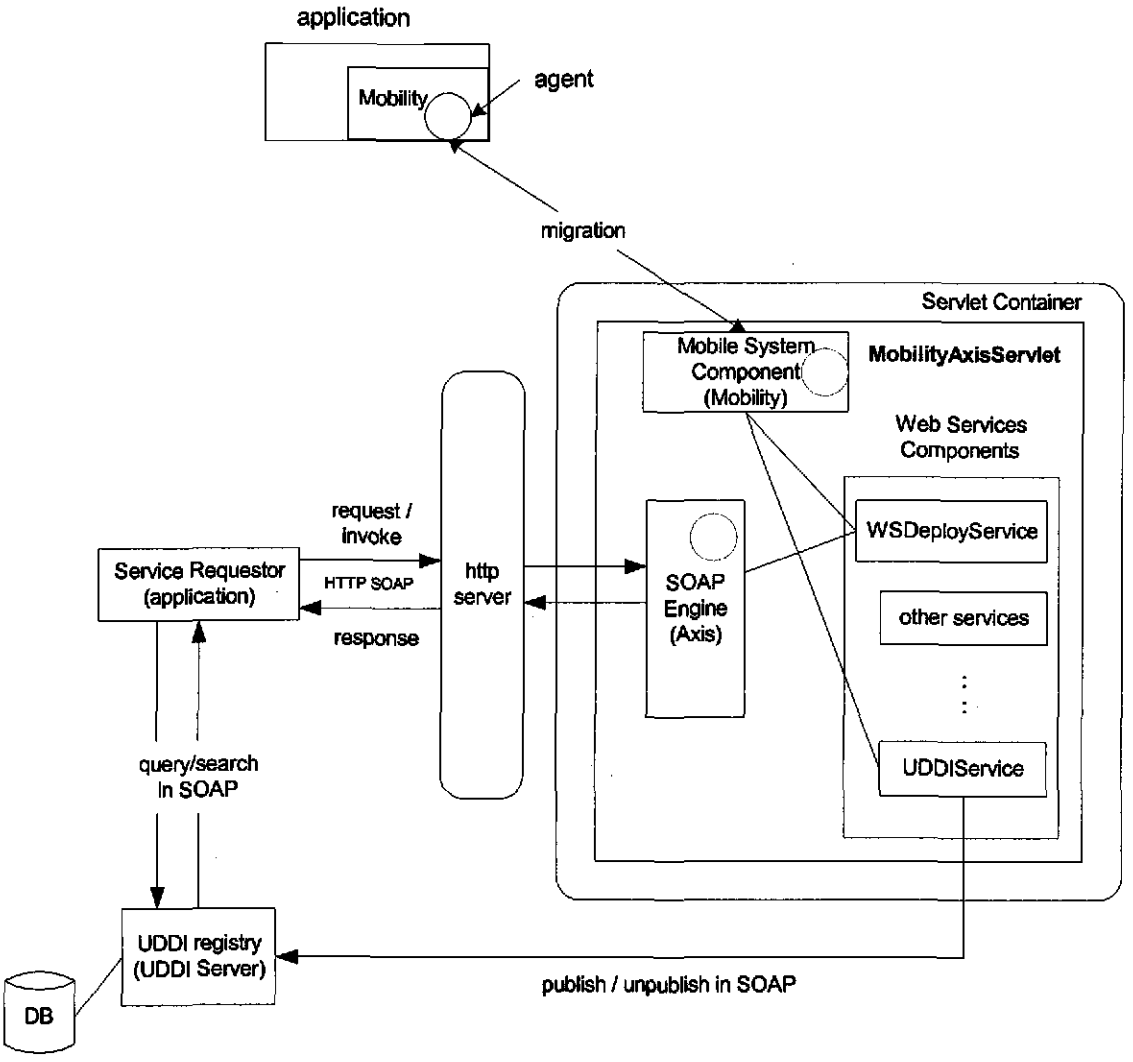

Figure 15. Overview of the Implementation

The whole system is wrapped inside a servlet called MobilityAxisServlet. It contains three major subsystems (components): SOAP engine, a mobility Component and Web Services Component.

The mobility component called Mobility is actually a lightweight of mobile agent system with many features you would expect in a regular mobile agent system. It is a result of research work done at Computer Engineering Department, University of Coimbra. With this component inside of it, MobilityAxisServlet

becomes a capable mobile agent system. It is able to receive and send mobile agents besides many other features [4].

The SOAP engine enables MobilityAxisServlet to process and consume the SOAP messages (either as client or server), expose and invoke the appropriate web service. The engine we used here is called AXIS, an open source implementation from Apache Software Foundation.

Web Services component enables the incoming agent to be turned into a web service as it is being deployed to the SOAP engine. When the deployment is successful, the service information is being published to the Agent Service Registry, a public or private UDDI registry where the service requestor (client) can query and request the service. Web Services component is the soul of the system. The integration and development made it possible that the mobile agent to be deployed and registered as a web service,.

The Agent Service Register, a UDDI compliant Web Services registry is out of the MobilityAxisServlet. It is an UDDI 2.0 implementation from HP based on Java technology.

It comes no surprise that our Web Service enabled mobile agent system is built on a platform of Servlet compliant Web server. Most of industry vendors like BEA, IBM, SUN, implement their Web Service infrastructure by extending the web server to support emerging Web Services technologies, SOAP, WSDL etc. The extension is usually accomplished through using Servlet technology. The servlet technology provides a simple mechanism for extending and enhancing the functionalities of a web server. There are many web servers or stand-alone

servlet engines support Servlet specification [16]. We choose Tomcat 4.0, a standard compliant Servlet engine from Apache Software Foundation as our platform. We chose it because it is an open-source and has close tie with AXIS, an open-source HTTP SOAP implementation from the same publisher.

5.3 Implementation

For the implementation, we select varieties of software packages to construct the prototype. It involves a lot of integration and development of some necessary components. In this section, we go over the software packages, components we had use and the integration and development works.

5.3.1 Programming Language

Every single element of the implementation in our system is based on Java technology. Java technology gets matured with the growth of Internet. It becomes the de facto language in distributed computing world.

Java was designed to be a portable, easy to learn, network aware object-oriented language. Instead of compiling Java into native instruction codes, it is compiled into an intermediary format known as bytecodes. The bytecodes can then be interpreted on any platform that has a suitable java interpreter that is known as Java Virtual Machine (JVM). So, by nature Java supports weakly mobile code. The inherent platform independence supported through interpretation plus its security model and object serialization has made Java an extremely popular choice among mobile agent framework implementation. Almost all mobile agent systems developed in recent years chose Java technology as the development platform and language. The Mobility Component is not an exception.

Because of its platform independent, Java dominates as a leading choice of implementation language in distributed computing. All the subsystems we used in our system are developed on Java technology platform. That list includes Agent Service Registry (an UDDI registry), SOAP engine (AXIS), mobile agent system (Mobility), Servlet engine and web server. Our development and integration is no doubt to be implemented in Java.

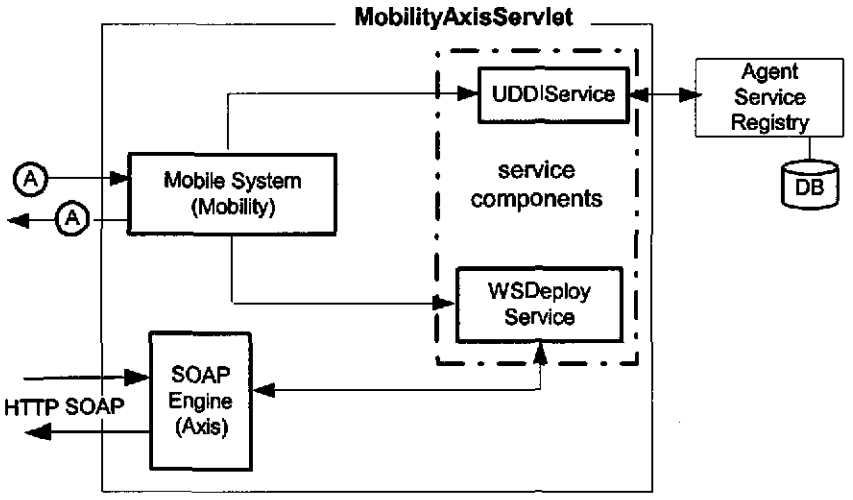

Figure 16. MobilityAxisServlet

5.3.2 MobilityAxisServlet

The MobilityAxisServlet we developed acts as the container of the system (see Figure 16). It contains all subsystems except the Agent Service Registry. It listens a port for the incoming HTTP SOPA request, and then passes it to SOAP engine. It also relays the HTTP SOAP response back to the requestor when SOAP engine returns response message. The Mobility component listens a different port for incoming mobile agent. The mechanism of connecting these three subsystems is event model.

As the container, MobilityAxisServlet's main role is to initiate the system when it starts. MobilityAxisServlet starts the SOAP engine (Axis), initiates the Mobility and service components. It then registers the service components to Mobility, and vice versa. The service components either implement Agent Lifecycle Events or Service Support Events or both. When the mobile agent arrives at or departs from Mobility, an event is trigged and the agent information is passed to the registered listeners. As the Mobility's listeners, Service components take corresponding action upon receiving the events. WSDeployService is the module to deploy the incoming agent as a web service. UDDIService is module to publish the new deployed web service to the Agent Service Registry.

Be a mobile agent system, the host of mobile agents, MobilityAxisServlet registers itself to the Agent Service Registry (UDDI registry) as a businessEntity. The web services provided by the mobile agents it hosts are the businessService under this businessEntity.

5.3.3 *Component-based Mobility*

While there are a number of mobile agent systems available, but when we come to build our prove-of-concept system, the choices are limited because of the concerns mentioned in section 5.1. We chose Mobility Component as our mobile agent system. The decision is based on following factors:

1. It is developed in Java programming language, and with courtesy of Paulo Marques from University of Coimbra, I was able to get the source code to get better understanding of its implementation.

2. It is component-based framework with very small footprint. It is easy to be used as a building block in application (Servlet).
3. It provides clear agent abstractions, agent lifecycle facilities, and mechanisms for extensibility.
4. It contains the notion of agent itinerary on which the agent migration is based.

We have briefly introduced the innovative project on mobile agent system at University of Coimbra in Chapter 3. Contrary to most existing mobile agent platforms, the researchers there present a component-based **Application Centric Mobile Agent System** (ACMAS) that enables ordinary applications to use mobile agents in an easy and flexible way. The system (framework) was implemented using the JavaBeans component framework, and is centered on the so-called Mobility Component.

The component Mobility provides the basic support for agent migration, management, and an extensibility mechanism that allows other components to connect to it. Mobility is very modular and small footprint. A very flexible plug-in architecture was devised in order to be able to develop any number of services and to seamlessly integrate them in the existing infrastructure. Applications can be developed using current object-oriented approaches and become able of sending and receiving agents by incorporating the mobility components. The system extensibility is based on an event model - a general architecture based on binary software components [5]. The extensibility is crucial to our implementation as it's the glue that transforms the mobile agents to Web

Services. To better serve to present and explain our implementation, we here brief the Mobility framework. For the detail about the framework, see [4][5].

The Mobility Component is coupled with the application and with the mobility supporting services through events (Figure 17). Events allow the component to be notified of changes on its surrounding environment and to notify the environment of changes in its execution state.

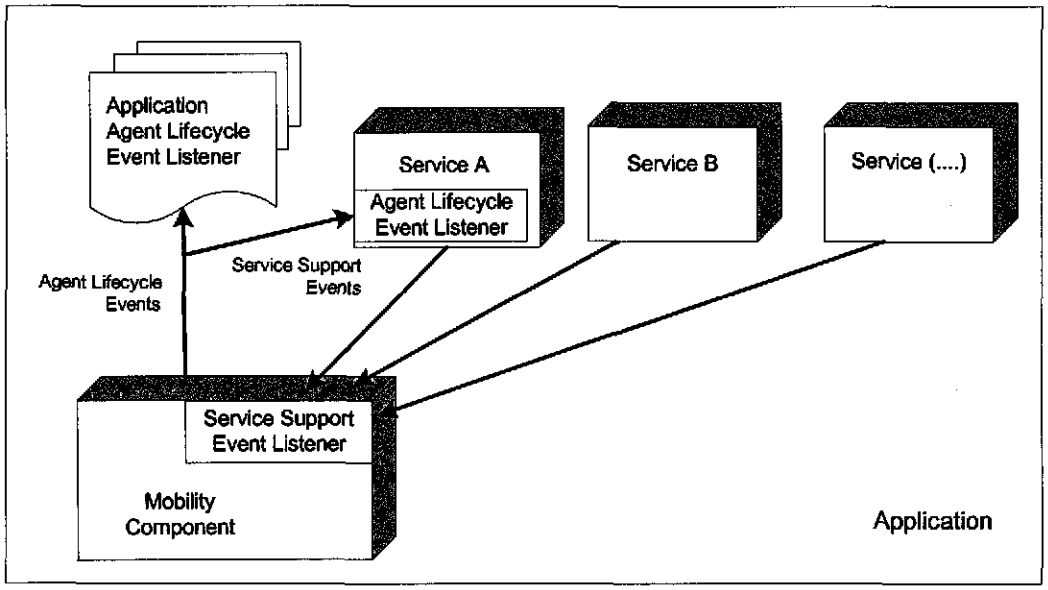

Figure 17. Interactions Between the Mobility Component and Application

There are two major sets of events: Agent Lifecycle Events and Service Support Events. Agent Lifecycle Events represent changes in the state of a running agent (as examples, when an agent arrives or departs from the application) in Mobility Component. Service Support Events enable the Mobility Component to incorporate new services at runtime and to be notified if they are no longer available.

Whenever an agent changes it execution state in Mobility Component, an Agent Lifecycle Event is fired. The application and the mobility services can register

their listeners with the Mobility Component, to be notified when one of those events occurs. When an event takes place, the listeners are able to examine the agent which the event be fired on and if necessary, call methods on the agent or other modules of the application. This mechanism is especially important to the high level of services since they can make use of those events to accomplish their functions.

Service Support Events exist to allow a service to be incorporated or removed from the system at runtime, and to allow services to be configurable. A service registers itself with Mobility Component as a source for those events and, whenever it becomes available, it fires an event that notifies the Mobility Component of the occurrence. Also if the service wishes to be removed from the list of available services, it fires an event requesting to be removed off.

The Mobility Component is extremely small and supports only the core functionalities needed to migrate and manage agents efficiently. Figure 18 shows the major sub-modules of the component.

The Agent Sender and Agent Receiver modules are responsible for sending and receiving agents. These modules take care of the serialization and deserialization of the agents, and make them available for other modules. The Mobility supports weak migration. The mobile agent (actually the code and state of agent) migrates from application to application based on its itinerary, or on the command of application or other agents.

The Event manager module is responsible for sending the Agent Lifecycle Events to the registered listeners, whenever an agent changes its state.

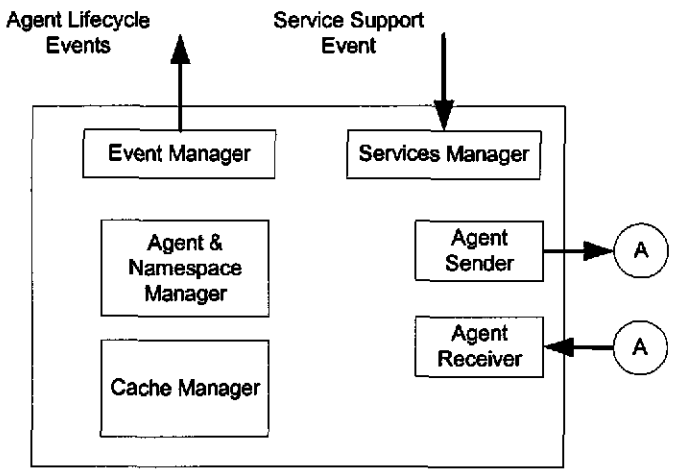

Figure 18. The Mobility Component.

The Services Manger is responsible for listening to the incoming Service Support Event, making higher-level service available or unavailable. Implementing high-level services implies implementing a ServiceFactory capable of creating service object instances on demand. Each of those instances implements a well-defined ServiceInterface that is made available to agents requesting the service. Associated with each service there is also a ServiceDescriptor that identifies the service. An agent can request a service by name or by ServiceDescriptor. It is also possible to ask which services are registered with the Mobility Component.

Agent Lifecycle Event Listener and Service Support Events Listener provide the mechanism of extensibility of Component Mobility, they allow the Component communicate with MobileAxisServlet and other service components.

Particularly we need two service components. One is called WSDeployService, which deploys the incoming agent as a web service when the system (actually Mobility Component) receives one. The other is called UDDIService, which

publishes the web service just deployed to the private or public Agent Service Registry (UDDI registry).

5.3.4 SOAP engine AXIS

We use AXIS (beta 2)[17], an open-source SOAP engine from Apache Software Foundation as our Web Service engine. Axis is essentially a SOAP engine, a framework for constructing SOAP processors as clients, servers, or gateway. Besides it is SOAP 1.1 compliant, it also includes a lot other features and functions, see [17] for the detail. Here we brief some architecture and features that concern our implementation and integration.

In our MobileAxisServlet, we use the concrete class AxisServer as our SOAP engine. The Figure19 depicts how SOAP message be processed at the server engine [18].

A service request, a SOAP message arrives (in some protocol-specific manner) at a Transport Listener. It's the Listener's job to package the protocol-specific (in our case, javax.servlet.http.HttpServletRequest) data into a Message object (org.apache.axis.Message), and put the Message object into a MessageContext. MessageContext is a structure that contains several important parts: a "request" message, a "response" message, and a bag of properties. The MessageContext is then loaded with various properties by the Listener. In our implementation, the request is HTTP SOAP (SOAP message embedded in HTTP). The Transport Listener is our MobilityAxisServlet, which listens the incoming message.

Once the MessageContext is ready to go, the Listener hands it to the AxisEngine. The AxisEngine's job is simply to pass the resulting MessageContext through a configurable set of Handlers, each of which has an

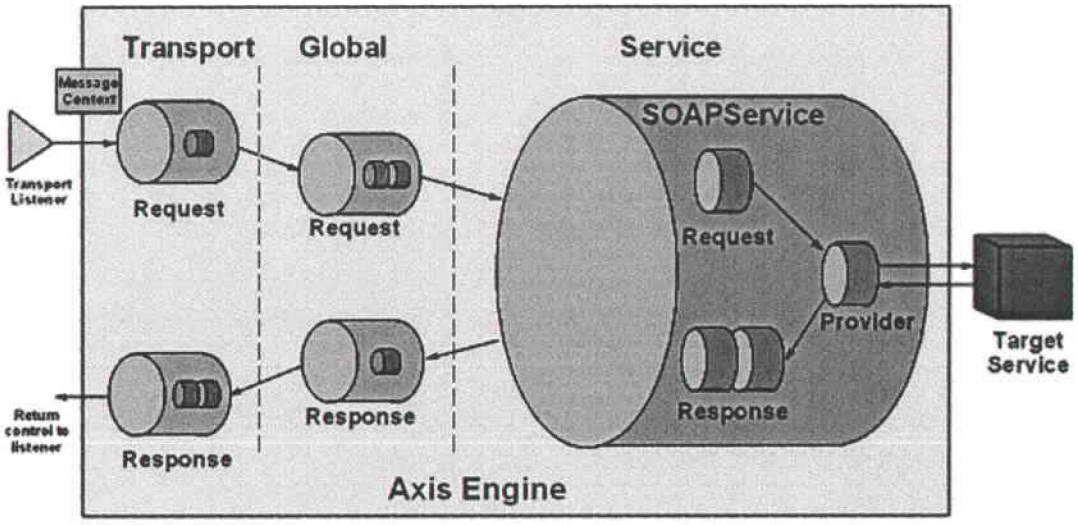

Figure 19. Axis Engine Architecture

opportunity to do whatever it is designed to do with the MessageContext. Finally the MessageContext reaches the Provider, which is a Handler responsible for implementing the actual back end logic of the service. In our case, the provider is the aorg.apache.axis.providers.java.RPCProvider class. When invoked, it attempts to call a backend Java object (shown as Target Service in the Figure 19) whose class is determined by the "className" parameter specific at deployment time. Then the result returned from Target Service is packed into MessageContext's "response" part to travel along the reverse path. When the response reaches the Listener, it's a well formatted SOAP message again, ready to send back to service requestor.

To make the backend Java class available to Axis engine as a target service, it must be deployed first. This is usually done by using Axis Web Service Deployment Descriptor (WSDD) tool. A deployment descriptor contains a few things you want to deploy. We list one below as an example. It's the wsdd file (or called descriptor) we use for in our evaluation case in later section.

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="StockQuoteService" provider="java:RPC">
    <parameter name="className" value="samples.stock.StockQuoteService"/>
    <parameter name="allowedMethods" value="*"/>
    <parameter name="scope" value="application"/>
  </service>
</deployment>
```

Figure 20. StockQuoteService Deployment Descriptor

The outermost element tells the engine that this is a WSDD deployment, and defines the "java" namespace. The service to be deployed is called StockQuoteService, which is a name given. The service provider is java:RPC, an Axis built-in Java RPC service. The actual class which handles this is org.apache.axis.providers.java.RPCProvider. The parameter className indicates the Java class to be instantiated. The allowedMethods tells the engine what class's public methods may be called via SOAP. "*" means any public method of that class could be called. Or list the method names with comma as delimiter if has more than one method. Parameter scope defines the scope of the service object. It could any value of "request", "session", or "application". The "application" scope will create a singleton shared object to serve all requestors.

Once have this file, send it to Axis server in order to actually deploy the described service. In Axis, AdminClient is one way to do that.

For undeployment, use undeployment descriptor (see Figure 21).

```
<undeployment xmlns="http://xml.apache.org/axis/wsdd/">
 <service name="StockQuoteService"/>
</undeployment>
```

Figure 21. StcokQuoteService Undpeloyment Descriptor

In the context of our implementation, we actually deploy the mobile agent to the AXIS engine because here the mobile agent is in fact a Java object, an instance of Java class. It is deployed as application scope since it serves all service requests. In implementation, we developed a service component WSDeployService for this deployment/undeployment process. Upon receiving a mobile agent, the Mobility Component fires an event to notify all its listeners. As one of the registered listeners, the service component deploys the mobile agent as a Web Service (one of Target Services in Figure 19). Reversibly, the service component undeploys the Web Service when the Mobility Component fires another event when the mobile agent migrates to other host.

5.3.5 Web Service Enabled Mobile Agent

When mobile agent migrates from place to place, it usually packages some information about itself, such as itinerary, execution state. Be a Web Service enabled agent, it has its own characteristics. In the context of Web Services, the agent should include information about deployment/undeployment, such as which methods allowed to be called, how it should be deployed/undeployed upon

arrival/departure, information about its service, such as TModelKey, ServiceKey along with others. The information can either be saved to an external file and be accessible through the URL link or carried with the agent as properties.

```
package webservice.agent;

import mm.mob.agent.Agent;
/**
 * Base interface for implement a web service enabled agent.
 */

public interface WebServiceAgentInterface
{
   public void migrate();
   public void migrate(String destURL);
   public void setWDSLInterfaceURL(String url);
   public String getWDSLInterfaceURL();
   public void setWSDLImplementationURL(String url);
   public String getWSDLImplementationURL();
   public void setWSDLURL(String url);
   public String getWSDLURL();
   public boolean IsAuthenticed();
   public void setDeployscript(String str);
   public String getDeployscript();
   public String getUndeployscript();
   public void setUndeployscript(String str);
   public void setAuthenticed(boolean bool);
   public boolean IsWebServiceEnabled();
   public void setWebServiceEnabled(boolean bool);
   public String getTModelKey();
   public void setTModelKey(String str);
   public String getServiceKey();
   public void setServiceKey(String str);
}
```

Figure 22. Web Services Enabled Agent Interface

This web service information is indispensable. To enforce that, we define a Web Service agent interface class called WebServiceAgentInterface (see Figure 22). All web service enabled mobile agents should implement this interface.

The agent we defined in our system is called WebServiceAgent. It is a class that extends from Mobility's Agent class and implements the WebServiceAgentInterface. All mobile agents should extend the WebServiceAgent if they intend to behave as web service enabled agent. Still it can change its property to disable its web service parts.

Since the WebServiceAgent inherits from Mobility's Agent class, it holds all characteristics of Agent class. Agent has a lifecycle that goes through several stages: creation, running, jumping, and death. Every state transition represents an event in the life of the agent. The agent, the application and other party may be informed of theses events, so that they can take appropriate actions. For agent, this is done using callback. The Agent class has several methods defined that are called by the Mobility when a state transition happens.

public void onCreation() - Called when an agent is created.
public void onArrival() - Called when an agent arrives at a host.
public void onTermination() - Called when an agent terminates. If the agent is abruptly killed by the application, this method is not called.
public void onMigration(String URLDest) - Called when an agent migrates to another destination. The URLdest is the address where the agent is going.
public void onPlaceChanged(String newPlace) - Called when an agent change places.

Figure 23. Agent Callback Methods

5.3.6 Service Components

The Service Components are the core enabling components of our infrastructure. They are the glue to allow Mobility and Axis components work accordingly. We developed two main service components called WSDeployService and UDDIService and some auxiliary classes.

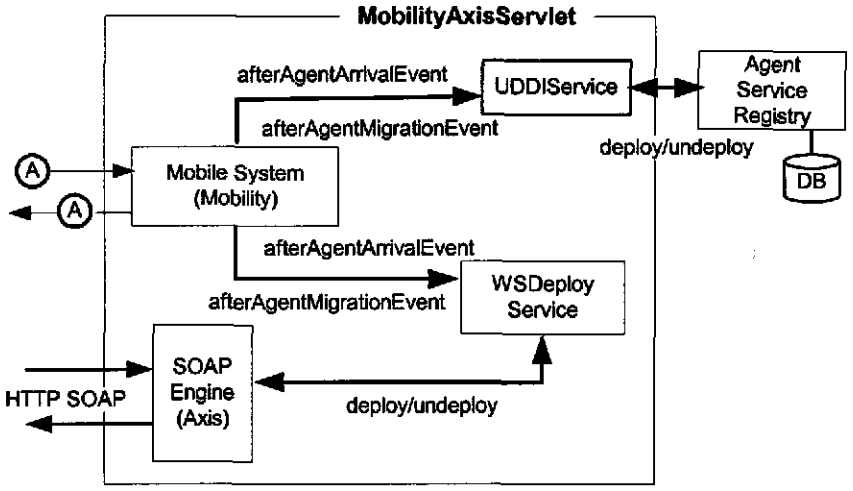

Figure 24. Agent Service Deployment and Publishing

An application or a service can listen for events generated by any agent running in the Mobility component after it is registered as a listener of Mobility. This is implemented following JavaBean event model. To catch agent events, the application or service must do two things: one is to implement the interface AgentLifecycleListenser, the other is to register itself with the mobility component. So whenever an agent running in Mobility changes its execution state, an Agent Lifecycle Event is fired and caught by that application or service. Then the application or the service is able to examine the agent responsible for the event and if necessary, calls methods on the agent or in other modules. Here is a list of the events that may be caught by the application or service.

onAgentArrival (AgentLifecycleEvent e)- *When the agent arrives at* the application.
afterAgentArrival(AgentLifecycleEvent e)- After the agent arrives at the application.
onAgentMigration(AgentLifecycleEvent e)- When the agent departs from the application.
afterAgentMigration(AgentLifecycleEvent e) - After the agent departs from the application
onAgentFailedMigration(AgentLifecycleEvent e) - When the agent fails to migrate
afterAgentCreation(AgentLifecycleEvent e)- When a new agent instance is created.
afterAgentClonning(AgentLifecycleEvent e)- When an agent is cloned.
afterAgentTermination(AgentLifecycleEvent e)- When an agent dies.

Figure 25. AgentLifcycleListenser Interface

WSDeployService. WSDeployService is a service that implements AgentLifecycleListener interface. It is registered with Mobility when the system (MobilityAxisServlet) starts. Particularly, the WSDployService concerns three events, afterAgentArrival, afterAgentTermination, and afterAgentMigration. It checks whether the incoming agent is asking for web service deployment when receiving afterAgentArrivalEvent and if it is, it deploys the incoming agent to Axis Server as a web service (a targeted service) according to the agent's deployment descriptor. Correspondently, with afterAgentMigration and afterAgentTermination events happening, WSDployService undeploys the agent from the Axis Server. The deploy/undeploy implementation basically uses the references of Mobility and Axis server and wraps method calls of some Axis server deployment APIs. See Apendix A, the detail code of the WSDeplyService class.

UDDIService. UDDIService provides the service to publish a web service to a public or private Agent Service Registry (UDDI registry) after a mobile agent is deployed to Axis server as a service. Also, it removes the service entry from UDDI registry when the agent leaves (migrate from) the system or it be terminated. UDDIService implements both Agent Lifecycle Listener and Service Support Event Listener. When MobilityAxisServlet starts, it registers UDDIService to the Mobility. When the agent arrives, the system checks that if the agent intends to deploy itself as a web service. If so, the system will deploy it and then publish the service to an Agent Service Registry (UDDI registry). The service that agent provides is described by a WSDL file or the information can be gotten by runtime. With the publishing information, UDDIService uses couple utility classes we developed to communicate to the Agent Service Registry. In the following section, we give the detail of web service publishing and its implementation.

5.3.7 Web Service Publish

WSDL service descriptions can be structured in multiple ways. If the reusable information is separated from the information that is specific to a given service instance, the use of WSDL and UDDI together becomes particularly simple. [24] gives a guideline of implementation and is referred by UDDI organization as the "best practice".

The **import** element in WSDL allows the separation of the service description into two parts, referred to as "service interface definition" and "service implementation definition". Typically, information common to a certain category of business services, such as message format, portType (abstract interfaces), and

protocol bindings, are included in the reusable portion, while information pertaining to a particular service endpoint is included in the service implementation definition portion. See Appendix C, D for examples of service interface definition and service implementation definition respectively.

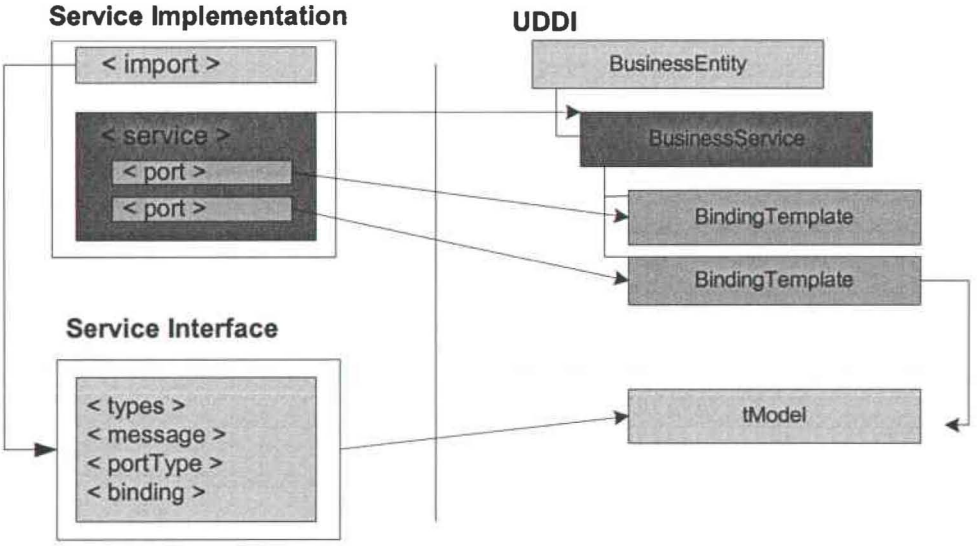

Figure 26. WSDL to UDDI Mapping

The Figure 26 represents the mapping of WSDL to UDDI, how WSDL supports the creation of UDDI businessService entities.

To realize the process of publishing Web Service, which really is to publish the elements of WSDL (description of the service) into UDDI registry, we develop several auxiliary utilities, PublishBusinessEntity, PublishServiceImplementation, PublishServiceInterface to help the process. The utilities are developed mainly using UDDI4j package, which is a Java API for programmers to use to interact with UDDI compliant registries. It's the result of joint development between HP, IBM and others. Service component, UDDIService uses these utilities to publish the service into UDDI registry.

5.3.8 Agent Service Registry – a UDDI Registry

Agent Service Registry is a UDDI registry which we use as our system directory service (see section 4.4 Framework). The UDDI registry implementation we chose for our system is HP's Web Services Registry™ 2.0 [25] that a UDDI version 2 compliant registry.

5.4 Evaluation

As any research project, the verification and evaluation is always a curial aspect. To our research, it is really to verify that our system works the way as it supposes to be. There is no comparison we can make with any system, as so far we believe our research in its area is the first in its kind.

To evaluate and verify our system, we develop a series of user cases to test our system. Here we introduce an interesting one. It is not only to demonstrate how our system works, it also brings the implication how the system we designed could be used to build applications in real world.

The demo application is to create a mobile agent that is able to provide a stock quote service. The agent travels to our system and be deployed as a web service. The mobile agent is called StockQuoteService, which extends our WebServiceAgent class. One of its methods getQuote is to use live stock quote service provided by XMLtoday.com. It requests a quote to that site through HTTP GET interface, and gets response in XML string. By processing the XML string, the method returns a quote in float. For the detail of StockQuoteService class, see Appendix E.

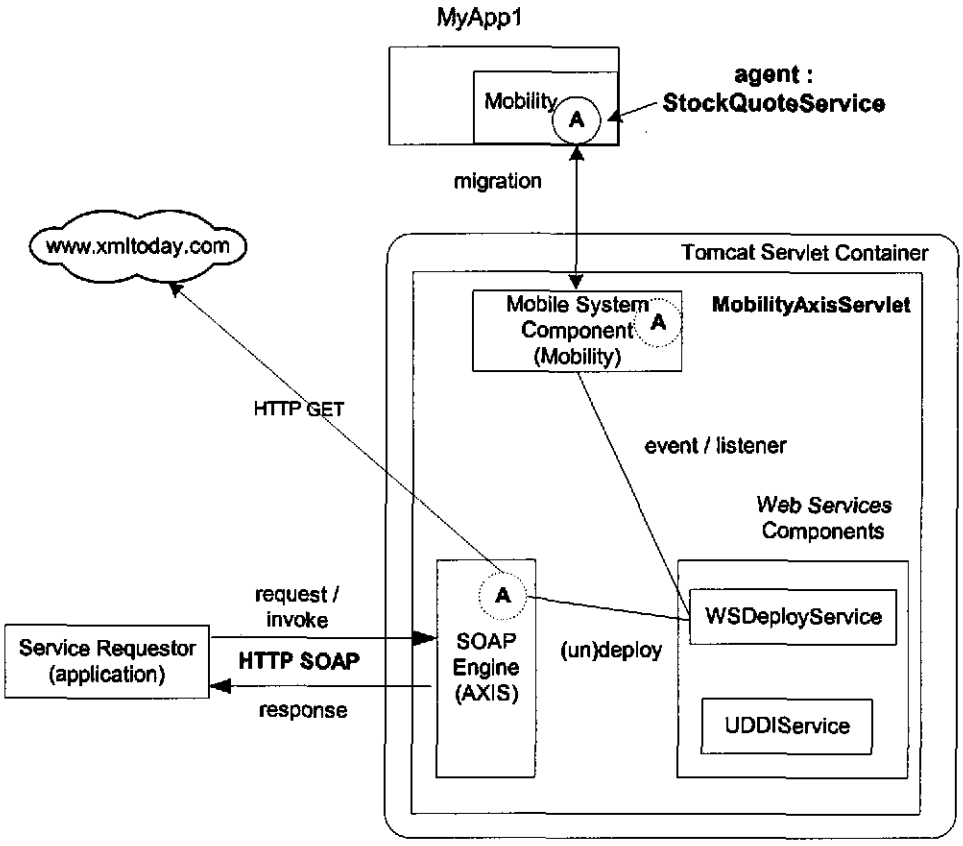

Figure 27. Demo Application

Figure 27 represents the case scenario. A mobile agent (an instance of StockQuoteService class) is created in an application. It then migrates to our system (MobilityAxisServlet) and is deployed as a web service upon its arrival. Another application (a service requestor), to make our demo simpler we assume it knows the existence of the stock quote service, makes a HTTP SOAP request to MobilityAxisServlet. MobilityAxisServlet passes the request to SOAP engine AXIS. Service is invoked and response be passed back to the service requestor. Upon receiving the response (stock quote), the service requestor sends the second request which is to call the migrate service. This call makes the mobile agent to migrate, leave the MobilityAxisServlet. No doubt, the web service the

agent provides is removed from the MobilityAxisServlet along with the gone of the agent. The third request from the service requestor, asking the stock quote again is really to test that the stock quote service in MobilityAxisServlet is gone, no existed any more.

Chapter 6 . Conclusion

6.1 Thesis Summary

Built on top of existing Internet protocols and based on open XML standard, Web Services technologies are emerging to provide a systematic and extensible framework for application-to-application interaction. It's platform and language independent and advocates the system flexibility, extensibility and interoperability. We Services technologies have been widely adopted and deployed in industrials and still keep evolving. These technologies may be used to help to architect and standardize Mobile Agent Systems, but have not been investigated. The purpose of this thesis is to research the possibility of combining the mobility and intelligence provided by mobile agents with the location and platform agnostic feature of Web Services, to research how to incorporate Web Service technologies into mobile agent systems.

6.2 Thesis Contributions

This thesis has three major contributions.

The first contribution is to make the argument that the emerging Web Services technologies would be good candidates to help architect and standardize the (mobile) agent systems. We discuss the benefit and advantage of this approach.

The second contribution is the proposal of a framework of Web Services Enabled Mobile Agent System. We give out the design and justification, and discuss the synergy of these two technologies would bring to us.

The third contribution is the concrete architecture and its implementation of such a system. A prove-of-concept demo application is developed for evaluation. The implementation uses a variety of newest software packages, components, tools from IBM, Apache Foundation, HP and others.

6.3 Future Directions

During this research work, a number of directions have been identified for future work. So far, our implemented system deploys/undeploys the incoming agents as web services, acts as server role that provides services for the clients. It would be desirable to develop the components and modules that allow the mobile agent as a client to query the service registry to look for wanted service, automating generation of SOAP request message to the intended service provider to invoke service. So that mobile agents or static agents would be able to communicate and consume web services when it needs to fulfill its tasks.

Other research direction might include: extend modules to allow deploying the incoming mobile agent as a web service in session or request scope, develop some more complicated applications to demonstrate the power of our Web Service enabled mobile agent system, research the areas that may mostly utilize our system.

Appendix A. SOAP Message

We list two SOAP messages below. They are the request and response SOAP messages generated in our demo case StockQuoteService web service. The service requestor (an application) sends "IBM" stock price SOAP request, MobilityAxisServlet sends back the SOAP response.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
  <ns1:getQuote xmlns:ns1="urn:xmethods-delayed-quotes"
            SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <symbol xsi:type="xsd:string">IBM</symbol>
  </ns1:getQuote>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 28. SOAP Request for Service

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
  <ns1:getQuoteResponse xmlns:ns1="urn:xmethods-delayed-quotes"
             SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <return xsi:type="xsd:float">133.625</return>
  </ns1:getQuoteResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 29. SOAP Response

Appendix B. WSDL File for StockQuoteService

The following list is a complete WSDL file describing a web service StockQuoteService.

This service is developed as a demo in our research project.

```
<?xml version="1.0" ?>
<definitions name="urn:GetQuote"
        targetNamespace="urn:xmltoday-delayed-quotes"
        xmlns:tns="urn:xmltoday-delayed-quotes"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">

 <!-- message decIns -->
 <message name="testRequest">
  <part name="string" type="xsd:string"/>
 </message>
 <message name="GetQuoteRequest">
  <part name="symbol" type="xsd:string"/>
 </message>
 <message name="GetQuoteResponse">
  <part name="return" type="xsd:float"/>
 </message>

 <!-- port type decIns -->
 <portType name="GetQuote">
  <operation name="getQuote" >
   <input  message="tns:GetQuoteRequest"/>
   <output message="tns:GetQuoteResponse"/>
  </operation>
  <operation name="test" >
   <output message="tns:GetQuoteResponse"/>
  </operation>
 </portType>
```

Figure 30. StockQuoteService WSDL, Part 1

```
<!-- binding declns -->
 <binding name="GetQuoteBinding" type="tns:GetQuote">
  <soap:binding style="rpc"
          transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getQuote">
   <soap:operation soapAction="getQuote"/>
   <input>
    <soap:body use="encoded"
          namespace="urn:xmltoday-delayed-quotes"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   </input>
   <output>
    <soap:body use="encoded"
          namespace="urn:xmltoday-delayed-quotes"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   </output>
  </operation>
  <operation name="test">
   <soap:operation soapAction="test"/>
   <input>
    <soap:body use="encoded"
          namespace="urn:xmltoday-delayed-quotes"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   </input>
   <output>
    <soap:body use="encoded"
          namespace="urn:xmltoday-delayed-quotes"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   </output>
  </operation>
 </binding>

 <!-- service decln -->
 <service name="GetQuoteService">
  <port name="GetQuote" binding="tns:GetQuoteBinding">
   <soap:address location="http://localhost:8080/axis/servlet/AxisServlet"/>
  </port>
 </service>

</definitions>
```

Figure 31. StockQuoteService WSDL, Part 2

Appendix C. Service Interface Definition

```
<?xml version="1.0"?>
<definitions name="StockQuoteService-interface"
 targetNamespace="http://www.getquote.com/StockQuoteService-interface"
 xmlns:tns="http://www.getquote.com/StockQuoteService-interface"
 xmlns:xsd=" http://www.w3.org/2001/XMLSchema "
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

 <documentation>
  Standard WSDL service interface definition for a stock quote service.
 </documentation>

 <!-- message declns -->
 <message name="GetQuoteRequest">
  <part name="symbol" type="xsd:string"/>
 </message>
 <message name="GetQuoteResponse">
  <part name="result" type="xsd:float"/>
 </message>

 <!-- port type declns -->
 <portType name="GetQuote">
  <operation name="getQuote" >
   <input  message="tns:GetQuoteRequest"/>
   <output message="tns:GetQuoteResponse"/>
  </operation>
 </portType>

 <!-- binding declns -->
 <binding name="GetQuoteBinding" type="tns:GetQuote">
  <soap:binding style="rpc"
          transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getQuote">
   <soap:operation soapAction="getQuote"/ >
   <input>
    <soap:body use="encoded"
          namespace="urn:xmltoday-delayed-quotes"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   </input>
   <output>
    <soap:body use="encoded"
          namespace="urn:xmltoday-delayed-quotes"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   </output>
  </operation>
 </binding>
</definitions>
```

Figure 32. StockQuoteServiceInterface

Appendix D. Service Implementation Definition

```
<?xml version="1.0"?>
<definitions name="StockQuoteService"
 targetNamespace="http://172.21.66.155:8080/wsdl/StockQuoteService"
 xmlns:interface="http://172.21.66.155:8080/wsdl/StockQuoteService-interface"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

 <documentation>
  This service provides an implementation of a standard stock quote service.
  The Web service uses the live stock quote service provided by XMLtoday.com.
  The XMLtoday.com stock quote service uses an HTTP GET interface to request
  a quote, and returns an XML string as a response.

  For additional information on how this service obtains stock quotes, go to
  the XMLtoday.com web site: http://www.xmltoday.com/examples/soap/stock.psp.
 </documentation>

 <import namespace="http://172.21.66.155:8080/wsdl/StockQuoteService-interface"
  location="http://172.21.66.155:8080/wsdl/StockQuoteServiceInterface.html"/>

 <service name="StockQuoteService">
  <documentation>Stock Quote Service</documentation>

  <port name="GetQuote" binding="GetQuoteBinding">
   <documentation>Single Symbol Stock Quote Service</documentation>
   <soap:address location="http://localhost:8080/axis/services/GetQuote"/>
  </port>
 </service>
</definitions>
```

Figure 33. StockQuoteService Implementation Definition

Appendix E. StockQuoteService Class

StockQuoteService class extends from WebServiceAgent, a Web Service enabled mobile agent. It's used in our demo application.

StockQuoteService.java

```
package samples.stock;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import java.net.URL;

//our import
import webservice.agent.*;
import mm.mob.agent.Agent;
import mm.mob.mobility.exceptions.UnableToMigrateException;

public class StockQuoteService extends WebServiceAgent{

   //private String WDSLInterfaceURL=null;
   //private String WSDLImplementationURL=null;
   private String WSDLURL=null;
   //private String deployscript=null;
   private boolean isAuthenticed=true;
   private boolean isWebServiceEnabled = true;
   private String TModelKey = null;
   private String serviceKey = null;

   /**
    * hard coded here, it could be replace by property file
    */

   private String WDSLInterfaceURL =
"http://localhost:8080/StockQuoteServiceInterface.html";

   private String WSDLImplementationURL = "http://localhost:8080/StockQuoteService-
Impl.html";
```

```
  private String deployscript = "<deployment
xmlns=\"http://xml.apache.org/axis/wsdd/\"" +
       " xmlns:java=\"http://xml.apache.org/axis/wsdd/providers/java\">\n" +
        " <service name=\"StockQuoteService\" provider=\"java:RPC\">\n" +
          " <parameter name=\"className\"
value=\"samples.stock.StockQuoteService\"/>\n" +
          " <parameter name=\"methodName\" value=\"*\"/>\n" +
          " <parameter name=\"scope\" value=\"Application\"/>\n" +
       " </service>\n" +
     "</deployment>";

  private String undeployscript = "<undeployment
xmlns=\"http://xml.apache.org/axis/wsdd/\">" +
                    " <service name=\"StockQuoteService\"/>" +
                    " </undeployment>";

  public StockQuoteService(){
    setWebServiceEnabled(isWebServiceEnabled);
    setWDSLInterfaceURL(WDSLInterfaceURL);
    setWSDLImplementationURL(WSDLImplementationURL);
    setAuthenticed(isAuthenticed);
    setDeployscript(deployscript);
    setUndeployscript(undeployscript);

  }

  private boolean firstTime = true;
  public void run(){
       try{
         Thread.sleep(60000);
         jump("mob://localhost:6000");
       }catch (Exception e){
         e.printStackTrace();
       } // jump to MobilityAxisServlet's Mob

  }

 public String test() {
  return( "Just a test" );
 }

 /**
  * Web Service
  */
```

```
public float getQuote (String symbol) throws Exception {
  // get a real (delayed by 20min) stockquote from
  // http://www.xmltoday.com/examples/stockquote/. The IP addr
  // below came from the host that the above form posts to ..

  if ( symbol.equals("XXX") ) return( (float) 55.25 );

  URL        url = new URL( "http://www.xmltoday.com/examples/" +
                    "stockquote/getxmlquote.vep?s="+symbol );

  DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
  DocumentBuilder        db  = dbf.newDocumentBuilder();

  Document doc  = db.parse( url.toExternalForm() );
  Element  elem = doc.getDocumentElement();
  NodeList list = elem.getElementsByTagName( "stock_quote" );

  if ( list != null && list.getLength() != 0 ) {
    elem = (Element) list.item(0);
    list = elem.getElementsByTagName( "price" );
    elem = (Element) list.item(0);
    String quoteStr = elem.getAttribute("value");
    try {
      return Float.valueOf(quoteStr).floatValue();
    } catch (NumberFormatException e1) {
      // maybe its an int?
      try {
        return Integer.valueOf(quoteStr).intValue() * 1.0F;
      } catch (NumberFormatException e2) {
        return -1.0F;
      }
    }
  }
  return( 0 );
}

public static void main(String[] args){

   StockQuoteService stock = new StockQuoteService();
  try
  {
   System.out.println(stock.getQuote(args[0]));
  }
  catch (Exception e)
```

```
    {

    }
     // Exit
    System.exit(0);

  }
}
```

References

[1]. Agent list: http://mole.informatik.uni-stuttgart.de/mal/preview/preview.html

[2]. Robert Gray, David Kotz, George Gybenko and Daniela Rus, "Mobile agents: Motivations and state-of-the-art systems", Handbook of Agent Technology, AAAI/MIT Press, 2000.

[3]. David Kotz and Robert Gray, "Mobile Agents and the Future of the Internet", Proceedings of the Workshop, Mobile Agents in the Context of Competition and Cooperation (MAC3) at Autonomous Agents '99, pages 6-12, Seattle, Washington, USA, May 1999.

[4]. P. Marques, P. Simões, L. Silva, F. Boavida, J. Gabriel, "Providing Applications with Mobile Agent Technology", in Proc. 4th IEEE International Conference on Open Architectures and Network Programming (OpenArch'01), IEEE Computer Press, Anchorage, Alaska, April 2001.

[5]. P. Marques, L. Silva, J. Silva, "Addressing the Question of Platform Extensibility in Mobile Agent Systems", International Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce (MAMA'2000), Wollongong, Australia, December 2000.

[6]. Dejan Milojicic, "Mobile Agent Application", 1999 July-September IEEE Concurrency.

[7]. On the Structuring of Distributed Systems: The Argument for Mobility, by Todd Papaioannou, PhD dissertation, February 2000.

[8]. FIPA ACTS Project, http://www.labs.bt.com/profsoc/facts/

[9]. FIPA Agent Management Specification, October 2001. FIPA.

[10]. FIPA specification, http://www.fipa.org/

[11]. FIPA Agent Management Support for Mobility Specification, May, 2002. http://fipa.org/specs/fipa00087/DC00087C.html

[12]. Gavalas D., Greenwood D., Ghanbari M., O'Mahony M., "Advanced Network Monitoring Applications Based on Mobile/Intelligent Agent Technology", Computer Communications Journal, special issue on "Mobile Agents for Telecommunication Applications", Vol. 23, No 8, pp. 720-730, April 2000.

[13]. Gavalas D., Greenwood D., Ghanbari M., O'Mahony M., "An Infrastructure for Distributed and Dynamic Network Management based on Mobile Agent Technology",

Proceedings of the IEEE International Conference on Communications (ICC'99), Vol. 2, pp. 1362-1366, Vancouver, Canada, 6-10 June 1999.

[14]. Agentcities Web, http://www.agentcities.org/

[15]. Web Services Conceptual Architecture (WSCA 1.0). IBM. May 2001.

[16] Sun Microsystems Inc. "Java Servlet Specification 2.3", http://java.sun.com/products/servlet/download.html

[17] AXIS project, http://xml.apache.org/axis/

[18] AXIS Architecture Guide, http://xm.apache.org/axis/

[19] Gunjan Samtani and Dimple Sadhwan, EAI and Web Services, http://www.webservicesarchitect.com/content/articles/samtani01.asp

[20] Orchestra Networks Inc., White Paper: Web Service, Business Ojects and Component Models, July 2000.

[21] Web Services Description Language (WSDL) 1.1, W3C. http://www.w3.org/TR/wsdl

[22] Accenture, Ariba, Inc., Commerce One, Inc., Fujitsu Limited, Hewlett-Packard Company, i2 Technologies, Inc., Intel Corporation, International Business Machines Corporation, Microsoft Corporation, Oracle Corporation, SAP AG, Sun Microsystems, Inc., and VeriSign, Inc., UDDI Version 3.0 Specification. http://www.uddi.org/specification.html

[23] Ariba, IBM, Microsoft, UDDI Data Structures Reference, September 2000.

[24] Accenture, Ariba,Inc, Intel Corporation, "Using WSDL in a UDDI Registry, Version 1.07", *UDDI Best Practice*, May 2002.

[25] HP's Web Services Registry™ 2.0, http://www.hpmiddleware.com/products/hp_web_services/registry/

[26] L. Silva, P. Simoes, G. Soares, P. Martins, V. Batista, C. Renato, L. Almeida, N. Stohr, "JAMES: A Platform of Mobile Agent for the Management of Telecommunication Networks", in Preceedings of LATA'99, Stockholm, 1999.

[27] D. Gavalas, D. Greenwood, M. Ghanbari and M. O'Mahony. "Advanced network monitoring applications based on mobile/intelligent agent technology", Computer Communications, Volume 23, Issue 8, April 2000.

[28] Andrzej Bieszcazad, Bernard Pagurek, Tony White, "Mobile Agents For Network Management", Fourth Quarter 1998, Vol.1 No.1, IEEE Communications Surveys.

[29] White, J. E, "Telescript technology: the foundation for the electronic marketplace", White Paper, General Magic, Inc., 1994

[30] James Hart, "How Web Services Came to Be", http://www.webservicesarchitect.com/content/articles/hart01.asp

[31] Schreiber, Richard. "Middleware Demystified." Datamation 41, 6 (April 1, 1995): 41-45.

[32] Agent projects, http://www.agentlink.org/resources/agentprojects-db.html

[33] Grasshopper, http://www.grasshopper.de/

[34] CORBA, http://www.corba.org/

[35] OMG Agent Platform Special Interest Group, http://www.objs.com/agent/index.html

[36] Agent Technology Green Paper, Agent PSIG, September 2000, http://www.objs.com/agent/index.html

[37] Mobile Agent Facility Specification V1.0, January 2000, http://www.omg.org/cgi-bin/doc?orbos/97-10-05