uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Zhiyong Weng

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.Sc. (Computer Science)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

A Framework for Mobile E-Commerce Based on Intelligent Agents

TITRE DE LA THÈSE / TITLE OF THESIS

Thomas Tran

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Anil Somayaji

Jochen Lang

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

A Framework for Mobile E-Commerce Based on Intelligent Agents

Zhiyong Weng

Thesis submitted to the
Faculty of Graduate and Post Doctoral Studies
In partial fulfillment of the requirements
For the Masters in Computer Science Degree*

School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa

---

* The Masters program in Computer Science is a joint program with Carleton University, administered by the Ottawa-Carleton Institute for Computer Science

Your file Votre référence
ISBN: 978-0-494-49291-8
Our file Notre référence
ISBN: 978-0-494-49291-8

Abstract

Mobile e-commerce has offered new channels for trading products in e-marketplaces over mobile devices by applying the intelligent and mobile agent technologies. These existing e-commerce applications have two specific limiting factors. First, it is difficult for them to be integrated together due to the lack of common communication languages and interaction protocols. Secondly, they might cause expensive participation cost because mobile devices are required to maintain a longtime connection with the network when mobile agents are employed. This thesis proposes a feasible FIPA-compliant framework to address these problems. The standard agent communication language and interaction protocols defined by the FIPA specifications are incorporated in the framework. This enables agents to talk in multiple marketplaces. The use of mobile agents is improved to reduce connection time and simplify the manipulation of mobile devices.

Acknowledgment

I would like to thank my supervisor, Dr. Thomas Tran, for his constant guidance, motivation, and support throughout my master's program. I would also like to acknowledge constant encouragement and appreciation from my parents through all stages of the program. Finally, I would like to extend my thanks and gratitude to all others who helped me to succeed.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ACL	Agent Communication Language
B2B	Business-to-Business
B2C	Business-to-Consumer
C2C	Consumer-to-Consumer
CFP	Call for Proposals
CORBA	Common Object Request Broker Architecture
FIPA	Foundation for Intelligent Physical Agents
GSM	Global System for Mobile communication
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IIOP	Inter-Object Request Broker Protocol
J2ME	Java 2 Micro Edition platform
JADE	Java Agent Development Environment
LAN	Local Area Networks
MAS	Multi-Agent System
MASIF	Mobile Agent System Interoperability Facility
OMG	Object Management Group
PDA	Personal Digital Assistant
SMS	Short Message Service
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discovery and Integration
UML	Unified Modeling Language
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WS	Web Services
WSDL	Web Service Description Language
XML	Extensible Markup Language

Chapter 1 Introduction

1.1 Introduction

As a growing area of electronic commerce, Consumer-to-Consumer (C2C) applications have helped individuals to conduct their own e-business transactions. Without a sole reliance on companies any longer, consumers are increasingly connected within e-business via such channels as online auction sites, classified ads sites and for-sale forums or newsgroups. Nowadays, mobile devices such as personal digital assistants (PDA) and mobile phones are gaining popularity among business and consumer users. Many people professionally use mobile devices to access any type of information through the Internet. According to JupiterResearch, there were nearly 195 million mobile phone users in the world in 2006 [21]. Ever-increasing mobile phone users are pinpointed in e-commerce markets. Advances in mobile and wireless technologies are making anywhere and anytime computing a reality. As a result, consumers become mobile because new wireless communication channels are more and more applied in the e-commerce domain.

Obviously, e-commerce environment will be more complicated and extended with the applications of agent and mobile technologies. Growingly accepted into normal daily life, agents (also called personal agents) are being deployed in mobile devices, to cater the demands of mobile users, who desire to the ability of participating in e-businesses from anywhere and anytime. Therefore, it is necessary and important to construct frameworks for developing e-marketplaces that facilitate the transactions between consumers with mobile access and user support capabilities. In the next section, we will discuss the emerging challenges in the process where the traditional e-business structure evolves into the new mobile e-commerce environment. After outlining the potential problems, a framework is proposed to address the challenging issues.

1.2 Problem Description

Compared to traditional Internet-based e-commerce applications that are typically developed over the Web architecture for human-computer interactions, agent-based e-marketplaces do not require that consumers have to login to the intended Web sites from their static personal computers in person. Nonetheless, most agent-based application developments involve wired infrastructure and do not satisfy the mobile demand due to their lack of a wireless channel for consumers' participation. With the convergence of Internet and mobile communication networks, mobile e-commerce is approaching the business forefront and enables the conduct of business and services over mobile devices.

Indeed buying small things through mobile devices brings great convenience to mobile consumers and saves them a lot of time. Technically speaking, personal agents are deployed and customized in mobile devices to meet the demands of consumers, whatever they want and whenever they want (e.g. [11][17][22][26]). However, it should be noted that mobile devices suffer not only from the limitations of battery life, memory, data storage, and computing power, but also from small screens, cumbersome input, and limited network bandwidth and network connections [59]. Most existing mobile e-commerce applications (e.g. [5][22][33][37]) have introduced mobile agent technology to overcome the limitations of mobile devices. Mobile agents are initiated from mobile devices, are able to move to e-marketplaces through the Internet, and conduct trading on behalf of buyers or sellers. These applications, nevertheless, do not address and solve the potential issues that they could involve long-time manipulation of mobile devices and require costly long-standing connection with the Internet. Also, they are standalone applications without an interface for linking to other mobile e-commerce systems. They only support their own agents and thus might not be interoperable with other e-marketplaces.

Furthermore, the distributed nature of e-commerce indicates that consumers may require access to worldwide Internet markets. Obviously, those standalone applications could not interconnect with each other and thus could not form a scalable mobile e-commerce community. If they could comply with a commonly recognized standard, they would be more interoperable, can be integrated together and provide dynamic connectivity among

buyers and sellers in the multiple e-marketplaces. Mobile agents therefore need to operate in heterogeneous and dynamic e-marketplaces. Hence, mobile agents will have to comply with a set of standards concerning the agent communication language and interaction protocols to be used. Most existing e-marketplaces have been designed with a single communication language or interaction protocol. Consequently, local agents are not able to communicate with other remote agents from outside these systems. Mobile Agent System Interoperability Facility (MASIF) [34] is a standard of Object Management Group (OMG) that provides definitions and interfaces for the interoperability between mobile agent systems. MASIF focuses on standardizing agent management, agent transfer, and names for agents and agent systems. MASIF extensively addresses inter-system agent communication by Common Object Request Broker Architecture (CORBA). However, MASIF does not address the issue of agent communication. Therefore, no communication or negotiation protocols are defined and these are left to implementers.

1.3 Motivation

Mobile e-commerce means making online purchases and/or transactions anywhere and anytime. We believe that the next phase of electronic business growth will be in the area of mobile e-commerce. In Section 1.2, we already discussed the issues emerging in the existing mobile e-commerce applications. We should take these issues into consideration when designing a framework for the mobile C2C commerce. We strongly believe that an approach combining agent mobility and intelligence will offer us abilities to design and develop a multi-agent e-market framework. We can deploy some intelligent agents that cooperate with each other, represent the user's needs, and contact appropriate partners in the marketplaces when some pre-specified condition holds. As the remote representatives of users, mobile agents will migrate over the Internet anytime to perform trades on behalf of mobile device users, thus mitigating the suffering of mobile devices. Although there is not a universally accepted set of standards for developing multi-agent systems, the Foundation for Intelligent Physical Agents (FIPA) [10], which aims at providing one language commonly understood by most participants in e-commerce, is obtaining a growing acceptance. Agent

communication language is well-formed defined by FIPA whereas it is not addressed by MASIF. With FIPA becoming a *de facto* standard in this field, we are able to develop a FIPA-compliant mobile e-commerce system with enhanced interoperability.

1.4 Thesis Objectives

The main aim of this thesis is to construct a feasible framework that is:

- Based on mobile, intelligent agents. Agents can pro-actively monitor trading opportunities, search for trading partners, and make trading decisions to satisfy users' preferences in an e-marketplace and visit other e-marketplaces if required. A trade pattern is also designed for reducing users' participation cost via mobile devices.
- Compliant with FIPA standard. This is to improve the interoperability of the framework, which can be integrated with other FIPA-compliant e-marketplaces, thus enabling agents to migrate among them.
- To resolve the current constraints in mobile devices. The framework is to assist mobile users to perform tasks in e-commerce by manipulating their mobile devices with minimal intervention.
- Extensible. Our current work focuses on employing mobile agents, which act on behalf of consumers and migrate to remote marketplaces to participate in a C2C e-commerce application. As an expansion from online B2C e-commerce, C2C transactions will continue to grow in popularity and provide facilities for B2C and even B2B exchange. One obvious reason is that individual consumers can become small businesses when they post items or services for sale more often. Thus, the framework should also extend, encourage and assist e-commerce not only between consumers but also between businesses.

1.5 Thesis Contributions

This thesis proposes a feasible framework that combines agent mobility and intelligence for consumer-oriented e-commerce applications. The main contributions of this thesis are listed as follows:

- A paradigm is presented for future mobile e-commerce applications—an anytime anywhere mobile multi-agent e-market framework—that can be integrated with other global multi-functional e-marketplaces in a cost-effective manner.
- This framework can complement the current Web-based Internet commerce systems via means of mobile agents. Mobile devices would not suffer limitations and mobile users survive from expensive participation cost.
- This thesis provides an implementation to evaluate the feasibility of the conceptual framework. This implementation lays a foundation for practical applications in the future.
- The agent communication languages and interaction protocols are based on FIPA standard. This technology further ensures an interoperable architecture for a fully distributed dynamic wireless e-commerce environment.
- This framework has been exposed to the research community of the field and resulted in two technical paper publications: One paper was published in the International Journal of Information Technology and Web Engineering [60] and another was published in the Proceedings of the Sixth International Conference on Mobile Business [61].

1.6 Summary and Thesis Organization

Chapter 1 provides a brief introduction on why mobile and agent technologies should be integrated into e-commerce applications during the evolution of e-commerce. This chapter also points out the possible problems commonly faced by most agent-based e-commerce systems. In addition, the objectives and contributions of this thesis were introduced. The following outlines the remaining structure of this thesis.

Chapter 2 introduces the background knowledge and related work regarding mobile agents, which are necessary investigations for the compilation of the requirements for the proposed framework. This chapter is intended to establish a firm understanding of mobile agent technology and survey the current state-of-the-art technology in existing mobile agent-related research.

Chapter 3 provides a detailed specification of the system framework, discussing the basic concepts, functions, and features. This chapter also addresses generic operations and possible scenarios. All types of agents are explained in aspects of their functionality. Mobile agents are also characterized under two areas, migration behavior and negotiation process.

Chapter 4 presents a reference implementation of the framework, as a proof of concept, using a scenario as an example for simulation. This chapter also describes the design methodology of the framework, presents the different technologies used in the framework, and summarizes some lessons learned.

Chapter 5 analyzes the benefits that our framework brings in and various existing problems based on the experiments. Chapter 6 concludes the thesis and discusses future work.

Chapter 2 Background and Related Work

2.1 Agent Technology

2.1.1 What is an Agent?

Agent technology is one of the most promising areas of software applications, increasingly emerging as a new paradigm for developing e-commerce applications. Agent paradigm has been claimed to be the next breakthrough in the development of innovative e-commerce applications [26]. This thesis is concerned with software agents and adopts the definition of agents proposed by Object Management Group CFTF RFP3 in Section B2.1 [38]: "Agents are programs that have an ability to move and have an ability to start executing autonomously." An intelligent agent is a software program, which differs from the traditional one, is widely characterized by five properties: autonomy, reactivity, social ability, pro-activeness and mobility [63], exhibited as the following concepts:

(1) Autonomy: a software agent is an autonomous entity, able to control its actions and to operate without the direct intervention of humans or other systems.

(2) Reactivity: a software agent is aware of its situated environment and responds to changes occurring in the environment.

(3) Social ability: a software agent cooperates with other agents (and possibly humans) to fulfill its tasks via agent communication language.

(4) Pro-activeness: a software agent is capable of behaving in a goal-directed manner by taking the initiative.

(5) Mobility: a software agent has the ability of moving among the hosts in the network.

In the argument over the relationship of agents and objects, some hold that agents are objects, and others hold that objects are agents. Below we offer some points of view on how the relationship of objects and agents can be defined:

(1) Agents are similar to objects. Like objects, conceptually agents are objects, in the sense that agents have identity (to tell one agent from another), have their own state and behavior, and have interfaces by which they communicate with each other.

(2) Main differences between agents and objects:

- Agents are autonomous and reactive-- they can individually decide whether to respond to messages from other agents. In contrast, objects do what they are asked to do. Agents model things in an active manner; objects might be considered to model things in a passive manner.
- Intelligent agents use agent communication language to communicate with each other. They can change their behavior based on what they learn, and respond to different messages over time. Objects, by contract, use a fixed set of messages in communication.

Shoham [46] proposed an Agent-Oriented Programming (AOP) paradigm that introduces a formal language with syntax and semantics to describe agents in terms of the mentalistic, intentional notions that represent the properties of agents. Shoham's first attempt at an AOP language was the AGENT0 system. In the AGENT0 programming language, an agent is specified in terms of a set of capabilities (things the agent can do), a set of initial beliefs and commitments, and a set of commitment rules which determine how the agent acts [57].

2.1.2 Agent-based E-commerce

Software agents can autonomously mediate purchases for consumers when they shop over the Internet. In traditional e-commerce, consumers typically visit many websites to search and compare the prices of particular products. This tedious and time-consuming process could involve consumers gathering all relevant information and considering the availability of products in person. However, with agent technology being applied to the e-commerce domain, consumers can delegate an agent to handle all of the information gathering, price

comparing, decision-making, and payment processing. In this case, consumers simply state what they want and do not need to direct an agent manually how the task is done. Several wired agent-based e-commerce systems have been proposed to address these problems. Kasbah [4], for example, serves as an electronic marketplace where buying and selling agents can carry out trade transactions. Bidding agents are realized on eBay [9], the world's number one online auction site, to facilitate individuals to trade their privately owned items. Ebay is considered one of the most successful C2C e-businesses. Other examples of C2C applications, such as Monster.com and Workopolis.com, provide a valuable service for consumers to look for jobs. Consumer-to-Consumer applications are a growing area of e-commerce. With the expansion of online business, peer-to-peer transactions in C2C continue to grow in popularity. Nonetheless, these existing systems do not satisfy the mobile demand of consumers due to their lack of providing a wireless channel for consumers to participate.

2.2 Mobile Agent Paradigm

2.2.1 Introduction of Mobile Agent

In contrast to stationary agents that execute only on the system on which they are started, mobile agents are defined as programmed entities that travel autonomously and freely through a computer network. On behalf of users, mobile agents fulfill a task specified by their users, such as gathering information, or getting closer to the necessary resources to exploit them locally rather than remotely. A mobile agent is not bound to the system on which it begins execution, and hence can be delegated to various destinations [23][27]. Started in one execution host, it has the capability of carrying its state and code with it to remote servers, where it resumes execution [28]. Mobile agents are a specific form of mobile code and execution state. The term "state" typically means the attribute values of the agent that help it determine what to do when it resumes execution at its destination. In the object-oriented context, "code" means the necessary class code for an agent to be executed. During the self-initiated migration, mobile agents work autonomously and communicate with other agents and host systems.

Several mobile agent systems have been designed in recent years. Telescript [62] is the first commercial mobile agent system developed by General Magic. Telescript provides transparent agent migration and resource usage control. Aglets from IBM [27] is also a mobile agent system based on the concept of creating special Java applets (named aglets that are capable of moving cross the network). JADE [1] is one of the agent development tools that can support efficient deployment of both agents' mobility and intelligence in e-business applications. As a middleware implemented in Java and compliant with the FIPA specifications, JADE can work and interoperate both in wired and wireless environment based on the agent paradigm. JADE supports weak mobility; that is, only program code can migrate while no state is carried with programs. NOMADS [49] supports strong mobility and secure execution; that is, the ability to preserve the full execution state of mobile agents and the ability to protect the host from attacks. Telescript describes a scenario in which a personal agent is dispatched to search a number of electronic catalogs for specific products and returns best prices to a PDA from where it starts [62]. An integrated mobile agent system called Nomad allows mobile agents to travel to the eAuctionHouse site (http://ecommerce.cs.wustl.edu) for automated bidding and auction monitoring on the user's behalf even when the user is disconnected from the network [40]. They all aim at reducing network traffic and latency.

2.2.2 Benefits and Applications

There are several advantages in using the mobile agent paradigm rather than conventional ones such as client-server based technologies [3][28][31][32]:

(1) Reduced network traffic by moving computation to data as compared with the client-server paradigm. The general idea is to move code close to a large database instead of transferring lots of data to a client; thus only relevant data is sent back to the client, which saves network bandwidth.

(2) Introduction of concurrency and asynchronous execution on multiple heterogeneous network hosts. This feature makes mobile agents suitable for nomadic computing,

meaning that mobile users can start their agents from mobile devices that offer only limited bandwidth.

(3) Delegation of tasks. A user can employ a mobile agent as a representative to roam the network and perform tasks for her. This goal would be very desirable in the short term while the user is not connected with the network.

The typical applications of mobile agents in the past decade included electronic commerce, personal assistance, distributed information search and retrieval, information monitoring and notification, real-time control, and parallel processing [3][28]. Recently, Satoh proposed a framework for building and operating agent itineraries for network management systems [42]. Satoh also explored mechanisms of mobile computing for mobile agents to mask temporal disconnections in the networks and to migrate through unstable networks [41]. *Flying Emulator* [43] applied mobile agent technology in software testing methodology by means of running software on mobile computers. In future, mobile agents will be useful and essential technologies in real distributed systems. In particular, we focus on the applications of mobile agents in electronic commerce. When more than one single shopping website is involved, e-commerce becomes distributed and dynamic in nature. Consequently, the mobile agent paradigm is introduced to offer more coverage of information sources. Mobile agents offer delegation of tasks and asynchronous task execution by interacting with a number of distributed websites. With the capacity of traveling the network, mobile agents can roam on the Internet on consumers' behalf to search many electronic catalogues for interested products with the best possible prices. To meet customers' needs, mobile agents can carry out such duties as choosing products and vendors, interacting with other agents, negotiating the purchase and arranging payment. With asynchronous interaction, mobile agents can perform these tasks even when consumers' computers are disconnected from the network. When consumers' computers are reconnected, mobile agents will migrate back to their hosts with results. Therefore, one obvious benefit of the mobile agent paradigm is that mobile devices need only a few seconds of connectivity. We believe that mobile agents will be deployed widely and become the main paradigm for the next generation of distributed systems.

2.3 Multi-agent Systems

A Multi-agent System (MAS) is a system consisting of multiple intelligent agents interacting with one another [39]. The agents can cooperatively share a common goal or pursue their own interests selfishly. MAS-technologies focus on the development of agent communication languages (ACL), interaction protocols, and agent architectures. An agent communication language allows agents to communicate while hiding the details of their internal workings. Two major ACLs have been introduced, Knowledge Query and Manipulation Language (KQML[1], a language and protocol for exchanging information and knowledge) and FIPA ACL (more details in Section 2.4.2). Agent interaction protocols govern the exchange of a series of messages among agents. Several interaction protocols have been devised for agent systems, such as coordination protocols, contract net protocols, blackboard systems, negotiation protocols and market mechanisms [33]. MAS-based approach is the best way to design distributed e-commerce applications. In agent-mediated e-commerce, agents operate effectively and negotiate with each other productively, exchanging goods or services. Multi-agent infrastructure facilitates the virtual electronic market, governing the interaction of self-interested agents and handling multiple and various negotiation protocols. In addition, MASs have the ability to solve problems that are too large for a centralized agent to solve because of resource limitations; to provide solutions to problems that can naturally be regarded as a society of autonomous interacting agents (e.g. a scheduling agent that manages the calendar of its user in meeting scheduling); to provide solutions that efficiently use information sources that are spatially distributed, for example, information gathering from the Internet [50].

2.4 FIPA Standard

2.4.1 FIPA Agent System

The Foundation for Intelligent Physical Agents (FIPA) is an IEEE Computer Society standards organization that focuses on the promotion of agent-based technology and issues

related to the interoperability of agents [12]. FIPA was originally formed as a Swiss based organization in 1996 and was officially accepted by the IEEE as its eleventh standards committee in 2005. Current thirty-six members of the IEEE FIPA Standards committee are mainly from famous companies (e.g. Boeing Company and Siemens) and educational institutes (e.g. Carnegie Mellon University). Some working groups and study groups have been formed and approved within the scope of the FIPA Standards Committee, such as Agents and Web Services Interoperability Working Group and Review of FIPA Specification Study Group [12]. As we introduced earlier, the JADE middleware was developed based on the FIPA specifications.

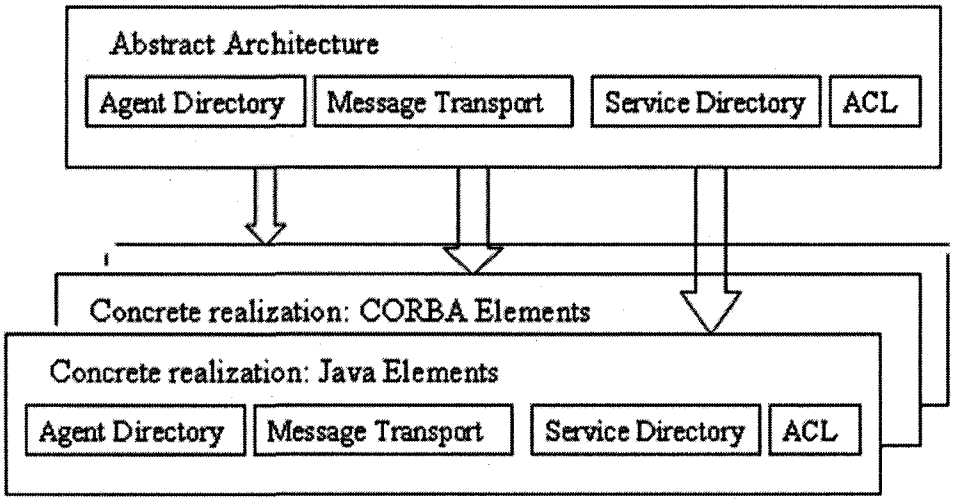


Figure 2-1: FIPA Abstract Architecture Mapped to Various Concrete Realizations [13]

FIPA defines issues on agent communication, including agent communication languages, message transport protocols, ontologies[2], and agent mobility. Figure 2-1 depicts the mapping between FIPA abstract architecture and various concrete realizations. To achieve interoperability and reusability, this architecture focuses on creating semantically meaningful messages. Agents may exchange messages using different messaging transports, different ACLs, or different content languages. This model includes Message Transport Service which

[1] More information on KQML can be found at http://www.cs.umbc.edu/kqml/.

[2] An ontology provides a vocabulary for representing and communicating knowledge about some topic and a set of relationships and properties that hold for the entities denoted by that vocabulary. More information on ontologies can be found at http://www.fipa.org/specs/fipa00086/XC00086C.html.

provides functions to support the sending and receiving of transport-messages[3] between agents, Agent Directory Service which provides a location where agents register their descriptions as agent-directory-entries (contains basic information related to how to communicate with the agent, such as agent name and location), Service Directory Service which provides a location where services can register their service descriptions as service-directory-entries (allows agents to search services which match their interest), and ACL message structure. FIPA agent message transport protocol specifications deal with different network transport protocols for delivering ACL messages. An agent system may use OMG Internet Inter-ORB Protocol (IIOP), HTTP protocol, or Wireless Application Protocol to transmit ACL messages.

2.4.2 Agent Messages

In FIPA agent systems agents communicate with each other by sending messages encoded in an agent communication language based on the speech-act theory. Originally formulated by the British philosopher John Langshaw Austin, speech-act theory categorizes speech, distinguishing between informing, requesting and offering; each communicative act involves different presuppositions and has different effects [47]. FIPA ACL is the language that provides a set of language primitives for agents to exchange regardless of the ontology or content languages they use. A FIPA ACL message consists of a set of message parameters [14]. The following parameters are some crucial ones contained in an ACL message: *performative*, which is a required parameter to denote the communicative act of the message (such as *inform, request, propose*, etc.); *sender*, which is an omissible parameter to identify the name of the agent that sends the message; *receiver*, which is a compulsory parameter to identify the name of the intended receiving agent; *content*, which expresses the meaning of the message; *language*, which denotes the language used to express the content of the message; *ontology*, which indicates the ontology used to construct the content expression. An example of ACL message structure, (inform :sender X :receiver Y :content "weather (today, raining)" :language Prolog :ontology weather), shows that the sender X informs the weather

[3] A transport-message is an encoded ACL message, plus the envelope which includes the information about how to send the message.

condition (today is raining) to the receiver Y upon Y's query (e.g. what's the weather like today?). In this structure, the programming language Prolog is used to express the content and the weather ontology attempts to describe the notion of weather.

2.4.3 Interaction Protocol

FIPA defines a set of interaction protocols that indicate certain message sequences exchanged among agents in typical conversations [15]. There are some typical patterns in the interaction protocols. Figure 2-2 depicts one example of the Contract Net Protocol, which is used in fully automated negotiation among agents, for instance, between buying and selling agents in electronic marketplaces. In this protocol, an agent can be an Initiator or a Participant or both at any time. The Initiator sends out a Call for Proposals (CFP) and multiple Participants review CFP and bid on feasible ones. Finally, the Initiator chooses the best bid and awards a contract to that Participant.

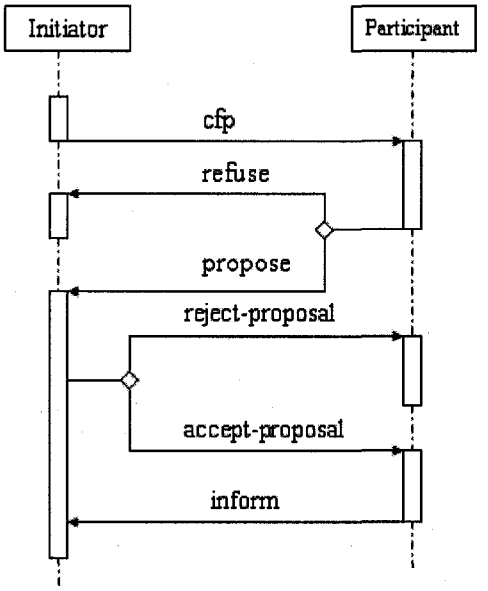


Figure 2-2: FIPA Contract Net Interaction Protocol

2.5 Mobile e-commerce

2.5.1 Definition

Mobile e-commerce [35], also known as m-commerce, is the buying and selling of goods and services through Internet-enabled wireless handheld devices such as cell phones and PDAs. As next-generation e-commerce, m-commerce enables users to buy, sell, and advertise goods and services by accessing the Internet anywhere and anytime via their mobile devices.

2.5.2 Technologies

The Wireless Application Protocol (WAP) is the emerging technology behind m-commerce that enables access to the Internet by means of a mobile phone or PDA. Inherited from Internet standards, WAP is an application communication protocol used to access services and information. WAP enables the creating of Web applications for mobile devices by using Wireless Markup Language (a mark-up language inherited from HTML but based on XML). The purpose of WAP is to enable easy, fast delivery of relevant information and services to mobile users. To fit into a small wireless terminal, WAP uses a Micro Browser that is a small piece of software that makes minimal demands on hardware, memory and CPU. To exploit the potential of the m-commerce market, some handset manufacturers such as Nokia, Ericsson, and Motorola are cooperating with carriers such as AT&T Wireless to develop and deploy WAP [36]. Most wireless handset manufacturers have adopted the WAP standard. WAP enables operators, infrastructure, terminal manufacturers and content developers to develop value-added services, for example, the delivery of mobile video, mobile game, mobile ring-tone and mobile music.

Extensible Markup Language (XML) [57], released as a World Wide Web Consortium (W3C) Recommendation in 1998, is a mark-up language that uses an XML Schema to describe data that is wrapped in XML tags and stored in plain text format. XML tags are not predefined so that XML allows users to define their own tags. Providing a cross-platform, software and hardware neutral way of exchanging information, XML has become the most common tool for manipulating and transmitting data. Making use of a more efficient

bandwidth, XML is playing an increasingly important role in the data exchange of e-commerce.

Web Services (WS) technologies are currently the most appropriate technologies to realize e-commerce applications switching from the wired Internet to the mobile Internet. The mobile Internet refers to the access of Web sites from mobile devices, which still suffers from interoperability problems. This is partly due to the incompatibility of mobile devices with different operating systems. Providing a standards-based model, WS technologies therefore offer a simple and flexible solution for binding the different platform applications together over the Internet. Web services are self-contained and self-describing application components that communicate using open protocols (e.g. HTTP). WS can offer application components like currency conversion, weather reports or even language translation as services. WS architecture, based on XML, consists of some critical technologies including Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Universal Description, Discovery, and Integration (UDDI) [7][56]. SOAP provides a framework for packaging and exchanging XML messages while WSDL describes Web services. UDDI is used to register a Web service with the repository and find the appropriate Web service. The basic Web services platform is XML plus HTTP, where the HTTP protocol is the most used Internet protocol and XML provides a language for expressing complex messages and functions.

Short Message Service (SMS) is a service for sending text messages of up to 160 characters (the length depends on the language used) between mobile phones that use Global System for Mobile (GSM) communication [45]. SMS was first introduced in 1991 in Europe, with three basic features: high reach, low cost and high retention. SMS messages do not require the mobile phone to be active and within range. Once a message is sent, it is received by a short message service center (SMSC), which provides a store-and-forward mechanism for message sending, meaning that if a recipient is not reachable, the message is stored in the queues and delivered to its recipient at a later time when the device is within range. There are many services available on the Internet that permits users to send text messages to a mobile device free of charge.

2.5.3 Applications and Limitations

M-commerce can be applied to a B2C area, such as mobile shopping, mobile banking, mobile retailing, mobile auction, mobile ticketing, and mobile advertising. It can also be applied to a B2B area. Leung & Antypas [30] suggested that mobile commerce can enhance business efficiency by distributing information to the workforce remotely and by offering new channels on which to interact with customers. Varshney et al. [53] further suggested that organizations that are capable of harnessing the power of mobile technologies to automate and streamline business processes may reap the benefits of improved productivity, lowered operational cost, increased customer satisfaction, and improved decision-making. Compared to traditional e-commerce, m-commerce has some different features: (1) ubiquity: business entities are able to discover customers anywhere and anytime through mobile devices; (2) personalization: information and services are personalized to present to the specific users; and (3) flexibility: mobile users may be engaged in running activities because of the portability of mobile devices.

Internet technologies are mostly designed for desktop computers in reliable data networks and supported with high bandwidth connectivity. Compared with desktop computers, mobile devices present a more constrained computing environment [54]: less powerful CPUs, less memory, smaller displays, and clumsy input. Compared with wired networks, wireless data networks present a more constrained communication environment than traditional networks [54]: less bandwidth, more latency, and less connection stability. Nonetheless, mobile devices also open the door to new applications and services, by following their owners wherever their owners go. They make it possible to access the Internet while driving, looking for a nearby restaurant or gas station; or to arrange for taxis to come and pick us up after business meetings; or to launch a mobile agent to perform remote e-business. With benefits as discussed in Section 2.2.2, mobile agent paradigm is a promising technology to overcome those constraints on which mobile devices impose. We discuss more mobile agent applications in the next section.

2.6 Related Work

Mobile agents have been recognized as a promising technology for mobile e-commerce applications. The interest of implementing mobile agent systems for mobile devices has increased during the recent years. The growing research on deploying mobile and intelligent agents in advanced e-commerce includes location-aware, mobile and networked comparison shopping, and mobile contract negotiation.

2.6.1 Location-aware Shopping

Impulse [17] explores a scenario in which e-commerce meets concrete commerce through a system of buying and selling agents representing individual buyers and sellers that undertakes multi-parameter negotiation and runs on wireless mobile devices. *Impulse* deploys personal agents on mobile devices to help users seek agreement on the terms of purchase. In this setting, users are not limited to performing comparison-shopping from their homes or offices because mobile devices can be carried by them around wherever they go. However, the personal agents are directed to move to online shops to participate in negotiations. This results in a potentially long-time and costly connection with the Internet.

Agora [11] is a project conducted at HP Labs to develop a test-bed for the application of agent technology to a mobile shopping mall. A typical scenario consists of mobile shoppers with PDAs interacting with store services while in the mall, on the way to the store, or in the store itself. The Zeus agent toolkit, developed by British Telecommunications, was used to implement all agents in the *Agora* project. Only the infrastructure agents speak the FIPA ACL, causing the framework to conform partly to FIPA. Although more effort in conformance is needed, the purpose of the *Agora* project is to gain experience in agents' communication protocols and to realize the significance of architectural standards.

EasiShop [22] introduces a context sensitive system using mobile agent interaction facilitated by the Bluetooth wireless transmission medium. It describes a scenario in which a shopper equipped with a PDA is alerted to services and products when passing by retail outlets. A shopper barter agent on the Bluetooth-enabled PDA might migrate to the marketplace

installed with an Agent Virtual Machine and negotiate the price with the retailer barter agent. Bluetooth [2], which is a radio standard and communications protocol, primarily provides wireless connection and information exchange for mobile devices in short range. This framework is also implemented through the Zeus agent toolkit.

2.6.2 Comparison Shopping

Mihailescu and Binder [33] proposed an agent-based framework for m-commerce, which provides three kinds of agents, i.e. device agents, service agents and courier agents. The device agents reside on mobile devices and provide access to wireless services. The service agents are owned by service providers that handle service requests from users and only operate within the wired network. The courier agents are lightweight agents that can migrate from a service agent to a device agent. However, they are single-hop agents that cannot travel back to the service agent or mobilize to another location. A test bed has been implemented for a shopping center scenario where consumers do product comparison by accessing a Web portal wirelessly via their PDAs.

MAgNET [8] is a mobile agent-based system for networked electronic trading, developed with Java and IBM Aglets SDK at University of California. MAgNET presents some decision-making capabilities that allow the shopping agents to compare quotes from different seller sites that they can visit. In this framework, a buyer dispatches a mobile agent that compares quotations for a product obtained from those online suppliers by moving among their sites in its itinerary. The mobile agent then returns to the buyer with the best offer that it has obtained and the reservations it has made. A supplier can also create a mobile agent to survey customer response from potential buyers.

2.6.3 Mobile Contract Negotiation

InterMarket [25] is a project based on a mobile agent system, Tracy (a general-purpose framework developed at the University of Jena). InterMarket proposes intelligent trading agents to automate the users' decision-making and negotiation tasks in e-marketplaces, and mobile agents to support deployment of the trading agents and offer mobile access and

communication to e-marketplaces from mobile devices. Its purpose is to enable mobile access and automated trading in e-marketplaces based on integration of mobile agents and intelligent decision-making agents, demonstrated as an add-on commercial component in a real-world e-commerce trading scenario. The work on the InterMarket attempts to demonstrate several benefits such as reduced participation costs, easy access from different mobile devices and increased efficiency of trading in e-marketplaces.

DynamiCS [52] is an actor-based framework, proposed by University of Hamburg, to implement a negotiation strategy into mobile agents. The framework is based on a plug-in mechanism enabling dynamic composition of mobile negotiation agents and involves integration of intelligent decision-making capabilities into mobile agents. It also uses rule management mechanisms to manage actors and coordinate plug-in mobility.

2.6.4 JADE-LEAP-based Applications

Moreno et al [37] used JADE-LEAP (JADE Lightweight Extensible Agent Platform) to implement a personal agent on a PDA. This agent belongs to a multi-agent system that allows the user to request a taxi in a city. The personal agent communicates wirelessly with the rest of the agents in the multi-agent system, in order to find the most appropriate taxi to serve the user.

IMSAF [5] is a framework designed to fulfill the requirement of impulse-introduced mobile shopping and implemented using JADE-LEAP tools. Six agents are instantiated on the mobile device when a mobile shopper turns on it. IMSAF is designed to assist users to make customizable decisions when impulse-induced purchases happen. LEAP can also be used to deploy multi-agent systems spread across mobile devices and servers; however, it requires a permanent bi-directional connection between mobile devices and servers. Considering the current expensive connection fees for cell phones, such a required permanent connection is not affordable for consumers in practice.

2.6.5 Challenging Issues

The transition from traditional e-commerce systems to mobile e-commerce systems offers many potential advantages. Nevertheless, the adoption of m-commerce is still hindered by many obstacles as we mentioned earlier in Section 1.2, primary including mobile device limitations, high prices of wireless mobile Internet access, the lack of ubiquitous wireless network coverage, and the lack of global standards for agent communication. As a result, consumers and businesses have not fully accepted m-commerce as previously predicted. Therefore, m-commerce faces and needs to address many challenging issues. Some of the key challenges are further discussed below:

First, it is important to consider the limitations of mobile devices, such as low-battery, low bandwidth, high latency, limited computing ability and expensive connection fees. The small screens of mobile devices have created many design challenges (e.g., inadequate to display meaningful data). Also, their limited input capabilities have made applications hard to use. In addition, the high costs of Internet access together with the slow access speed have not helped to add value to the mobile commerce. Besides this, the short battery life is another limitation in mobile devices. These constraints should be taken into consideration while designing m-commerce solutions.

Second, global technical issues such as standardization and integration of different multi-agent systems must be addressed. The m-commerce field is dominated by a variety of platforms and standards. Each application differs from others in terms of agent communication languages. With the globalization of m-commerce, it introduces the question of how to interact with the different systems due to agents' mobility and agent communication languages. There are already a number of research works in mobile e-commerce mentioned earlier in this section. Most systems, we think, were designed with a single communicative protocol for all agents. This presents drawbacks due to the heterogeneity of exchanged information and leads to an inflexible environment, which is only able to accept those agents especially designed for it. The fact that consumers in our physical world may need to access the worldwide markets and distributed e-commerce environments requires the agents to operate in heterogeneous and dynamic environments and talk a

common language. An advance on this is provided by standardization activities such as FIPA, which offers formal definitions of several standard interaction protocols. An ontology is also a key technology to solve this problem, contributing to communication and interchange, and used as a method of exchanging meaning between different agents.

Third, mobile devices also introduce new risks in addition to the advantages that they offer for users. Internet security threats and privacy issues are the main concerns in a mobile e-commerce environment. Security is a critical component regarding the safety of the information exchanged over the wireless networks. Mobile devices are likely to be lost or stolen such that sensitive information becomes vulnerable. Another privacy concern for consumers is that their location might be revealed to interested businesses at all times.

Clearly, the above problems are interdependent and their solutions may affect each other. We must be sensitive to these attributes.

2.6.6 Proposed Framework

In contrast to the works presented above, we are motivated to propose a mobile agent-mediated framework that enables users' wireless participation in several e-marketplaces through their mobile devices. The mobile agents can move across the network and perform trading tasks on behalf of their users when the users are disconnected from the network. Our framework also attempts to overcome the shortcomings raised by the challenging issues. The choice of the design is based on the objectives as mentioned in the introduction and primarily inspired by the following considerations. Mobile users would be engaged in e-commerce environments from anywhere at anytime, such that they can reach out to a global market. Also, the target users' personal preferences would be considered. The proposed framework provides an interoperable solution to allow users dynamically to connect to the network by means of their mobile devices only when needed. Especially, the mobile devices will not suffer those limitations mentioned above, when an agent is deployed in the mobile device and creates the related mobile buying or selling agent in a mediator server. In this framework, the advantage of mobile agents will be more obvious if the user has a mobile phone and is

interested in minimizing connection costs. Furthermore, this framework is compliant with the FIPA standards.

2.7 Summary

In this chapter, current situations of agent technology and mobile agent paradigm were outlined, and hence gave an overview on how mobile agent techniques have been applied in m-commerce applications. It has also described some examples of mobile agents and different approaches for developing m-commerce applications. In addition, challenging issues were pointed out, as well as the directions of current research. The chapter ended by discussing the advantages and drawbacks of the existing research and introducing the proposed model. The next chapter is dedicated to describing the design of this model.

Chapter 3 System Architecture

This chapter presents the concepts and design of a mobile agent-mediated framework. First, the goal and requirements of the system are identified. Following that, an overview of the framework is provided and a trading scenario is used to illustrate the functionality of the system. Then, the components and the processes of the architecture are explained. The details of all agents in the system are summarized. Also, the activities inside the agents as well as communication protocols between the agents are detailed.

3.1 Objectives

Based on the thesis objectives that we have outlined in Section 1.4, we further discuss the goals, the requirements and the methodology of the designed framework in this section.

3.1.1 Goals

The major goal of this thesis is aimed at implementing a multi-agent e-commerce system to assist with experimentation of real-world wireless C2C e-commerce scenarios. This chapter delivers a conceptual architecture that combines intelligence and mobility to facilitate e-business in the dynamic wireless e-commerce environment. It provides mobile device users with a model for navigation of the Internet. A key requirement for agents is the ability to relieve mobile users from time-consuming e-business processes in a cost-effective manner. Agent mobility is used to avoid direct involvement from the mobile users.

3.1.2 Design Constraints of Wireless Clients

The constraints imposed by a mobile network are significantly larger than those of a typical Web browser connected to the Internet. Generally speaking, mobile devices suffer from high latency, limited bandwidth and intermittent connectivity. In the design of the wireless client-

side architecture, some objectives are adopted to address the mobile device limitations: a mobile client should connect to the network only when needed; a mobile client should consume only as much data as needed from the network; a mobile client should remain operable when disconnected.

3.1.3 System Requirements

An intelligent and mobile agent paradigm is adopted and a set of high-level requirements for the proposed framework is identified:

- Interoperability: The system should provide an e-marketplace for automated trading and allow mobile access for remote agents. The agent communication languages and interaction protocols in the system should be compliant with the FIPA standard. Therefore, the system can be integrated with other FIPA-compliant systems and agents can communicate and negotiate with each other in these systems.
- Mobility: The mobile agent should be able to migrate to remote e-markets to participate in dynamic negotiations. Mobile users should be capable of conducting e-business anyplace via mobile devices.
- Usability: Users can easily learn how to use their mobile devices to participate in the marketplaces.
- Extensibility: the architecture should not be constrained on C2C domain. For e-commerce applications, the model should also be suitable for B2C and B2B fields.

These requirements can be regarded as boundary conditions that always need to be kept in mind when discussing mobile agents with intelligent capabilities.

3.1.4 Methodology

Modeling languages for multi-agent systems help to explore the use of agents and other abstractions. Thus, the Unified Modeling Language (UML) is a good starting point to design the concepts and notations of agent systems since it is widely accepted as a *de facto* standard for object-oriented modeling. In this chapter, UML diagrams are used to describe the design

of the proposed framework. Use Case Diagrams are used to define the functionality and to describe the relationships between agent roles. Activity Diagrams are used to model the behaviors of agents. Sequence Diagrams are also used to model the interactions between agents.

3.2 Overview of Framework

3.2.1 Introduction

Fundamentally, the proposed framework offers a Web service that can be accessed from a mobile device. This Web service helps a mobile device user to conduct business in a FIPA-compliant environment via a mobile and intelligent agent. This agent will return the result of a business transaction to the user via the mobile device. The Web service will be published in the UDDI Registry Server[4] via a UDDI agent[5], to indicate that our framework complies with FIPA, provides wireless connections and supports agent migration. This framework also represents a distributed e-marketplace that allows local agent to move out and permits foreign agents to move in. In this thesis, we assume that the migration function of e-marketplaces is primarily oriented to buyers. This means that an e-marketplace assembles local selling agents and attracts local or remote buying agents to participate in the trading. Local selling agents do not move out but local buying agents are allowed to move to remote e-marketplaces. We will further discuss this assumed feature in Section 3.4.2.6. Figure 3-1 shows a distributed C2C wireless e-commerce environment and a traditional wired e-commerce one.

[4] UDDI Registry Server [10] enables businesses to publish service listings and discover each other, and defines how the services interact over the Internet.

[5] Detailed descriptions of the UDDI agent and publishing are provided in Section 3.4.2.5.

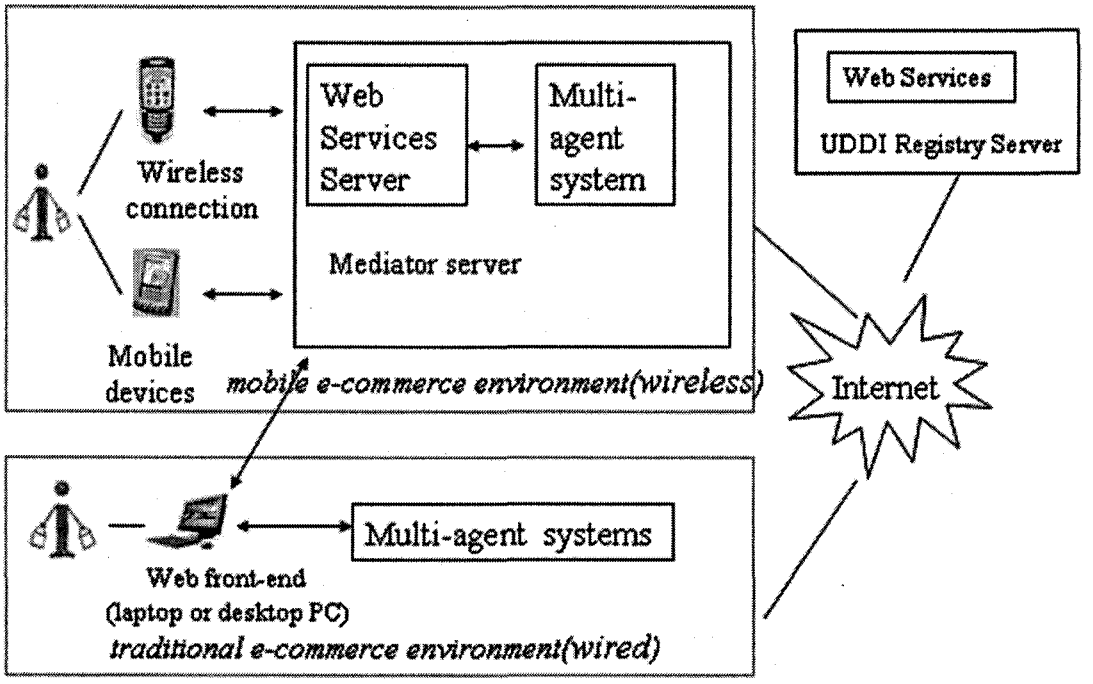


Figure 3-1: Distributed E-commerce Environment

Consumers can connect their mobile devices, such as a PDA or mobile phone, to the mediator server through wireless connection and then send a request for creating a mobile buying or selling agent to undertake a specific business task on their behalf. The mobile agents will autonomously travel to multiple agent-based servers on the Internet. These agent-based servers also publish their Web services in the UDDI Registry Server and can be discovered by the mediator server via the UDDI agent. Also, these agent-based servers offer places for selling and buying agents to meet and negotiate with one another. The mediator server sits in the fixed network and provides services such as generating mobile agents according to various consumers' needs. The proposed mediator server contains two main components: the Web services server, which offers Web services using standard technologies (e.g. WSDL, as we introduced in Section 2.5.2) for service descriptions and deployment, and the multi-agent system, which manages the agents and plays the role of a marketplace similar to an agent-based server. Also, the mediator server provides a Web-based interface, as shown in the figure, for a consumer to connect a laptop or a desktop PC to the network and launch

an agent to execute on the mediator server[6]. In this thesis, we focus on the electronic trading of second-hand products for owners of mobile devices.

Web services server and multi-agent system are not new, respectively. The main idea is to combine these two technologies into the mediator server. In our architecture, a consumer will request the mediator server to create a buying or selling agent and then dispatch it to agent-based servers on the Internet. The major operation that occurs in an agent-based server is price negotiation, where buying agents negotiate price with selling agents. According to the consumer's preferences, the buying agent may travel to different e-market sites, known by the mediator server, to seek goods when the consumer desires to conduct a global multiple markets comparison. The consumer's preferences can be represented in XML format.

3.2.2 A Trading Scenario

To understand the environment better, let us consider a typical scenario taken from daily life, where two hypothetical customers, named Mary and Tom, try to participate in an eBay-like auction. Mary wants to sell her used Sony MP3 player. At her office, she initiates a selling agent from a PDA, through a wireless LAN connection with the mediator server in the building. Then this selling agent lives in the server and waits for potential shoppers. Due to some unpredictable event, Mary may have to leave her office and cannot access the selling agent via her PDA (because of no available wireless LAN network coverage). However, she is able to reconnect later on.

Haphazard, Tom enters his buying preferences into his Java-enabled mobile phone, trying to buy a second-hand Sony MP3 player under a maximum price. The personal agent on his mobile phone establishes a connection with the mediator server in his community and asks the server to launch a mobile buying agent according to his preferences. Then Tom disconnects his cell phone from the server. The mobile agent knows where and how to migrate, as instructed in the migration itinerary. As days pass while this buying agent is roaming around the Internet, it enters into Mary's mediator server and searches for services

[6] In the experiment (described in Chapter 4), we developed a GUI for users to launch a buying or selling agent.

provided. After the negotiation between the selling agent and buying agent, they reach an agreement on the item and price. With that, the buying agent will return to its host server and sends a SMS-based notification to the personal agent running on Tom's mobile phone, about the potential seller gathered from the Internet. Also the selling agent sends an Email-based notification to the personal agent running on Mary's PDA.

Finally, things left to Mary and Tom seem to be simple and easy since they could find either the cell phone number or the email address from the information reported by the personal agents, respectively. As we have seen, mobile consumers only need a small bandwidth connection twice, once for initiating a migrating mobile agent and once for collecting the results when the task is finished.

3.2.3 Roles and Functionality

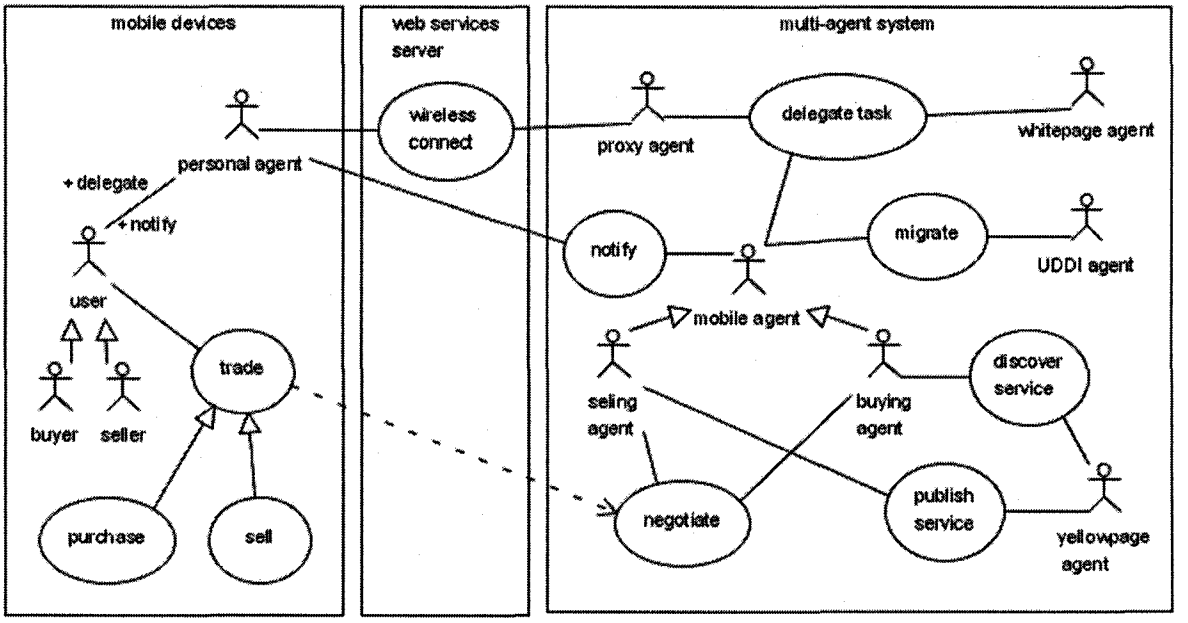


Figure 3-2: Use Case Diagram including Agent Roles

Figure 3-2 describes the use case diagram that models the trading scenario. Use cases are an effective way to capture the functional requirements of this system. The approach is to use the concept of agent roles in the use cases to capture their behaviors. These behaviors define the communication and cooperation between agents. First, the different agent roles need to be

specified in this system. From the system's point of view, a user can be viewed as a kind of special agent, and two roles, buyer role and seller role, are developed in the trading case.

Considered as a true representative of the user, a personal agent that resides on the mobile device is needed to interact directly with the user. The personal agent represents the user's interests and allows the user to have a choice of delegating desired tasks, purchasing or selling. The personal agent needs to convey the user's tasks to the proxy agent in the multi-agent system via wireless connections. The tasks will be delegated to the mobile agents. Before the delegation, collaboration between the proxy agent and the white-page agent is required for the creation of either mobile buying agent or selling agents. On the behalf of the users, the buying agent and the selling agent enter into the marketplace. Selling agents publish their products or services to the marketplace via the yellow-page agent, and buying agents query the yellow-page agent in order to discover relevant services. With common interests, buying agents and selling agents meet together, negotiate with each other, and make a purchase decision. Upon a task finished, the mobile agents are responsible for notifying the user via the personal agent. Both the buyer and the seller are instantly informed about the results of trading. If for some reason the user would like to conduct a global marketing, mobile agents need to migrate to several remote marketplaces. A UDDI agent provides information about remote sites for mobile agents' migration. The details of agents' roles and responsibilities will be refined in Section 3.4.

3.3 Architecture

3.3.1 Hardware Components

From the hardware components perspective, as described in the upper-left block of Figure 3-1, two classes of components exist in the scope of the conceptual framework, mobile devices and a mediator server. The relationship between them is many-to-one, that is, many mobile devices can wirelessly connect to one mediator server. Carrying the mobile devices, the user can move around within the wireless coverage. From the functional aspect, as we can see from Figure 3-2, the mediator server is decomposed into two parts, the Web services server,

which facilitates the personal agent to interface with other agents in the multi-agent system, and the multi-agent system, which manages the agents and plays the role of a marketplace similar to an agent-based server. From the deployment's point of view, the Web services server and the multi-agent system can be deployed in the different servers. The mediator consequently is viewed as a server cluster and such a cluster provides a stronger computing capability and more resources. The Web services server consists of one or more components, each of which is called the Web container that controls servlets'[7] execution and lifecycle, as depicted in Figure 3-3. From the networking perspective, this system allows two types of connections, wireless connections for mobile devices connecting to the mediator server and Internet connections for the mediator server connecting to other servers on the Internet.

3.3.2 Software Components

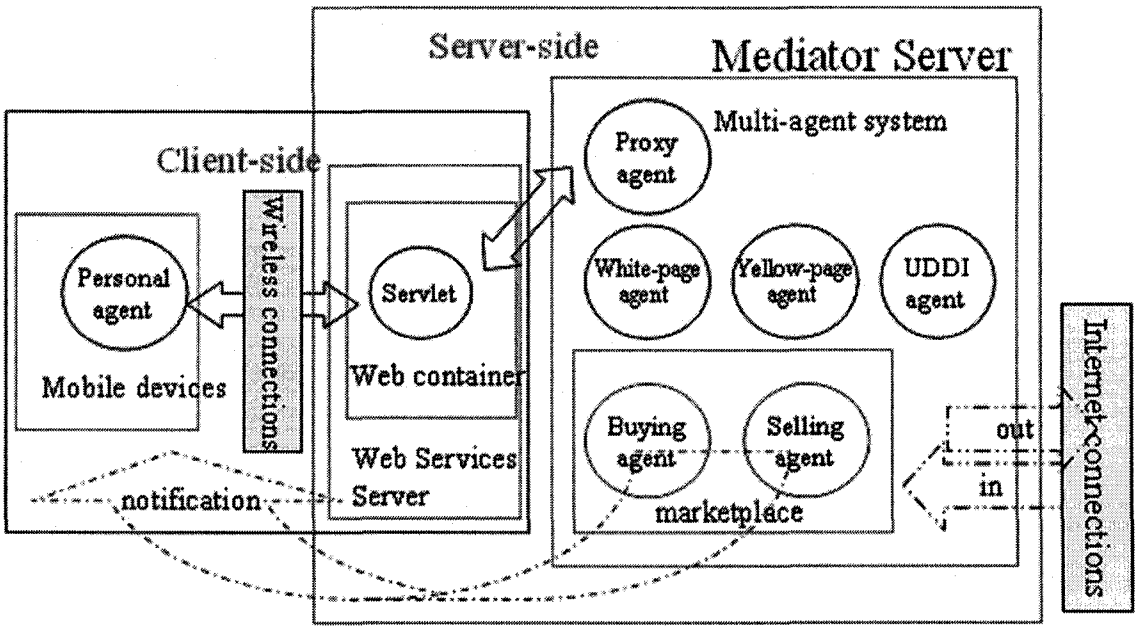


Figure 3-3: Architecture Components

From the software components perspective, as illustrated in Figure 3-3, this architecture is split into two tiers, the client side and the server side. As we can see, these two tiers overlap

[7] Servlets are modules running in a web container to answer client requests, most commonly used with HTTP.

on the Web container component. Therefore, the Web container component can be viewed as the intermediary between mobile devices and the multi-agent system. The Web container component constructs a bridge for communication between mobile devices and the multi-agent system. Both sides provide places for agents to live. Consequently, agents are primary software components in the framework.

3.3.2.1 Client-side Agent Components

On the client side, the personal agent running on the mobile device provides a graphical user interface (GUI)[8] for its user. As a manager of the user's personal preferences, the personal agent gathers user trading item information through the user's input and displays results to the screen of the mobile device after receiving a notification from the mobile buying agent or selling agent. It is also in charge of connecting the mobile device to the mediator server. In the communication phase, the personal agent provides the functions for encoding the request data into XML format and parsing the XML-based response message into text format.

3.3.2.2 Server-side Agent Components

On the server side, the servlets in the Web container transmit the personal agents' requests of creating mobile agents to the multi-agent system. The multi-agent system provides four interfaces in total: the creation interface, the marketplace interface, the migration interface and the notification interface. The creation interface handles all requests from the servlets via a proxy agent. The proxy agent[9] is an agent in the multi-agent system that interacts with the servlets. The marketplace interface offers a place for the negotiation between buying and selling agents. The migration interface enables the mobility capacity of mobile agents, allowing local agents to travel over remote systems and permitting remote agents to migrate into the local system. The notification interface provides a function for buying and selling agents to inform end users, by two approaches: notifying mobile phone users using SMS-based messages or notifying PDA users using email-based messages. The mediator server

[8] The user interface requirements for mobile devices are different from those for desktop computers. We will discuss it more in Section 4.3.1.

[9] So called "proxy", more details in Section 3.4.2.2.

therefore provides the required supports for the creation of mobile agents, messaging among agents, as well as the migration, collaboration, destruction and control of mobile agents. The JADE mobile agent framework has been proposed to provide such a supporting environment. Obviously, any multi-agent system can be used here as long as it provides the required supports mentioned above.

3.3.3 Operation Process

This section explains how the proposed system works as a whole. Figure 3-4 illustrates the system operation process and introduces the agents existing in the system. Solid arrows denote communications between agents that are in direct contact and dashed arrows denote the movement of mobile agents.

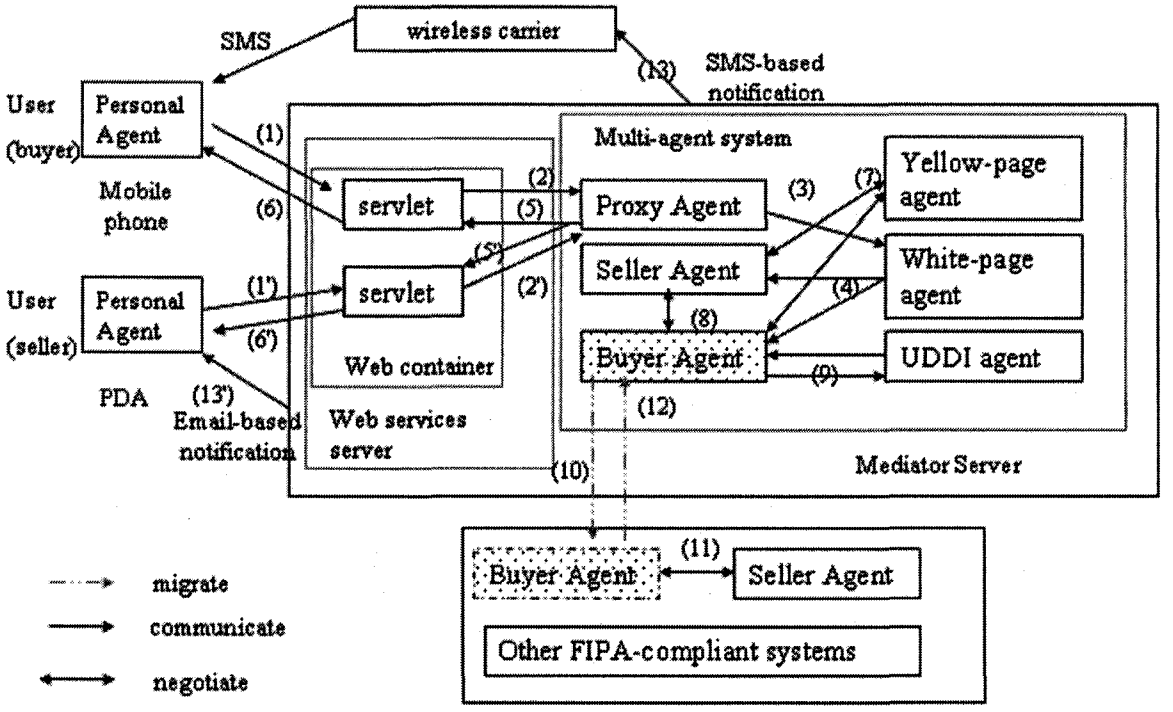


Figure 3-4: System Process Flow

As shown in the figure, mobile devices are supported by personal agents and connected to the mediator server via a wireless connection. A personal agent is a static agent running in a mobile device and offers a GUI for its user to communicate with the system. The mediator

server is connected to the Internet where other mediator servers or other FIPA-compliant systems exist. In a Web container of the Web services server, a servlet answers any requests from a personal agent and is linked to the behavior of a proxy agent in charge of handling the requests. The proxy-agent interfaces with several servlets and constructs a bridge between the Web-service server and the multi-agent system. The proxy-agent has multiple behaviors[10] to handle multiple servlets' requests in parallel. Each behavior of the proxy agent connects to the white-page agent (Agent Management Service as defined in FIPA, i.e., the AMS agent[11]), asking for the creation of a buying or selling agent in the multi-agent system, as well as providing a response to the servlets. Each buying or selling agent can connect to the yellow-page agent (Directory Facilitator as defined in FIPA, i.e., the DF agent[12]). For a selling agent, it publishes its services to the DF agent. For a buying agent, it retrieves a list of selling agents advertising services with the DF agent. A remote buying or selling agent must register in a local mediator server via the AMS agent once it migrates into this mediator server. Then it can communicate with the DF agent residing in this local mediator server.

A complete process of a user performing a trading is separated into two parts: the user delegates a task resulting in the creation of a relevant buying or selling agent, and the buying or selling agent takes up the task until the completion of the task. It is a synchronous process when the user requests the dispatching of an agent, while it is an asynchronous process when the agent is doing business on behalf of the user, as further detailed in the subsequent sections. The term "synchronous" means that the user has to keep a connection with the mediator server and can not launch a second request before a preceding request is responded. The term "asynchronous" means that the user can either disconnect from the mediator server after a request is responded or keep the previous connection for launching more requests after a request is responded.

[10] In the programming context, an agent is implemented as a class with multi-threaded capacity.

[11] Detailed description of the AMS agent is provided in Section 3.4.2.4.

[12] Detailed description of the DF agent is provided in Subsection 3.4.2.3.

3.3.3.1 Synchronous Process

This synchronous process requires that mobile devices keep the connection status with the mediator server. As illustrated in Figure 3-4, the synchronous procedures from (1) to (6) for mobile phone users (or from 1' to 6' for PDA's users) depict how users, according to their preferences, create buying or selling agents:

(1) (or 1') At this first step a user configures his (or her) preferences via his (or her) personal agent. The personal agent then establishes a connection with the server and sends an XML-based request to the servlet.

(2) (or 2') The servlet accepts the request and communicates with the proxy agent.

(3) The proxy agent cooperates with the white-page agent, which lives in the same multi-agent system to create a buying or selling agent.

(4) If the buying or selling agent is created successfully in the system it might be mobilized to other systems to undertake the task or just stay in the system for local interactions.

(5) (or 5') The proxy agent sends the agent creation information to the servlet.

(6) (or 6') The servlet responds to the personal agent and finishes its task. After receiving the response from the servlet, the personal agent informs the user of the relevant buying or selling agent being created. Meanwhile, information about the buying or selling agent is recorded in the mobile device. Subsequently, the user might cancel the task by looking up the agent.

After this synchronous process, the user can disconnect from the server at will. He (or she) can gather the task results in the following asynchronous process.

3.3.3.2 Asynchronous Process

This asynchronous process does not require mobile devices to be connected with the mediator server, because at this point, buying or selling agents can take over the task on behalf of their users. As illustrated in Figure 3-4, the procedures from (7)-(13) describe the behaviors of a buying or selling agent during its lifetime when targeting a task:

(7) From the beginning of its lifetime, the selling agent publishes the user's products or services in the system via a yellow-page agent, and the buying agent acquires a list of sellers through a query to the yellow-page agent.

(8) Once relevant products or services are matched, the buying agent and the selling agent negotiate with each other and attempt to reach an agreement.

(9) In the case of doing a global search, the buying agent might migrate to remote sites. In this case, a UDDI agent prepares a detailed migration itinerary for mobile agents.

(10) Mobile agents start their travel and prepare to enter the next possible hopping site.

(11) In the new site, mobile agents communicate with local agents. Note that remote mobile agents actually become local at this stage.

(12) Mobile agents need to migrate back to the host site.

(13) (or 13') Buying and selling agents send notifications to their personal agents.

The above is a process during which the user can disconnect from the network at will. Even if the user decides to disconnect from the network, he (or she) will still receive an SMS-based notification from the mediator server via an interface with the wireless carrier, or an email-based notification from the mediator server via an interface with a mail server, as soon as he (or she) reconnects to the network.

3.4 Different Types of Agents in the Framework

The following agents co-exist in this framework, namely personal agents, the proxy agent, the UDDI agent, buying or selling agents, the yellow-page and white-page agents. Among them only buying agents are mobile agents (selling agents' mobility is not activated according to our assumption), while personal agents, the UDDI agent and the proxy agent are stationary agents. Both the yellow-page and white-page agents are fixed at the mediator server. The details of these agents are described as follows.

3.4.1 Naming Scheme

According to the FIPA specifications, an Agent Identifier (AID), which is a globally unique identifier, identifies each agent. A very simple mechanism is used to label and distinguish an agent, in particular, a buying or selling agent. An AID is constructed by concatenating an agent name to its home site, separated by the '@' character. The agent name reflects the properties of this agent, including the functionality provided by the agent, owner, and creation time (timestamp). The home site consists of the server name where the agent is created, in convention, using a form of the type *service.organization.domain* [48] (e.g., www.site.uottawa.ca) in the context of the Internet. Therefore, a full name of an agent could be *BuyBook5550000Jan12006180001@www.site.uottawa.ca*, meaning that the user with the id 5550000 (id is generally combined with user's telephone number in this case) created an agent for buying books, starting from January 1st 2006 18:00:01. Adopting a timestamp as a part of an agent's name guarantees the name's uniqueness when the same user requests the same kind of functionality. The aim of such a naming scheme is to provide a good human readability and to select a suitable name to represent the agent in the distributed and mobile agent environment.

3.4.2 Agents in the Framework

The role and responsibilities of agents were derived from the use case diagram as shown in Figure 3-2. Responsibilities decide the behaviors of the agents.

3.4.2.1 Personal Agent

A personal agent is a stationary agent that runs on a user's mobile device and provides a graphical interface to allow the user to configure a mobile buying or selling agent from the mobile device. The personal agent can establish a wireless connection between the mobile device and the mediator server when the user needs to access the mediator server to perform tasks. It will close the connection when the user stops accessing the network. When starting the personal agent on the mobile device, the user can choose either to initiate a new buying or

selling agent or to recall a previous agent. The responsibilities of the personal agent are described in Table 3.1.

Table 3.1 Responsibilities of Personal Agent

Agent type	**Responsibilities**
Personal agent	1. Accepts users' requests to initiate or terminate trading and forwards the requests to the Web services server. 2. Allows the user configure preferences about products or services. 3. Presents incoming notifications from buying or selling agents to users. 4. Manages wireless connections between the mobile device and the mediator server.

3.4.2.2 Proxy Agent

The proxy agent is also a stationary agent that links the multi-agent system to the servlet. It is one of the agents that is always up and running in the multi-agent system. It does a job that forwards messages between the servlets and the white-page agent. We therefore call it "proxy". The proxy agent cooperates with the AMS (white-page) agent to create a buying or selling agent for each user. There is only one proxy agent per mediator server due to its unique multi-behavior ability. The responsibilities of the proxy agent are described in Table 3.2.

Table 3.2 Responsibilities of Proxy Agent

Agent type	**Responsibilities**
Proxy agent	1. Responds to the servlet's requests from the personal agent 2. Forwards servlet's requests to the white-page agent.

3.4.2.3 Yellow-page Agent

The yellow-page agent (i.e. FIPA DF agent) provides the service of yellow pages, by means of which agent can register information about available products, or find other agents providing necessary services to achieve its goal. The yellow pages contain some registries. Each registry consists of the service type, the service name or product name, and the identifier of the agent that provides the product or the service. In the case of trading, it is supposed that only sellers should have permission to advertise their products; however, buyers are allowed to post products they are looking for. Selling agents update yellow pages by publishing their services via the yellow-page agent. Once a service is no more available, selling agents must deregister it from the yellow pages via the yellow-page agent. Buying agents query relevant services from the yellow-page agent. As a response, the yellow-page agent will return to the buying agents with a list containing the identities of relevant selling agents that provide the required services. Buying agents may come from the local system or remote systems. The mechanism of the yellow-page agent allows for publishing and discovering products and services offered by an agent. The responsibilities of the yellow-page agent are described in Table 3.3.

Table 3.3 Responsibilities of Yellow-page Agent

Agent type	**Responsibilities**
Yellow-page agent	1. Provides the registration and deregistration of products or services for selling agents. 2. Offers product or service discovery for buying agents.

3.4.2.4 White-page Agent

The white-page agent (i.e. FIPA AMS agent) represents the management of white pages and provides naming services. The white pages store the information about identifiers of all agents in the system. These agents are either generated in the local system or come from remote systems. A request from the proxy agent that asks for the creation of a local buying or selling agent must be cooperated with the white-page agent. Then the created agent is

registered in the system and is assigned a valid and unique AID through the naming services of the white-page agent. By naming scheme, the naming service ensures that each agent in the system has a unique name (an AID). Both buying and selling agents update white pages by registering in or deregistering from the system via the white-page agent. All remote agents must register with the white-page agent before they can operate in a local system regardless of where they come from. Likewise, they deregister from a local system before their leaving off. At the termination stage of a lifecycle, an agent must deregister from the host system via the white-page agent where it is originally generated. The mechanism of white-page agent allows for managing all local and remote agents that are allowed to operating in the system. The responsibilities of the white-page agent are described in Table 3.4.

Table 3.4 Responsibilities of White-page Agent

Agent type	**Responsibilities**
White-page agent	1. Responds to the requests from the proxy agent for creating a local buying or selling agent. 2. Provides the naming services for assigning a unique AID to a newly local agent. 3. Manages the registration and deregistration of all agents.

3.4.2.5 UDDI Agent

The mediator server should be exposed to the Internet for foreign agents migrating into, and be able to discover other FIPA-compliant servers and then to dispatch local agents to the remote servers. This mechanism can be realized by a UDDI agent, as described in Figure 3-5. As we introduced in Section 2.5.2, UDDI provides a standard way to publish and discover Web services using XML, WSDL provides a standard way to describe Web services using XML and SOAP provides a standard way to exchange messages. In the figure, UDDI Registry Server provides a central repository and a standard way to publish and discover Web services offered by our mediator server and other servers around the world. The repository contains a set of registries. Each registry mainly consists of the name of the server that

publishes its Web services, the description of the published Web services, and the location information associated with the server where the Web service is offered.

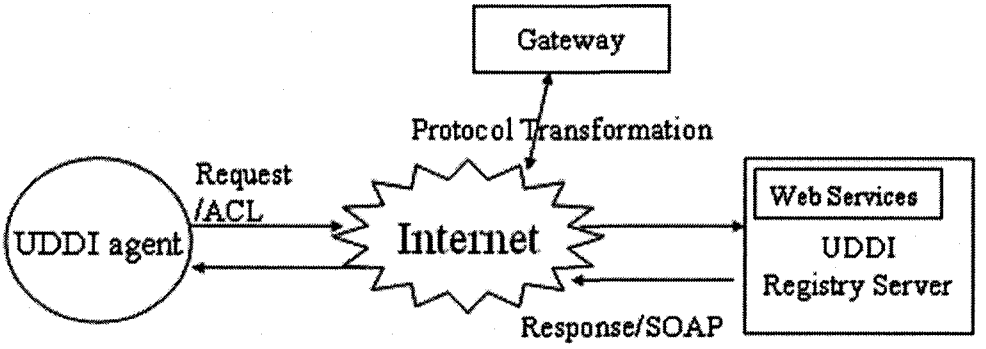


Figure 3-5: Expose or Discover Web Services on the Internet

The UDDI agent is responsible for publishing Web services to the UDDI Registry Server and discovering other Web services providers that are exposed in the UDDI Registry Server. Recall that Web services defined in this thesis means a kind of services that provide a FIPA-compliant environment for mobile agent to do trading. Note that specifications of Web services are governed by W3C [55][56], and specifications of MAS are governed by FIPA. This means that agents and Web services use different protocols, i.e., agents use ACL whereas Web services use SOAP. Therefore, we need a gateway that acts as a middleware between MAS and Web services framework without changing existing specifications of both technologies. Shafig, M.O. et al. proposed an AgentWeb Gateway middleware [44] that provides protocol transformation without changing any specifications of FIPA and W3C. Through this gateway, the UDDI agent can publish and discover the Web services in the UDDI Registry Server. By the UDDI agent's discovery, the mediator server maintains a list of remote servers' name and location information. Mobile agents will visit these remote servers in sequence according to a migration itinerary prepared by the UDDI agent. A migration itinerary contains a set of names and locations of several servers. Before launching a round trip, mobile agents request this migration itinerary from the UDDI agent. The UDDI agent is therefore responsible for organizing the itinerary and responding to the mobile agent's request. The responsibilities of the UDDI agent are described in Table 3.5.

Table 3.5 Responsibilities of UDDI Agent

Agent type	Responsibilities
UDDI agent	1. Publishes Web services in the UDDI Registry Server to expose the mediator server on the Internet. 2. Discovers Web services in the UDDI Registry Server so as to achieve a list of servers that will be visited by mobile agents. 3. Maintains the migration itinerary in the mediator server. 4. Responds to the agent requests for the migration itinerary.

3.4.2.6 Buying and Selling Agents

Buying agents are mobile agents, each of which is a peer agent of a corresponding personal agent and is involved in the traffic movement on the Internet. A buying agent first negotiates with other selling agents in the same host mediator server before migrating among multiple Web sites to communicate with other agents, provided that they can talk in a common language. The migration process of a buying agent is determined by the migration itinerary that is obtained from the UDDI agent. For selling agents, they are conceptually able to migrate over the Internet. If a selling agent wants to travel, its itinerary can also be determined by querying the UDDI agent. Recall that we made a reasonable assumption in Section 3.2.1; that is, the migration function of our framework is mostly buyer-oriented. The reason is due to a common fact that buyers do shopping by accessing multiple shops and sellers operate at the stationary sites in our real world. Therefore, such an itinerary is more appropriate for a buying agent. In addition, if a pair of potentially matching buying and selling agents migrates without any coordination algorithm, they may never meet with each other. In our design, for these reasons, we assume that only buying agents can migrate over the network and selling agents remain where they are created. Their responsibilities are described in Table 3.6.

Table 3.6 Responsibilities of Buying and Selling Agents

Agent type	**Responsibilities**
Buying agent	1. Serves the trading task from personal agent 2. Negotiates with selling agents 3. Sends a notification to personal agent 4. Migrates over the Internet
Selling agent	1. Serves the trading task from personal agent 2. Negotiates with buying agents 3. Sends a notification to personal agent. 4. Resides in the server.

3.4.3 A Prototype of Buying or Selling Agents

In this framework, the information on the GUI of personal agents presented to the mobile user reflects the properties of the buying and selling agents. A prototype for buying and selling agents is necessarily to be designed. It is presented with attributes as depicted in Table 3.7 below. This presentation means that a user will configure a mobile buying or selling agent on her mobile device, precisely according to the following characteristics. In the migration procedure, mobile agents will carry their execution code as well as these attributes.

3.4.4 Internal Behaviors in Buying and Selling Agents

The actual job an agent has is typically carried out within the agent's behaviors. The activity diagrams are used to describe the behaviors of agents. Figure 3-6 shows a general activity process for buying and selling agents: registration, migration, negotiation and deregistration (Note: as discussed before, selling agents have no migration process because their mobility will not be activated in our application). More details will be explained in the following subsections for buying agents and selling agents respectively.

Table 3.7 Attributes of Buying and Selling Agents

Attributes	**Description**
Agent type	Indicates the type that user can choose, either a buying agent or selling agent.
Agent server	Indicates the configuration of the mediator server address.
User id	Indicates the user identification: an email address, cell phone numbers or IMEI (International Mobile Equipment Identity).
Item	Describes information about the predefined product.
Quantity	Describes the numbers of predefined product.
Price	For a buying agent, this is the maximum price the agent can bid; for a selling agent, this is the minimum price the agent can accept.
Current Price Inquired	For a buying agent, this is the best price offer collected from the Internet.
Lifetime	Indicates the total time an agent can be away before being recalled or terminated.
Mobility	Specifies whether the user desires to enable the agent's migration ability, i.e. in the context of a local single or global multiple market comparison.
Per-server Activity Time	Indicates the maximum time an agent can spend on each server before migrating.

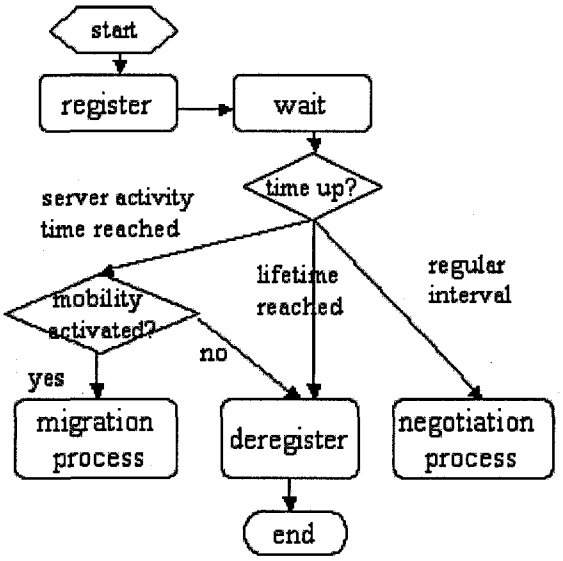


Figure 3-6: Activity Diagram for General Behaviours of Agents

3.4.4.1 Behaviors of Mobile Buying Agents

As illustrated in Figure 3-7(a), a mobile buying agent starts with its registration in the system and ends with deregistration at the point of timeout of its lifetime. These three events indicate the behaviors of the agent: *(i)* query time event; *(ii)* maximum server active time event; and *(iii)* total lifetime event. In query time event, the agent obtains a list of sellers by sending a request to the yellow-page agent, and starts its negotiation process at regular intervals (e.g. every minute). (The negotiation process is described in Figure 3-7(b) and is conducted concurrently until no more sellers are involved within the deadline.) In a maximum server active time event, the agent starts its migration travel. Note that no agent causes buying agents to migrate; but buying agents autonomously mobilize themselves. After traveling over a list of sites in sequence, the agent will start a new round trip. The migration itinerary is updated from a UDDI agent once the agent migrates back to the home site. The agent ends its lifecycle when its lifetime is exhausted. Wherever the agent is, it should go back to the home server and terminate execution after sending out the notification to the personal agent. The migration process will be discussed in subsection 3.4.4.3.

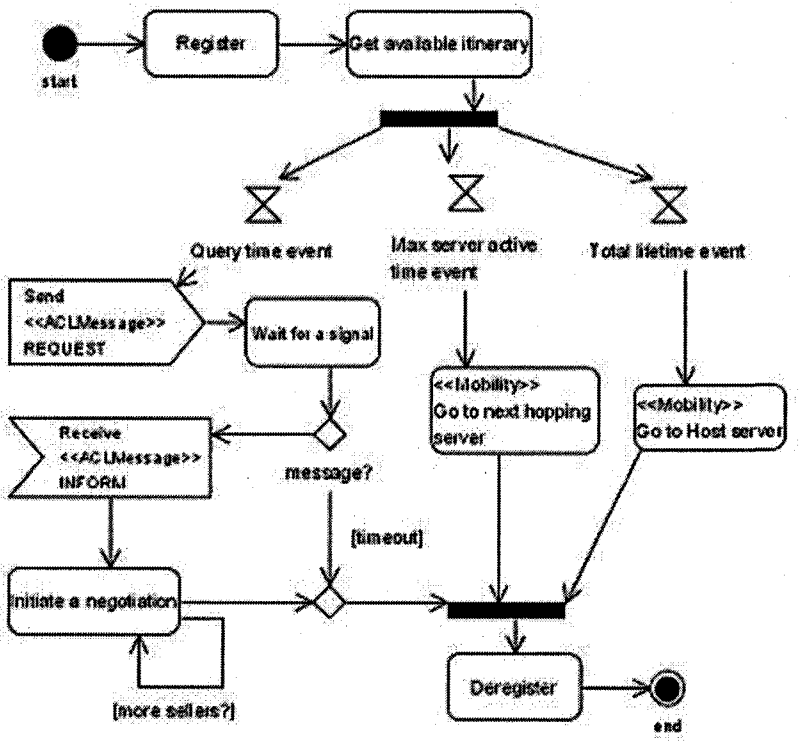


Figure 3-7(a): Activity Diagram of Mobile Buying Agents

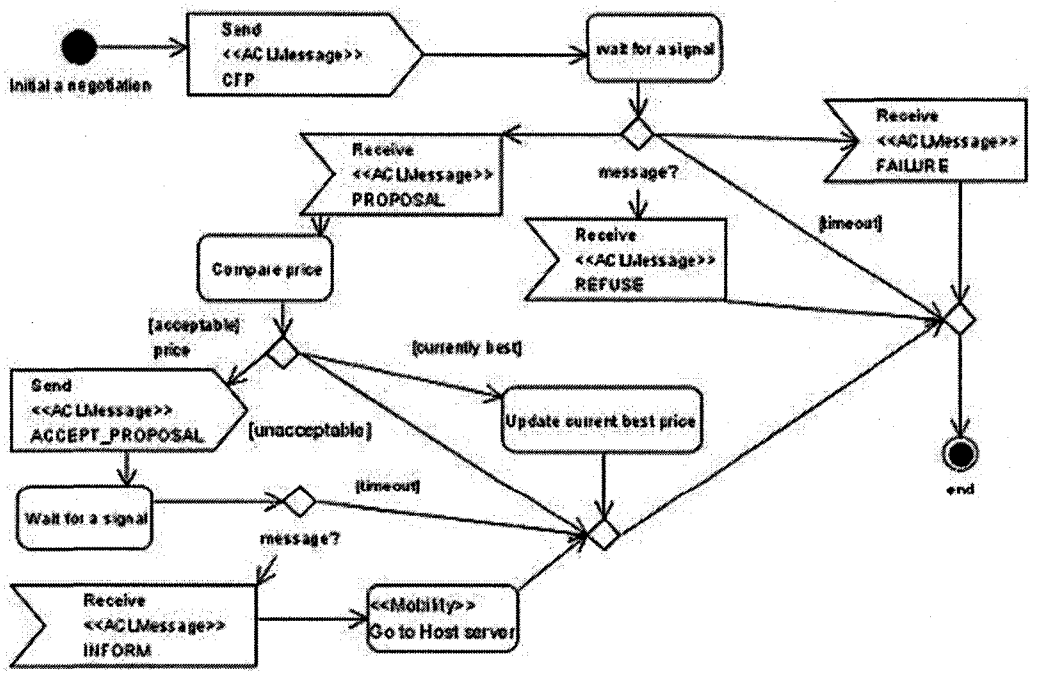


Figure 3-7(b): Activity Diagram of Mobile Buying Agents Negotiation

Figure 3-7(b) describes the negotiation process. This process is involved in buying agents and selling agents. Before entering the negotiation process, buying agents master a list of selling agents. The interaction between agents complies with the FIPA-Contract-Net Protocol [12]. This protocol allows the buying agent (initiator) to send a CFP to a set of selling agents (responders), evaluate their proposals, and then accept the preferred one (or even refuse all of them). Both initiators and responders should register in the system before they negotiate with each other. It is likely that buying agent will receive several proposals with different prices from the selling agents that are competing for the same buyer.

3.4.4.2 Behaviors of Selling Agents

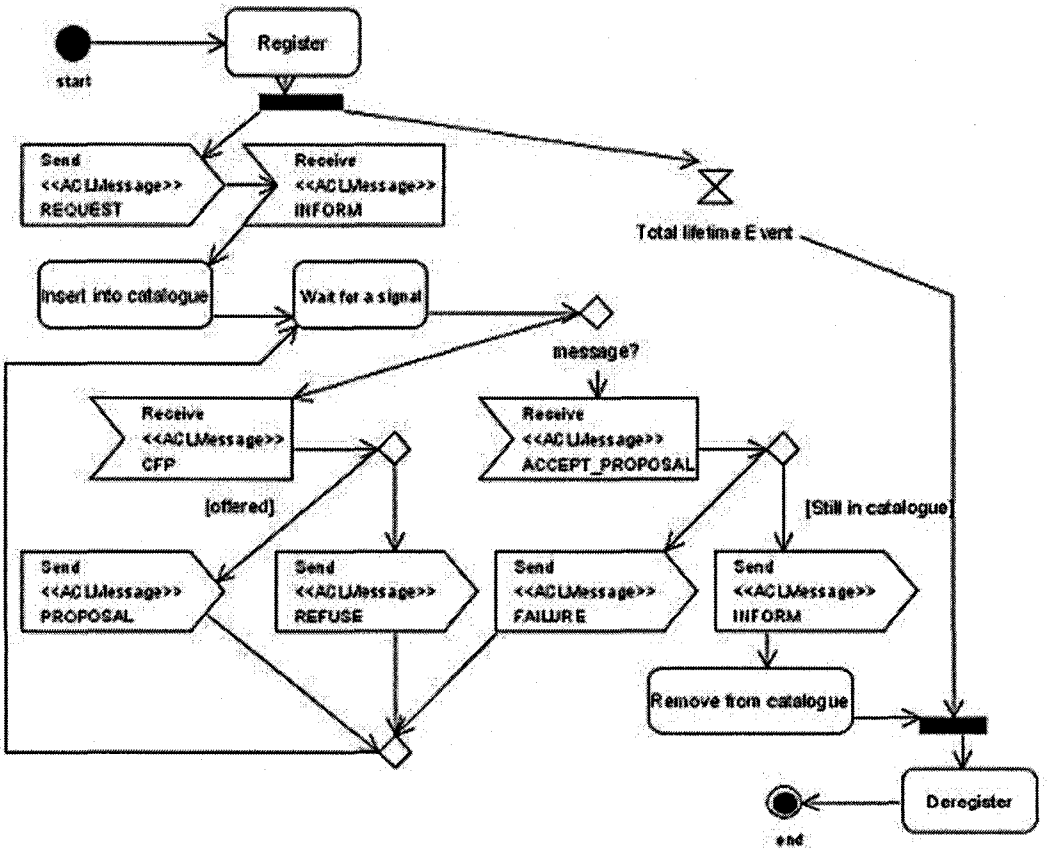


Figure 3-8: Activity Diagram of Selling Agents

Most behaviors of selling agents are similar to those of buying agents. At the beginning of a lifetime selling agents register in the system and deregister at the moment of end of lifetime.

As described in Figure 3-6, they have two events resulting in two processes: negotiation and termination. However, unlike buying agents, selling agents are not initiators but participators of the negotiation. Therefore the different negotiation behavior is described in Figure 3-8. Selling agents need to publish their products or services to ensure that it is possible for buying agents to discover them. Selling agents receive the CFPs from selling agents and then may refuse or suggest a proposal upon their product catalogue.

The selling agents and the buying agents work together to create a win-win relationship for both participants. While the buying agents collect information of preferred products and best prices, the selling agents sell the related products at sellers' will with maximum profits.

3.4.4.3 Migration Process

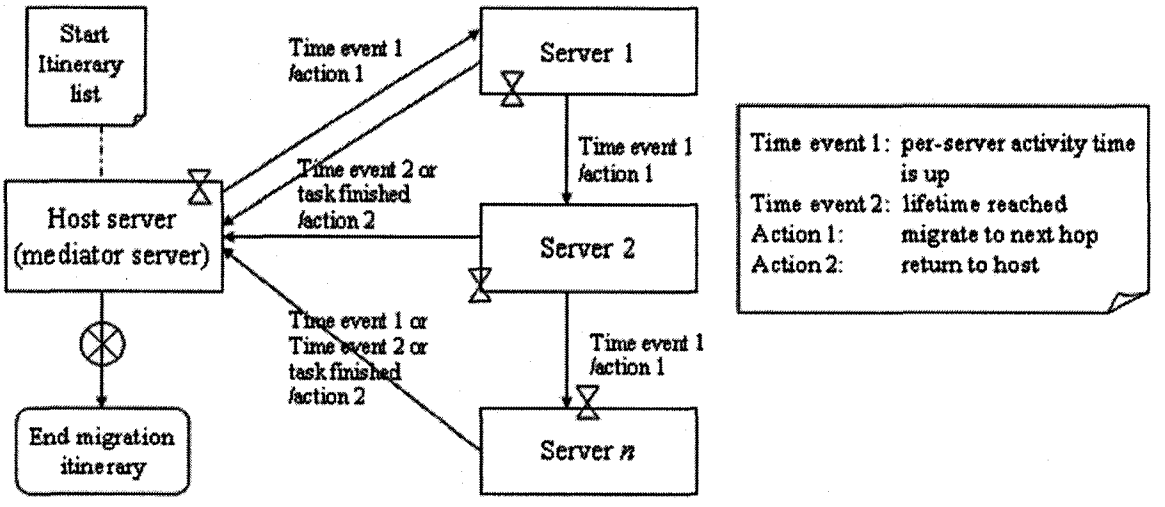


Figure 3-9: Agent Migration Process

The general process of migration is depicted in Figure 3-9. The buying agent starts its migration from its host server (i.e. the mediator server) with the migration itinerary acquired from the UDDI agent. (Assume that there are *n* servers, which will be visited by the agent in sequence.) In each server, two events happen resulting in two actions respectively: if the agent reaches its lifetime, it will return to its host wherever it was created, and then end the migration process; and if the agent exhausts its per-server activity time, it will migrate to the

next hop. Additionally, before the agent migrates to the next hop, it should also make the decision if it has fulfilled the task at the current server. As we know, the task is finished when the agent receives the acceptable offer from another agent. The migration process actually describes a scenario of price comparison (finding a price less than a buyer's reservation price for buying, or searching a price more than a seller's reservation price for selling). The agent may access its host server repeatedly during its lifetime and updates its itinerary list every time when visiting its host server.

3.5 System Interactions

In this framework, two types of interactions were identified: user-to-agent interaction and agent-to-agent interaction. This leads to the identification about communication medium and protocols within the architecture. Additionally, the ways of interaction between buyers and sellers in the real world can be inferred: SMS and E-Mail messages. SMS message enables communication between mobile phone users; and E-mail allows communication between worldwide email users, thereby expanding the world of communication for users.

3.5.1 User-to-Agent Interactions

In the wireless environment, users conduct wireless e-businesses via mobile devices. Since users speak human language, some kind of translation is required between user and agent. In this architecture, a human user directly interacts with the personal agent running on the mobile devices. The personal agent provides a GUI for user interaction and displays SMS messages on the GUI for user notification. The user configures personal preferences on the GUI specifying tasks. The GUI typically works on user's specific tasks so that the user indirectly interacts with other agents in the system. Therefore, the communication between user and the system is completed via the GUI of the personal agent.

3.5.2 Agent-to-Agent Interaction

It is noted that a personal agent is the only agent out of the multi-agent system in this architecture that is not managed. The main reason is that we take the scalability of the system into consideration. The issue of scalability will be discussed in Chapter 5. The communication between the personal agent and the proxy agent is done via the servlet. Figure 3-10 illustrates a basic wireless messaging scheme via a servlet between a personal agent and a proxy agent. The personal agent encodes a request and packages it in XML format. The servlet receives the request and forwards it to the personal agent. The proxy agent receives the request, decodes it, encodes it in ACL format and sends it to the multi-agent system. The proxy agent responds with XML-encoded messages to the servlet and the servlet forwards them back to the personal agent. The personal agent decodes the messages and presents them to the user. In this context, the proxy agent is a bilingual agent.

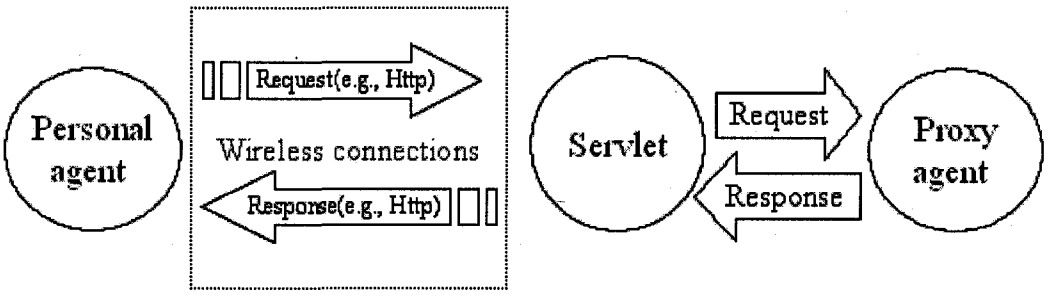


Figure 3-10: Wireless Messaging between Personal Agent and Proxy Agent

The interaction between the personal agent and the proxy agent is depicted in the part *a/1* of Figure 3-11. The interaction between the personal agent and buying or selling agents actually is one-way communication. As observed from Figure 3-11 also, the personal agent receives SMS messages from buying or selling agents regarding the results of tasks. Except that the communications between the personal agent and (*i*) the proxy agent and (*ii*) the buying or selling agent are in different format (XML or SMS), other agents in the multi-agent system communicate with each other in the form of FIPA ACL language. As described in Figure 3-11, the part *b/1* shows how the proxy agent forwards the request from the personal agent to the white-page agent. The part *c/1* and *c/2* explains how a buying agent and a selling agent are created. After born, both buying and selling agent register to the system via the white-page agent. The part *d/1* describes how a selling agent posts its products or services to the

yellow-page agent. The part *e/1* depicts the negotiation between the buying and selling agent. More details about the negotiation between the buying and selling agents will be explained in Section 3.5.3. After finishing delegated tasks, both the buying and selling agent deregister from the system via the white-page agent again.

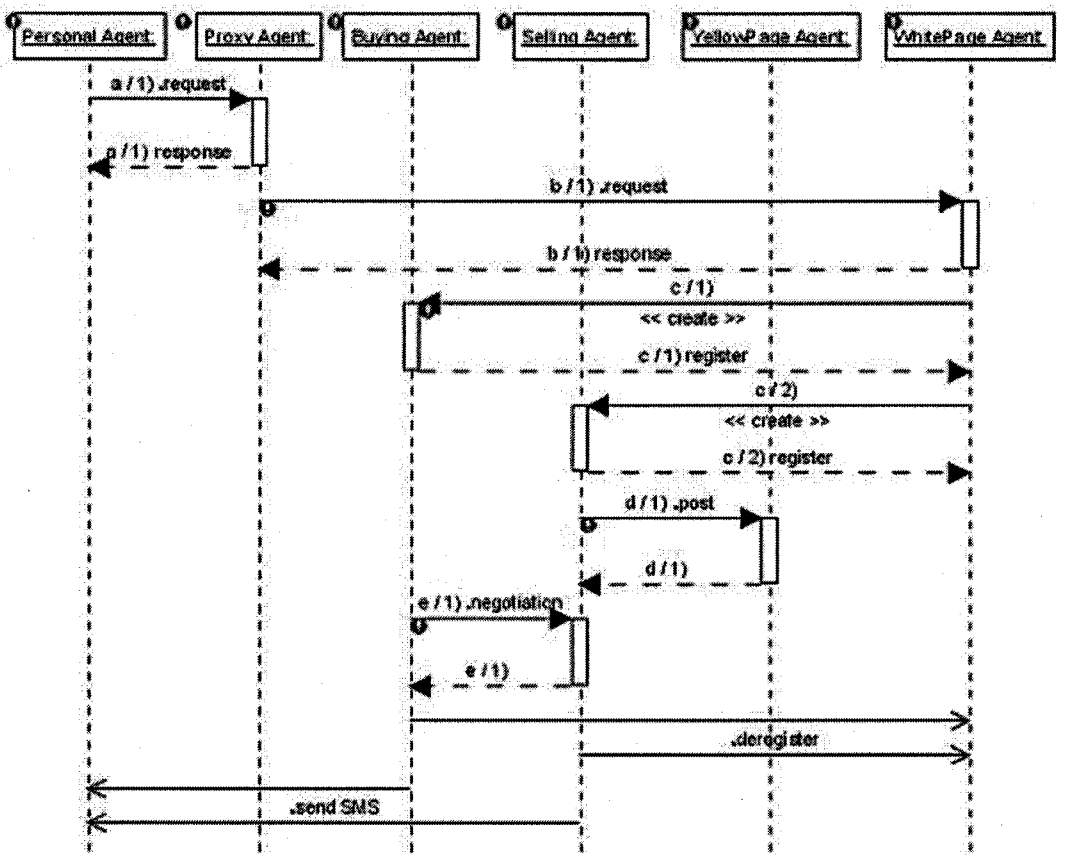


Figure 3-11: Sequence Diagram for Interactions of Agents

3.5.3 Communication Protocol in Negotiation

The negotiation is conceptualized as an e-commerce scenario that consists of requesting, matchmaking, negotiating and making decision. The infrastructure for negotiations is constructed through a number of agents: buying agent, selling agent and yellow-page agent. This thesis considers a classic situation that a selling agent offers a single item to the highest bidder (similar to eBay), and the simplest type of bid is an offer to buy or sell one unit at a specified price. As shown in Figure 3-12, the buying agent sends a CFP to all the available

selling agents (obtained from the yellow-page agent). After receiving the message, a selling agent can send the buying agent a proposal with the price for the product. If the product is not available or sold, it does not need to send any proposal. The buying agent will place a purchase order if the offer price is within the maximum price that the customer has specified. Results of price negotiations are sent back to the personal agent and showed in a graphical interface to the user. Since the system is fully asynchronous, an intention to make a purchase does not have to lead to a success. By the time the offer is made, other buying agents may have already purchased the last available item.

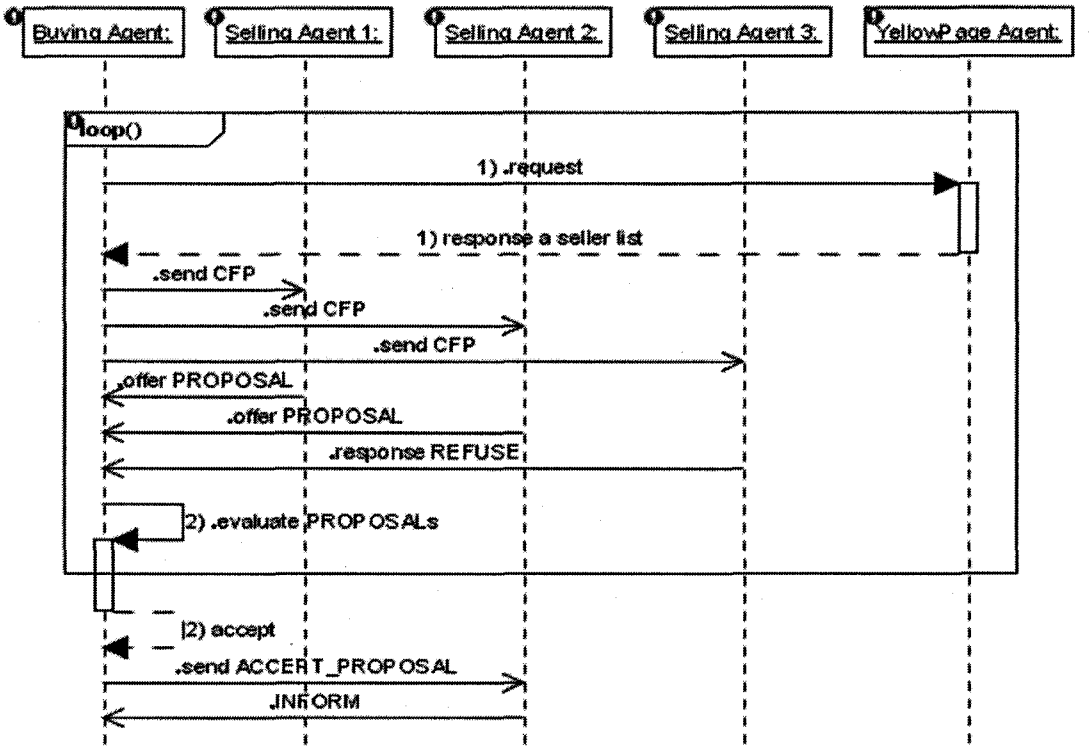


Figure 3-12: Sequence Diagram for Negotiation between Agents

3.6 Summary

The purpose of this chapter was to describe a feasible conceptual model to assist C2C e-commerce, using agent technologies. The agent aspects of intelligence and mobility were combined in the design. First, the objectives and some requirements of the system were listed.

An overview of the architecture revealed the functionality of the model. All roles and responsibilities of the agents were discussed and behaviors were refined. From the Object-oriented software engineering perspective, the UML diagrams are helpful to present the various interactions of the system components. Based on this work, the next chapter will present an implementation of the framework.

Chapter 4 System Implementation

This chapter will present the development of the proposed framework as follows. A simple prototype was implemented to evaluate the concepts proposed in the framework. Section 4.1 describes the topology of the environment. Section 4.2 introduces the used software tools. Section 4.3 explains the implementation details of agents. Section 4.4 presents the message standard used in the framework. How to construct the mediator server is illustrated in section 4.5. Finally, this chapter summarizes the concepts presented in this chapter.

4.1 Implementation Environment

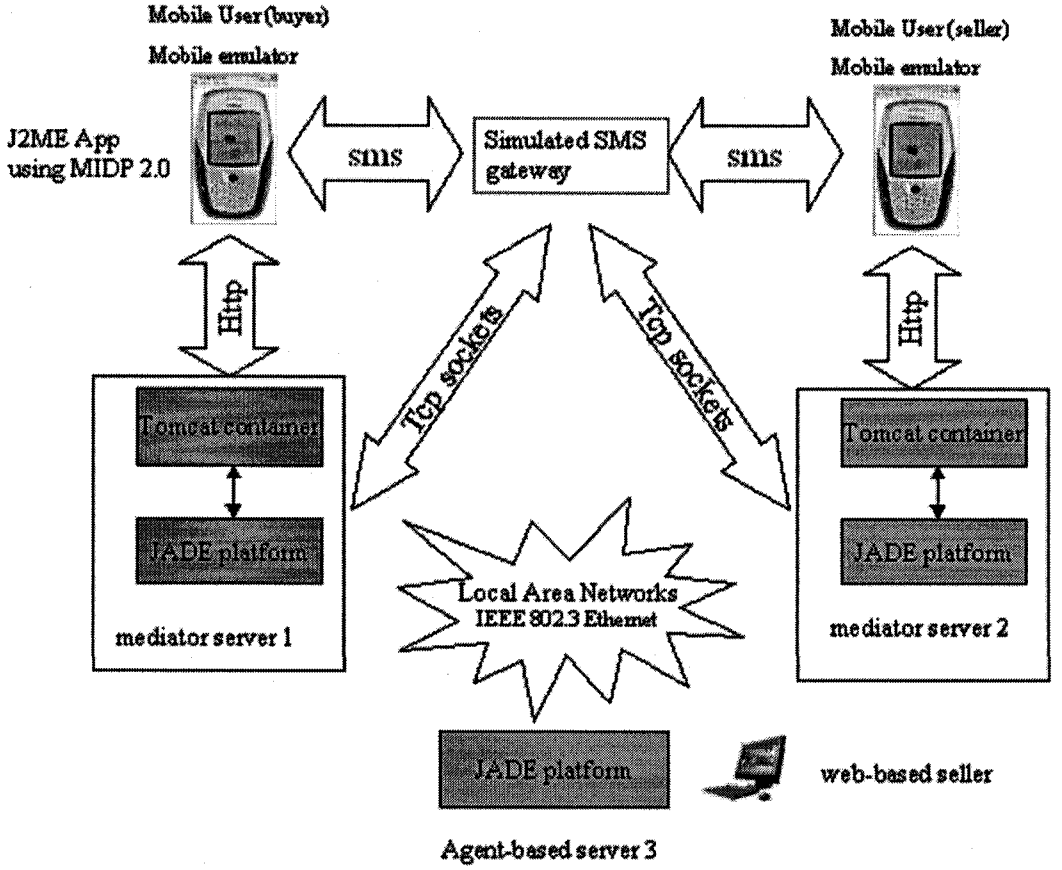


Figure 4-1: Topology of the Simulated Environment

Figure 4-1 depicts the topology of the elements that comprise the mediator-based framework. Three servers are deployed in the Local Area Networks (LAN, IEEE 802.3 Ethernet), two serve as mediator servers (run a Tomcat Apache Servlet Engine and a JADE platform) and the third one runs a JADE platform and serves as an online agent-based website. J2ME MIDlet[13] was also installed on two mobile phone simulators and a GUI is developed for the Web-based seller. So the development was divided into two main parts, mobile client and server applications. The client side provides User Interface support for mobile user to agent interaction and the server side offers services. The mobile client communicates with the server over HTTP protocol. The simulated SMS gateway was embedded in each mediator server for sending SMS to mobile phones over TCP sockets.

4.2 Software Tools

4.2.1 Programming Language

The object-oriented programming language, Java, was used for the programming and its latest version, Java 2 Standard Edition Development kit 1.5, was adopted. The advantages of using Java to develop are: (1) J2ME is the application platform for mobile devices; (2) in particular, Java supports working with XML and WSDL; (3) JADE is a FIPA-compliant agent platform developed in Java language; and (4) Apache Tomcat is the servlet container implemented in the specifications of Java Servlets and JavaServer Pages technologies. Thus, the framework was developed completely based on Java technologies.

4.2.2 JADE Middleware

JADE is one of the agent development tools that can support efficient deployment of both agents' mobility and intelligence in e-commerce applications. As a middleware for the development and run-time execution of peer-to-peer applications, JADE can seamlessly work and be interoperated both in wired and wireless environments based on the agent paradigm

[13] MIDlet is a Java program generally running on a cell phone, for embedded devices, more specifically the Java ME virtual machine [19].

[18]. JADE incorporates three basic management agents defined by FIPA: the Director Facilitator (DF), the Agent Management System (AMS), and the Remote Monitoring Agent (RMA) that manages a GUI of the JADE platform and all registered agents [18]. The RMA is not a part of the proposed framework. However, it provides a useful tool menu for us to interact with the DF and the AMS in the experiments, for example, view or modify the descriptions of registered agents. JADE also provides the service that supports migration between containers in the same platform. Integrating an add-on[14] into the platform supports the migration service between platforms. The platform consists of one Main-container and one or more non-Main containers. The reason for using the open-source JADE platform is that it has good scalability, one of the best modern agent environments compliant with FIPA. We used JADE to develop and deploy the multi-agent system in our framework. Tracy[15] is an alternative toolkit for developing multi-agent system, which major parts of Tracy are free software for non-commercial purposes supported by the University of Jena.

4.2.3 Apache Tomcat Servlet Engine

The Tomcat 5.0 servlet container was used to deploy the Web Services on a network. Apache Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlets and JavaServer Pages technologies. A servlet container is a runtime shell that manages and invokes servlets on behalf of users. The Java Servlets and JavaServer Pages specifications are developed by Sun under the Java Community Process. Java Servlets technology defines HTTP-specific servlet classes. Java Web Services Developer Pack (JWSDP) 2.0 is a tool kit providing APIs for core Java Web Services and the XML processing. The Web services architecture communications are based on JSR172[16], J2ME Web services, which includes two independent parts: the JAX-RPC and JAXP. J2ME JAX-RPC APIs subset solves how to access the SOAP/XML Web services and JAXP APIs subset solves how to processes the XML messages. XML is chosen as the standard way for clients to interact with backend servers so as to use the remote services. We used Tomcat to develop

[14] This add-on is a third-party software accepted by the JADE community [18].

[15] Tracy is available on http://wiki.tracy.informatik.uni-jena.de/mobileagents/tiki-index.php.

and deploy our Web services server in the framework. JWSDP was used to develop the servlet in the framework. Other than Tomcat, some Java application servers on the market can be also used in the development and deployment of the Web services server, for example, commercial WebLogic Server produced by BEA Systems [17] and open-source JBoss Application Server supported by JBoss community[18].

4.2.4 J2ME Wireless Toolkit

The Java 2 Platform, Micro Edition, Wireless Toolkit 2.2 [17] supports the development of Java applications that run on devices complaint with the Mobile Information Device Profile (MIDP) 2.0. The Wireless Toolkit also provides Wireless Messaging API (WMA) Bridge APIs for connecting J2SE clients to the toolkit's messaging environment. WMA specification supports SMS that allows us to simulate SMS gateway, peer-to-peer messaging. KtoolBar, which is included in the Wireless Toolkit, is a simple development environment for compiling, packaging and executing MIDP applications. Use of an additional IDE, Eclipse[19] is useful for editing and debugging of Java source files. A selection of mobile devices emulator is also provided for executing and testing of applications. The emulated devices have the capability of emulating the features in the CLDC, MIDP, and WMA specifications. The reason for choosing the Wireless Toolkit was primarily for the development of J2ME applications. J2ME has the following benefits: (*i*) it provides connectivity for applications with distributed system, talking in XML that is suitable for cross-platform; (*ii*) it offers user personalization with better manipulation; (*iii*) it allows an application to use multithreading; and (*iv*) it has low network usage and can operate when disconnected. We used J2ME wireless toolkit to develop the personal agent on the emulator that simulates a mobile device (e.g. a cell phone).

---

[16] JSR172: J2ME Web Services Specification 000172, to define an optional package providing standard access from J2ME to web services [19].

[17] WebLogic is available on BEA's web site, http://www.bea.com/.

[18] JBoss Application Server is available on http://labs.jboss.com/jbossas/.

[19] Eclipse: an open extensible development platform for Java applications (www.eclipse.org).

4.3 Implementation of Software Agents

This framework is composed of different kinds of agents: the personal agent, the proxy agent, the buying agent, the selling agent, the UDDI agent, and the AMS and DF agent. Each mobile phone runs a personal agent. Only one proxy agent is implemented in the system, and only one AMS and one DF agent exist in the system. The reason is that the principle of the design thought is to limit the number of the agents to be as minimal as possible, resulting in an increase in the scalability. Except the personal agent, other agents are JADE agents performing tasks that are modeled and implemented as behavior objects [18]. Each JADE agent adds these behaviors into the list of tasks and executes them concurrently. These simple behaviors are categorized into one-shot and cyclic behaviors. The AMS agent and the DF agent are provided by the JADE toolkit and we do not need to reinvent the wheel. We implemented the personal agent, the proxy agent, the buying agent and the selling agent from scratch. We did not implement the UDDI agent in the system because it cannot interact with the UDDI Registry Server via the gateway on the Internet under the experimental circumstance. We focused on the migration itinerary managed by the mediator server and we manually configured it as a text file in the experiments. Therefore, in the experiments, we assumed that our mediator server is known by other servers in the LAN and it knows other servers as well. The text file includes these servers' name and IP addresses. At the beginning of execution, a mobile agent loads this text file from its local mediator and then establishes an itinerary for its migration.

Figure 4-2 shows us a picture of agents living in the containers of the JADE platform. The main-container held the DF, AMS, RMA agents. The proxy agent was running in the container-1. The buying and selling agent was either in the main-container or in other non-main containers depending on the workload of the system. To balance the workload of the system, a simple algorithm was developed to distribute the mobile agents in the different containers. That is, each container holds a maximum of 10 agents at the same time. More containers will be created to hold the coming agents. The following sections present the implementation details for the agents (the AMS and DF are the default agents of JADE platform).

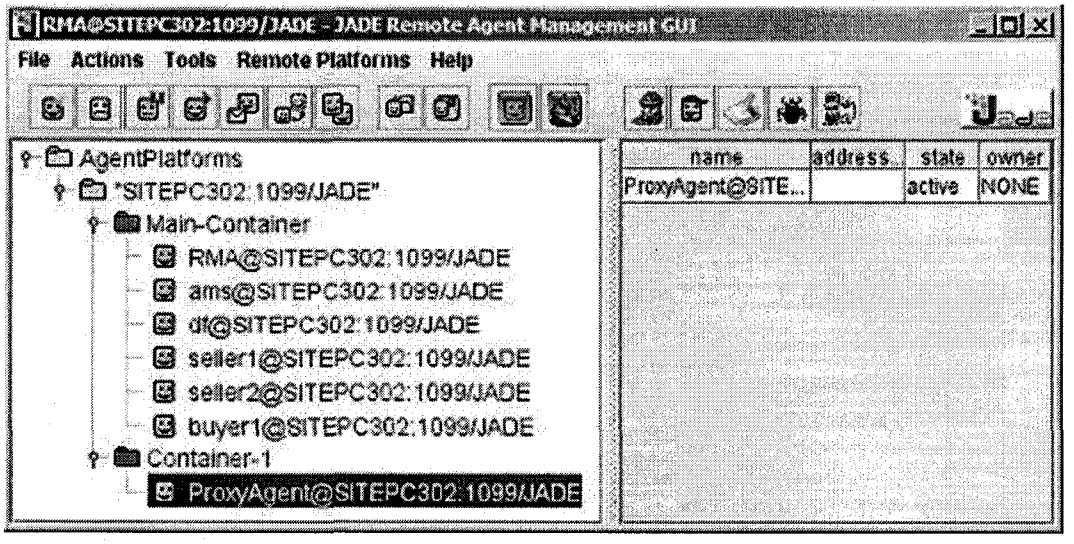


Figure 4-2: Agents in the JADE Platform

4.3.1 Personal Agent

The personal agent is developed as a J2ME MIDlet application that offers the GUI interfaces for users to initiate or recall mobile agents, and dialogue with the mediator server. It is modeled and implemented based on the design pattern, Model-View-Controller (MVC) [17], which consists of three parts: model, view and controller. In MVC, the user interface (called the view) registers itself as a listener to certain parts of the application's underlying business and functional logic, as represented by the model. The model then notifies all registered views whenever there is a change in the data. Completing the cycle, the controller receives user actions and dispatches them to the model. Using MVC pattern improves the readability and robustness. This MIDlet communicates with a Java servlet in the server via HTTP, and over a secure channel (HTTPS) when necessary. From the user's perspective, Figure 4-3 depicts the tasks the personal agent can perform:

(1) Creates a new agent: The user will be asked to create an agent profile with different agent type, buyer or seller. The user needs to configure user id with the telephone number, server URL, preferred item description and price, maximum per-server active time, and total lifetime. Optionally, the user can specify the mobility that decides whether the buying agent or selling agent travel among the servers or not.

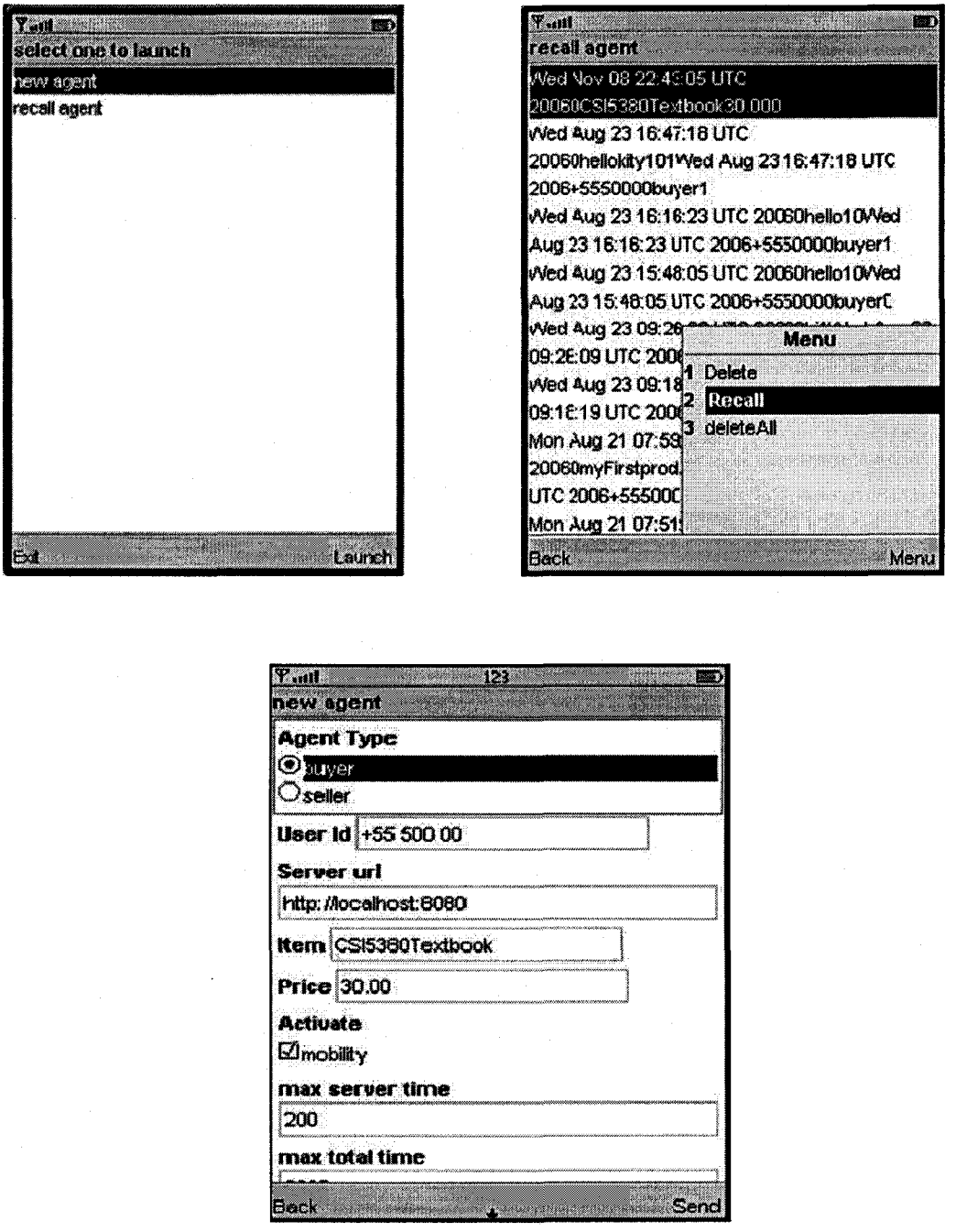


Figure 4-3: Screenshots of the GUI of Personal Agent

(2) Recalls an existing agent: After launching a buying or selling agent, the user may recall the agent to terminate the task or query the current state of the agent. The agent information is cached in the persistent storage of the mobile devices.

All these functions are implemented as separate modules in the personal agent. From the view of software engineering, they are reusable. Consequently, the user can repeat the

execution of functions at any desired time. Each task is performed under the circumstance of connecting the mobile phone to the server. All tasks, however, are only one round trip between the personal agent and mediator server. One of most important benefits is that it reduces the connection time. Personalization of information presented to the user by the personal agent on a mobile device allows complete user control. This leads to provide convenience for the user. In the MIDP, personal preference settings and mobile agent information were stored in persistent storage. Therefore they can be dynamically changed over time. It should be noted that the GUI classes included in the MIDP are not based on the Abstract Window Toolkit (AWT) which is designed for desktop computers. The central abstraction of the MIDP UI is a screen object that encapsulates device-specific graphics rendering user input, including components such as List and TextFields within a Form. The user can traverse through these items on the screen. Displaying SMS on the screen is not a part of our GUI programming but a functionality that is supported by the emulator of mobile devices.

4.3.2 Proxy Agent

The proxy agent is a JADE agent, linking the JADE platform and the servlet. It is synchronously created in the non-main container of JADE platform at the first initialization of the servlet. The purpose of that is to ensure that a proxy agent is associated to the servlets once and only once. Finally, the proxy agent uses the synchronizer to notify the servlet that it is ready to receive messages.

The proxy agent can communicate with personal agents, DF agent and the AMS agent. A personal agent contacts the proxy agent via a servlet when it needs to query the state of an agent, to create a new buying or selling agent, and to delete an existing buying or selling agent. The AMS agent passively contacts the proxy agent when it receives the request from the proxy agent regarding creating or deleting an agent. The DF agent also passively contacts the proxy agent when it returns the agent state upon query or it processes the registration of the proxy agent.

Table 4.1 Proxy Agent Behaviors

Request Received	**Response Action**
New an Agent	Requests AMS to create a new agent according to the information obtained from the servlet, and returns a result in a NewAgentResponse message.
Kill an Agent	Requests AMS to terminate a possibly existing agent according to the agent AID obtained from the servlet, and returns a result in a KillAgentResponse message.
Get Agent State	Obtains information of an agent with a given AID from AMS, and returns the information in an AgentStateResponse message.

The messages exchanged between them have a format specified by the ACL language defined by the FIPA for agent interoperability (expressed by JADE Management Ontology, FIPA Interaction Protocol and FIPA Semantic Language). The proxy agent has a cyclic behavior to accept all requests from the servlets and its other actions are defined by the one-shot behaviors as described in Table 4.1. In response, messages returned to the servlets are encoded in XML format.

4.3.3 Buying and Selling Agents

Both buying and selling agents are also JADE agents. They are identical agents created by the AMS agent upon the request from the proxy agent. They can communicate with the personal agent in one-way, that is, sending the SMS to the personal agent but it is not the case vice versa. The buying agent contacts the DF agent when it needs to search a list of active related sellers in the same platform. Similarly, the selling agent contacts the DF agent when it needs to post its products or services in the same platform. Buying agents attain the migration itinerary from the server once initialized. Their behaviors are autonomous and asynchronous.

Table 4.2 Behaviors of Buying Agents

Class Name: TickerBehavior	
Description: This behavior implements a cyclic task that must be executed periodically. It will be invoked at particular times.	
Given Particular Time	**Action**
Query sellers time	Searches the relative-service sellers from the DF, and updates the list of sellers.
Per-server activity time	Migrates to the next hopping server.
Total lifetime	Migrates back to the host server, sends a SMS to the personal agent, and terminates itself execution.
Class Name: RequestPerformerBehavior	
Description: This behavior is a multi-step task that implements a contract-net protocol.	
Message Sent/Received	**Response Action**
1. a CFP sent	Sends the CFP to all relevant selling agents.
2. PROPOSE received	Receives all proposals/refusals from all relevant selling agents, evaluates the best price offered and updates the present best offer.
3.ACCEPT_PROPOSAL sent	Sends the purchase order to the selling agent that provided the best offer.
4. INFORM received	Receives the purchase order reply, and sends a SMS to the personal agent.

Table 4.3 Behaviors of Selling Agents

<table>
<tr><td colspan="2">Class Name: TickerBehavior</td></tr>
<tr><td colspan="2">Description: This behavior implements a cyclic task that must be executed periodically. It will be invoked at particular time (i.e. lifetime) and sends a SMS to the personal agent, and terminates itself.</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="2">Class Name: OfferRequestBehavior</td></tr>
<tr><td colspan="2">Description: This behavior is a cyclic task to serve incoming offer requests from buying agents.</td></tr>
<tr><td>Message Received</td><td>Response Action</td></tr>
<tr><td>CFP</td><td>(1) If the requested product or service is in the local catalogue, replies with a PROPOSE message specifying the price.
(2) Otherwise, replies with a REFUSE message specifying not-available.</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="2">Class Name: PurchaseOrderBehavior</td></tr>
<tr><td colspan="2">Description: This behavior is a cyclic task to serve incoming offer acceptances from buying agents</td></tr>
<tr><td>Message Received</td><td>Response Action</td></tr>
<tr><td>ACCEPT_PROPOSAL</td><td>(1) If the relevant product or service is not sold to another buying agent yet, removes the relevant product or service from the catalogue, replies with an INFORM message to notify the buyer that the purchase has been successfully completed.
(2) Otherwise, replies with a FAILURE message specifying not-available.</td></tr>
</table>

The different actions inside buying agents and selling agents are shown in Table 4.2 and Table 4.3 respectively. With regard to the migration problem in the implementation, in the context of programming, a mobile agent was represented in a Java Archive (JAR), which contains the serialized state of the agent. Then the mobile agent is transported by means of JAR on demands.

4.3.4 Third-party Selling Agent

A GUI was also developed, as shown in Figure 4-4, for a third-party seller to launch a selling agent. This agent can represent a seller of a company conducting online B2C business.

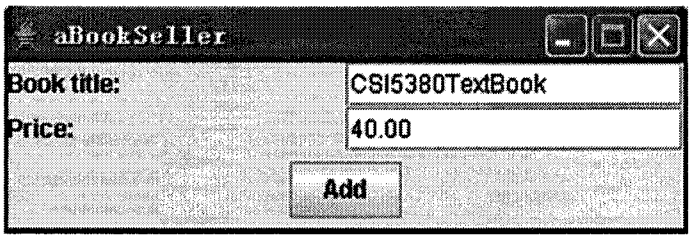


Figure 4-4: GUI of the Third-party Selling Agent

4.4 Message Standards

To improve the interoperability of the framework, the messages transmitted between the personal agent and the server are based on XML format, and the messages exchanged between JADE agents are encoded in the ACL message format.

4.4.1 XML Message

The communication between the personal agent and the proxy agent are encoded in XML format. XML provides a system-independent format for specifying the information to be exchanged. This design is to ensure that mobile phones can synchronize their data with the server where information is stored. This makes the data portable and improves the interoperability and flexibility of the personal agent. As described in Figure 4-5, XML document defined the structured data of a mobile buying agent. A set of data type definition

(DTD), such as agent type, item description and lifetime, is defined and known to the server. The server retrieved data from the XML document by parsing it based on its meaning and structure. Therefore, both the personal agent and the proxy agent can agree upon what format of messages will be used and be able to handle them respectively. In the implementation, Simple API for XML (SAX), an event-based XML parser, was used because it is an industry standard and consumes less CPU and memory.

```
<? xml version="1.0" ?>
<configuration>
    <agenttype>buyer</agenttype>
    <serverURL>http://localhost:8080</serverURL>
    <itemdescription>csi5389textbook</itemdescription>
    <preferedprice>50.00</preferredprice>
    <mobility>yes</mobility>
    <serveractivetime>100</serveractivetime>
    <lifetime>850</lifetime>
</configuration>
```

Figure 4-5: XML Document of a Buying Agent

4.4.2 ACL Message Delivery between Agents

The ability to communicate is one of the most important features of agents. In order to promote interoperability between agent platforms, FIPA ACL is used as a standard language to define the messages delivered between JADE agents. The asynchronous message passing as described in Figure 4-6 was adopted as the communication paradigm.

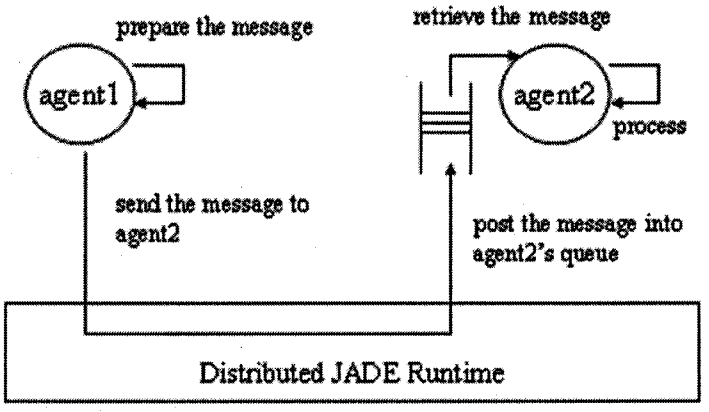
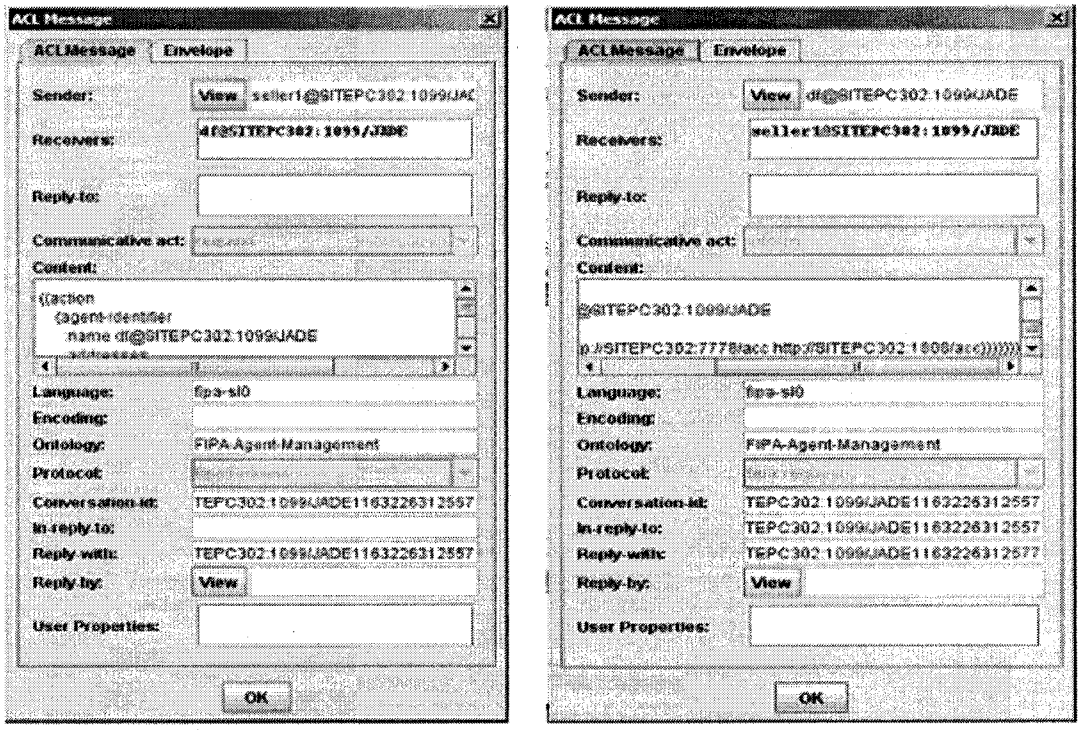


Figure 4-6: JADE Asynchronous Message Passing Paradigm [18]



(a) ACL message description when a selling agent tried to post products via DF agent

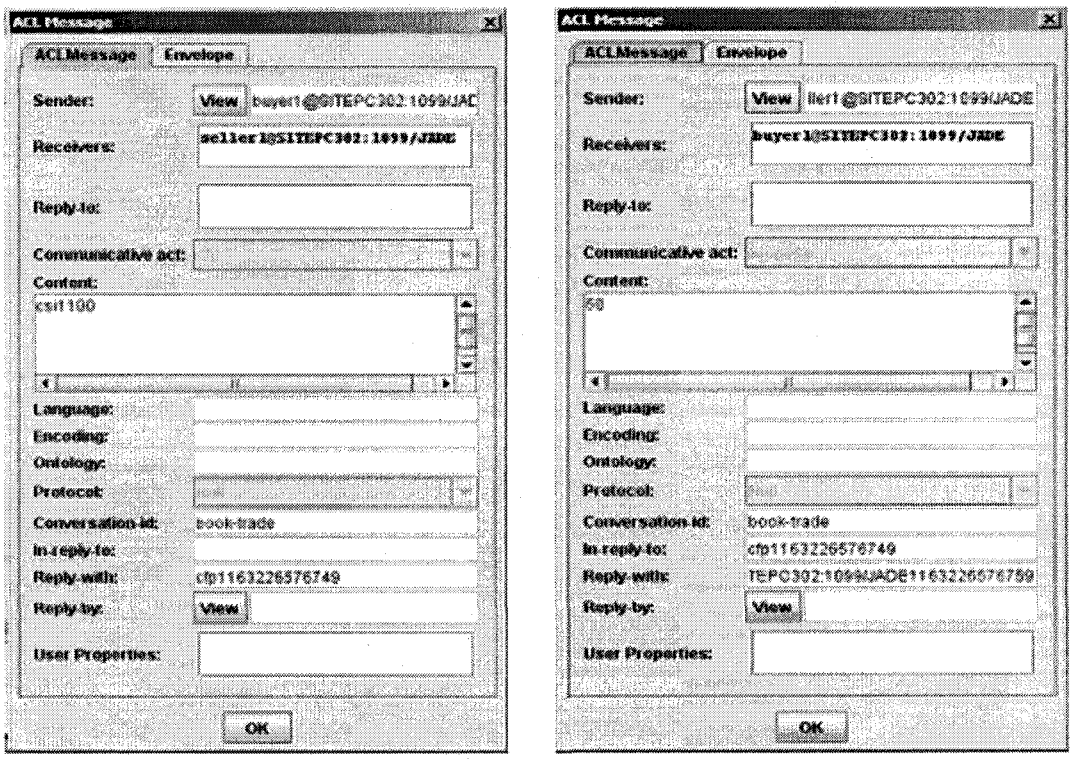
(b) ACL message description when DF agent accepted the registration of seller agent

Figure 4-7(1): ACL Messages Between DF and Selling Agents

ACL messages exchanged by JADE agents have a format as shown in Figure 4-7 (1) and (2). This format comprises a number of fields and in particular: the sender of the message; the list of receivers; the content (i.e. the actual information included); the language (i.e., the syntax used to express the content); the ontology (i.e., the vocabulary of the symbols encoded in the

content). Figure 4-7(1) describes the messages exchanged between the DF and a selling agent when the selling agent tried to register to the DF and post its products. The language FIPA S10 [10] was used in this case. The selling agent is able to encode or parse expressions complying with this syntax.

Figure 4-7(2) describes the messages exchanged between a buying agent and a selling agent when they were engaged in the negotiation. In this situation, the ontology of book trading was defined. The communicative act CFP expressed an intension that buying agent sent to the selling agent regarding a book request. The content of the message was the title of the book. The PROPOSE message carried the seller offers, e.g., the price of the book.



(a) ACL message description when a buying agent sent a CFP to seller agent

(b) ACL message description when a selling agent replied to buying agent with a proposal

Figure 4-7(2): ACL Messages Between Buying and Selling Agents

4.5 Mediator Server

The mediator server plays an important role in the framework, containing a Web services server and a multi-agent system.

4.5.1 Web Services Server

The Web server provides connection services for mobile phones. It responds to the HTTP requests from mobile phones and forwards the requests to the multi-agent system. The servlet technology was applied to provide a simple mechanism for the functionality of the Web server. Tomcat Servlet Engine was used to set up the Web server. Each request from the personal agent was associated with a servlet. Then the servlet forwarded the request to the proxy agent. The function of Web server was not provided too much because it was not the most important aspect of the implementation. In other words, the extra unnecessary overhead of the whole server was avoided.

4.5.2 Multi-agent System

The JADE middleware was used to develop the multi-agent system. Like threads scheduling and messaging support, JADE agents share resources and services in a run-time environment where they are executed. One or more containers provide the run-time environment, which consists of a JADE platform and hosted by the Java Virtual Machine (JVM) [17]. Each agent has one logical reference, Global Unique Identifier (GUID) [18] which is used in the ACL message and then resolved into a physical address by the platform. To speed up the agents' execution, "Just-in-Time" compilation that is provided by the JVM was used, and then most used parts of agent codes were compiled at run-time.

4.6 Scenario for a C2C E-commerce Application

The proposed system is simulated through a simple scenario. In the university community, students sell textbooks and other products to other students. They might advertise that they are subletting apartments (services) as well. More often than not, the neighborhood nearby

conducts garage sales. The University deploys several mediator servers in the university center and residence areas. In this case, both products and services are accessible by students (users) through their mobile phones connecting to the servers. It is also assumed that the servers have proper agreements with the wireless carriers.

4.7 Lessons Learned

This section summarizes our experiences when implementing our system, and aims to capture some practical lessons for future design and development.

4.7.1 Choosing an Appropriate Modeling Language

In our work, we modeled a multi-agent system at the agent level with an agent-oriented extension to UML and implemented it using the Java programming language. Role modeling was used to structure the collaborative capabilities of the agents. Especially, we used UML Activity Diagrams to model the dynamic behavior of mobile agents. However, the complexity of today's agent-based applications is increasing. Agent-specific concerns, such as autonomy, mobility, roles, learning, and error handling, have been involved in the software engineering of large MASs. The agent properties should be designed separately from the agent's basic behaviors because they are typically overlapping and crosscut the agent's basic functionality [16]. According to our knowledge, agent-oriented modeling languages and OO programming languages provide no adequate abstractions for separation of crosscutting concerns in the modeling and implementation levels. We discovered that developing new agents required extensive re-development of the existing system. As new agents were added, new functionalities and interfaces needed to be added to existing agents. When we chose an agent-oriented approach, we expected to use a small set of agents for the basic functionality of the system. As client requirements or technology capabilities evolved, specialized agents could be added or replaced with different versions.

4.7.2 Choosing Appropriate Tools and Technologies

We chose open source software, such as Apache Tomcat and JADE, as the tools of development and deployment of the system. The open source license allows users the freedom to run the program for any purpose, to study and modify the program, and to freely redistribute copies of the original or modified program. Also, open source forums are widely used in the community to ask questions and share answers. In our implementation, the JADE platform was used to support inter-agent collaborations and agent mobility. With the JADE middleware, we do not need to implement the multi-agent system from scratch. Therefore, we focused our development on the market mechanism rather than on the underlying communication infrastructure. Our system needs to handle a large number of clients, but JADE does not address scalability in any meaningful way. Finding an effective approach for scalability is extremely challenging. We therefore decomposed the platform between multiple containers in the several servers. This structure allowed us to decompose the message delivery tasks between containers and balance the workload of the whole system. Our hope for scalability depends on the enforced agent technology that is developed with more scalable mechanisms. In our implementation, we also ran into a number of bugs (especially when we used the third-party plug-ins). We found that errors could propagate from agent to agent through communication channels, making it difficult to identify the agent at fault. We finally fixed the problems due to contributions from the open source forums which is one of the benefits of open source.

4.8 Summary

This Chapter presented the implementation environments and tools integrated into the development of the mediator-based framework. The implementation used standard Web services technologies and Java technologies to show an approach with high feasibility. First, this chapter described the architecture topology, and then all kinds of software agents integrated in the framework were presented. Then the message standards and the construction of the server were illustrated. The next chapter will outline and discuss the evaluation criteria resulting from the issues listed in this thesis.

Chapter 5 Experimental Evaluation

5.1 Introduction

This chapter describes our experimental environment, our evaluation metrics and our results. Our main objectives are to present some criteria intended to validate the concept of the proposed framework and to evaluate the implementation through some experimental results. We will observe and measure the functionality of personal agents and mobile agents since personal agents represent the desire of users and mobile agents offer the delegation of tasks. The migration of agents will be examined to evaluate the mobility, which is an important aspect of our framework. We will look at communication languages used among agents to evaluate the interoperability and the extensibility. Also, we examine the scalability to evaluate the feasibility of our framework. Through these evaluation criteria, we verify the benefits for mobile device users, which are to meet the challenges discussed in earlier sections.

Our evaluation methodology is as follows. In the experiments, each agent has the ability of being tracked. Each agent recorded its lifecycle information in a log file, from the start of execution to termination. The log files store some necessary information, including timestamps, agent IDs, migration time, messages sent or received, and so forth. We can examine each agent's lifecycle using these log files. Each server also kept track of its agents' activities in some files, including the migration and negotiation information. These files provided effective information for our examining and analyzing the results of the experiments, for example, what price was offered to which buying agent by which selling agent, or into which server and at what time a buying agent migrated.

5.2 Experiments

Some experiments have been performed according to the scenario described in Section 4.6. Several measurements were taken in the experiments. Because of some restrictions in the availability of network nodes, we limited the number of servers that will be visited by the mobile agent. In particular, we observed some specific experiments related to the requirements in the design.

5.2.1 Hardware and Software Configuration

Only three IBM personal computers were used to simulate three servers as distributed in the topology of Figure 4-1. The three hosts were connected within a 100Mbit/s Ethernet LAN, showing the same characteristics that are described in Table 5.1. In the experiments, we distinguished two roles for the computers involved: the host server and the guest server. The host server is the computer on which the agent is started and the other computers visited by this agent are called the guest servers.

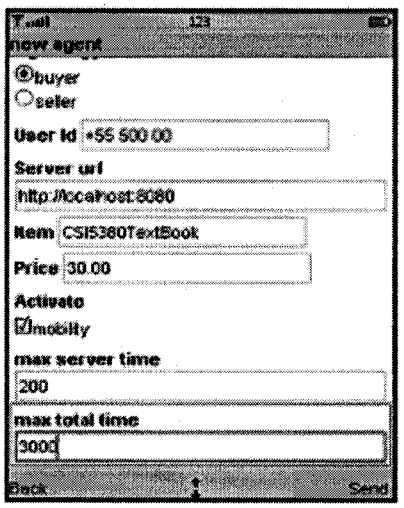
Table 5.1 Hardware and Software Information of the Three Servers

Item	**Description**
Operating System	Microsoft windows XP professional version 2002 service pack 2
Processor	Intel® Pentium® 4 CPU 3.40 GHz
Total Memory	3.39 GHz, 0.99 GB of RAM
Network Adapter	Broadcom NetXtreme Gigabit Ethernet
Java	Sun JDK 1.5

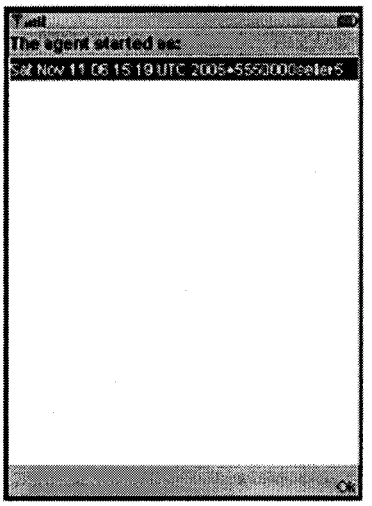
5.2.2 A Test Scenario

It is unnecessary to elaborate on each experiment that was conducted. Out of them, we are solely interested in elaborating a specific and typical scenario, which helps students search the ideal used textbooks (e.g., the best price) in the campus. In this scenario, a student (buyer)

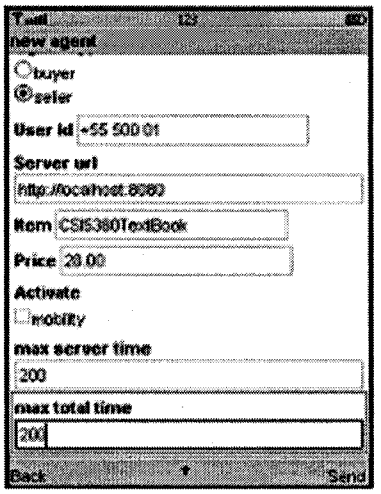
wanted to buy a used textbook, say CSI5389, and then launched a buying agent on the mediator server 1 via the emulator called MPE0. Another student (seller) desired to sell a used textbook, say CSI5389, and then started a selling agent on the mediator server 2 also via the emulator called MPE1.



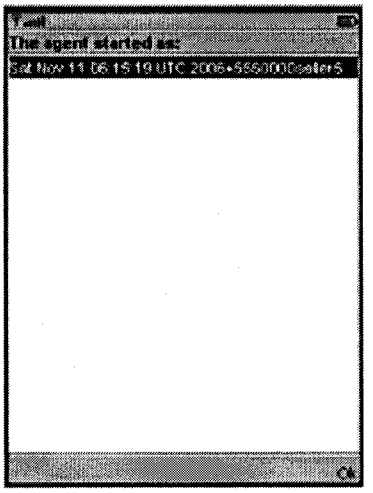
(a) MPE0 was ready to send a request of creating a buying agent



(b) MPE0 received message that buying agent was created



(c) MPE1 was ready to send a request of creating a selling agent



(d) MPE1 received message that selling agent was created

Figure 5-1(a)-(d): The Creation Procedure of Buying Agent and Selling Agent

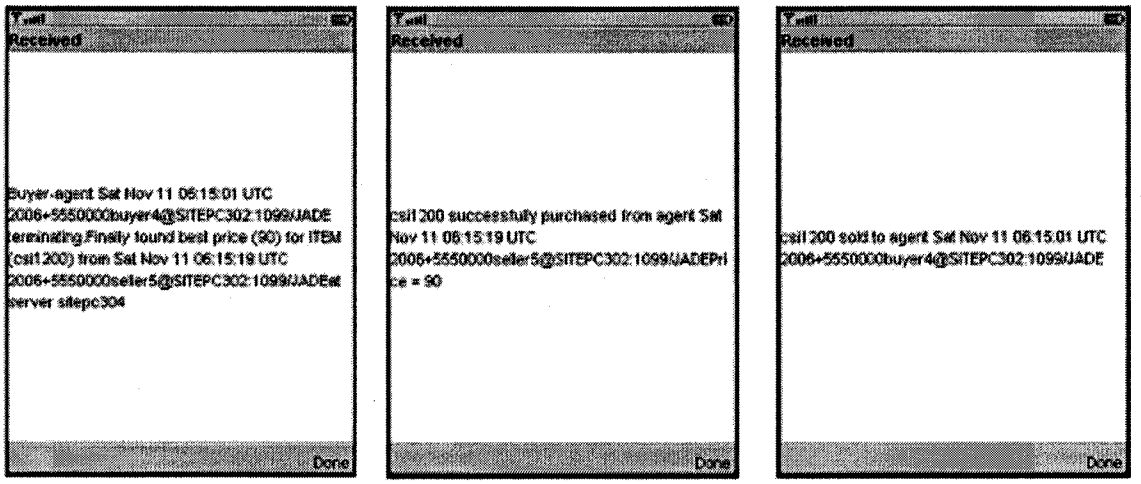
The two students were identified with their phone number, +5550000 and +5550001 respectively. The desired textbook cannot be found in the host server, but is available at a

guest server. In the buyer's emulator, the mobility of buying agent was activated. For the sake of tracking, the mobility of the selling agent was not activated in the seller's emulator. As depicted in Figure 5-1 (a-b), the creation of the buying agent was a synchronous procedure. It means that the personal agent was unavailable to operate until the request was processed in the server. Similarly, the creation procedure of the selling agent is described in Figure 5-1(c-d). At any time, students may add items via their personal agents and specify their preferences such as a time limit and a preferred price for the trading, e.g., more desired textbooks. Also, students may check the state of mobile agents or change their mind to cancel the task.

5.2.3 Measurements and Observations

First, we examined the functionality of the personal agent and observed the following results: Users, with the assistance of their personal agents, can connect to the mediator servers via HTTP and initiate mobile buying or selling agents in the mediator servers. They do not need to instruct their buying or selling agents of what to do after configuring their preferences. Also, users can add new items anytime and relevant mobile agents will be created to handle the trading of these new items respectively. In addition, users can send instructions to the servers to kill their buying or selling agents in order to cancel their tasks.

Second, we examined how a buying or selling agent worked well as a peer agent of the personal agent. Both buyers and sellers should be informed by SMS messages from their personal agents, regarding the state of their buying or selling agents. In fact, SMS messages reflect the important milestones during the lifecycle of buying or selling agents. Buying agents report the results to their buyers from time to time. As we can see from Figure 5-2(a), the buyer received a SMS message that the best price (90) was found for item (csi1200textbook) from the seller with id +5550001. Following that, the buying agent placed an order with that seller because the preferred price was acceptable to the buyer (we assumed that the acceptable maximum price of buyer was 100.).



(a) MPE0 received a SMS that the best price was found

(b) MPE0 received a SMS that the purchase order was placed

(c) MPE1 received a SMS that the product was ordered

Figure 5-2(a)-(c): SMS Messages Received by the Personal Agent from the Mobile Agent

As illustrated in Figure 5-2(b), a SMS message informed the buyer that the seller could be contacted once a purchase decision is made. On the seller's side, the selling agent also informed the seller with a SMS message, as illustrated in Figure 5-2(c), that the related item (csi1200textbook) was offered to the buyer with id +5550000. Both buyer and seller therefore were relieved from the tedious negotiation procedure. Through the negotiation process of the buying and selling agents, students gained valuable information about making trading decisions. Buying agents can finally make an agreement with selling agents when the required item and price are matched. Mobile users then receive text messages from their servers, displayed on the screen of the simulators.

Third, we looked the ACL messages delivered between agents. Figure 5-3(a) shows a Message Sequence Chart (MSC) that describes a creation process of a mobile agent. The proxy agent took the request forwarded from the servlet and sent a REQUEST message to the AMS agent. The AMS agent replied with an INFORM message and informed RMA that a new mobile agent was created. With that, this new agent needed to register in the system, and then the AMS agent received a REQUEST message.

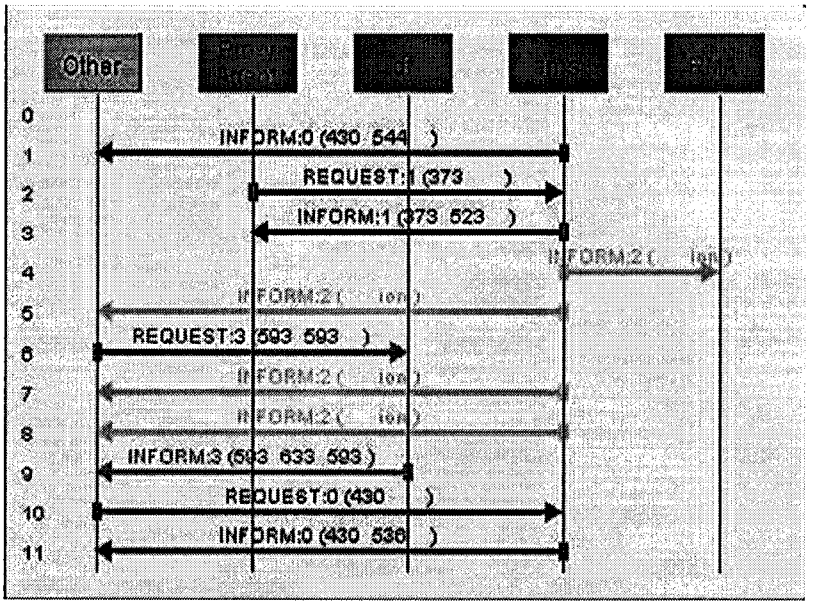


Figure 5-3(a): Message Sequence Charts Described a Creation Process of an Agent

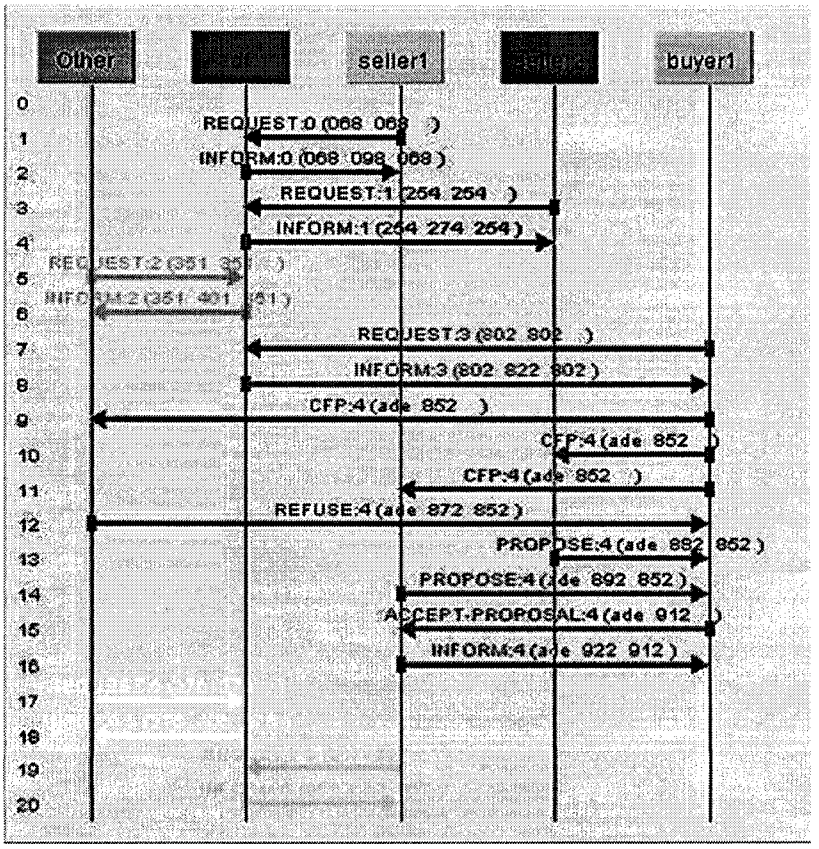


Figure 5-3(b): Message Sequence Charts Described a Negotiation Process of Agents

Figure 5-3(b) shows a MSC that described a CFP negotiation process of a buying agent interacting with other agents. In this scenario, we suppose that among the sellers which exist in the system, only seller1 and seller2 offered the item that buyer1 wanted. However, the price offered by seller1 was lower than that of seller2. The first two messages describe the procedure that seller1 registered and posted its service to the DF agent. After buyer1 entered this system, this buying agent initiated a negotiation process. At the beginning, buyer1 acquired a seller list from the DF by sending a REQUEST. After that, buyer1 sent the CFP to all related sellers. Then, buyer1 received a PROPOSE from seller1 and seller2 respectively, as well as a REFUSE from other sellers. At this point, buyer1 needed to estimate the PROPOSEs. Since seller1 offered a more ideal price than seller2, buyer1 took the desired one and then sent an ACCEPT-PROPOSAL to seller1. Seller1 replied with an INFORM message to buyer1 and then they reached an agreement on the related item and price. Seller1 finally deregistered the related item via the DF agent since this item was sold.

Finally, we observed the migration process of a mobile agent. Mobile agents should be active in their servers within the specified time and migrate among the servers. Thus, we developed a scenario where we supposed that a buying agent started from server1 and continued searching the related service in server2 and server3 sequentially. We set up two parameters for this mobile agent: maximum server active time was set to 100 seconds; and total lifetime was set to 850 seconds. A migration route was depicted in Figure 5-4. As expected, the buying agent contacted the other agents in the server1 and then migrated to server2 after approximately 100 seconds. Similarly, the buying agent communicated with other agents in server2 and then traveled to server3. The same things happened in server3. Because there were no more sites to be visited, the buying agent migrated back to server1, ending with its first round of migration. The second round was started since the total lifetime was not reached. All the ways, the buying agent did the same things. We assumed that no sellers offered the required services to this buying agent. As the time elapsed, the buying agent was in its third round and roamed into server2. In this phase, the buying agent used up it lifetime of 850 seconds and predicted an ending of its lifecycle. Therefore it migrated back to the host server1, even though the third round trip was not finished.

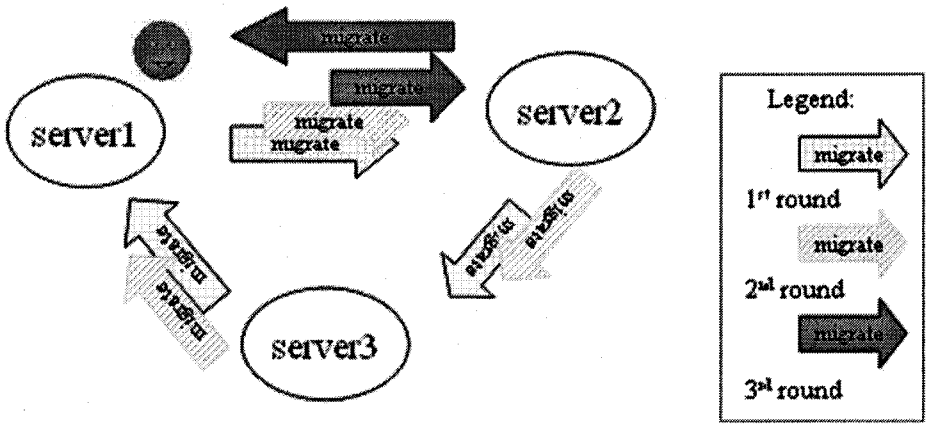


Figure 5-4: Migration Process of a Mobile Agent

In another scenario, we used the same parameters for the buying agent, except that the total time was increased to 1000 seconds. The difference was that a selling agent was dispatched in server2 at the moment the buying agent was ready to launch its third round trip. This selling agent exactly offered the service that the buying agent needed. As expected, the two agents met together and reached an agreement after negotiating with each other. This experiment confirmed that after completing its task, the buying agent migrated back to server1, regardless of more available lifetime to be exhausted. Figure 5-4 illustrates this experiment.

5.3 Analysis of Results

This section discusses some effective efforts we made towards the objectives. Some advantages and disadvantages based on the observations are presented as follows.

5.3.1 Usability and Mobile Devices Benefits

As a presentation to the user, the developed GUI of the personal agent did not require complex screen input and each function was implemented in one-step interaction. On the client side, therefore, it is easy for users to learn how to manipulate the mobile devices and participate in the e-businesses. Also, it requires users with minimal interaction and then takes users less time to accomplish a task. On the server side, the Web service allowed users to

access its functionality of launching the corresponding agents to perform tasks delegated by users. Therefore good usability was practiced in the experiments. For the mobile phone, it mitigates the limitations of smaller screen size and clumsy input capability, thereby reducing the complexity of user interactions. To make efficient use of the limited bandwidth of the wireless networks, we designed a method that a user can disconnect from the network after launching a buying or selling agent. Thus, the user only incurs minimal connection time and connection expense. In addition, no further computations are needed in the personal agent, because the buying or selling agent offloaded computations to available server resources. Mobile devices therefore can decrease the expense of battery life and survive from limited computing capacities. Since users' mobile devices are connected to the network only when needed, this makes efficient use of limited bandwidth and reduces the network traffic. Furthermore, it helps cell phone users save money.

One disadvantage here is the matching problem of the item. It is known that an item description may not be enough to describe a product or service. In the experiment, we developed an imperfect algorithm for agents to match the related item, that is, explicit comparison of description. We believed that, in future work, semantic technology could be applied to address a problem about fuzzy comparison. Considering the computing capacity and the presenting simplicity of the mobile devices, we had to make such a trade-off.

At the present, agents perform the tasks on their users' behalf in one-stop mode; that is, after specifying a desired item, the users do not need to interfere with the agents until receiving the negotiation results. It therefore brings the convenience to users but it also presents a drawback that users have to specify an exact item to buy, especially for a buyer. A possible solution to walk around this drawback is: buying agents could return a list of similar items for buyers to select before they make a contract with selling agents; the options display on the screen of mobile devices for buyers to refine their needs. However, this solution conflicts one of our objectives, that is, the manipulation on mobile devices will inevitably increases.

5.3.2 Mobility

The performance of agent migration depends on several factors, such as network quality, the size of agent code etc. Because there is no available benchmark suite for mobile agents, we did not assess the performance of the entire system and focused mainly on the specified migration times and fixed migration itinerary. In the experiments, we found that the two parameters, maximum server activity time and total lifetime, had effective impacts on the mobile agents' migration strategy. On the basis of these parameters, the mobile agents autonomously directed themselves to the next hopping server along the route or to the host server after completing their tasks. Conveniently, users, through the GUI on the mobile devices, can determine these parameters. To fulfill a given task, the buying or selling agent and the personal agent (the representative of the user) work well together. From the view of the networks load, the proposed framework avoids high network latency and reduces the network traffic since buying or selling agents locally communicate with other agents in the system. Also, the instant SMS message enables the buying or selling agent to inform the results to the personal agent, that is, they become a pair of peers.

5.3.3 Extensibility

The proposed framework is extensible. On the one hand, XML-based communication is used to enhance extensibility. On the other hand, the framework could be easily extended to B2C, or even B2B business models. Not only individuals but also business companies can be attached to the framework. Actually, an individual customer may be growing up to a small company once he/she operates the business more professionally and broadly. As a matter of fact, since mobile phones and PDAs are already being used as extended enterprise tools, business companies, such as retailers and suppliers, can publish their product services on their servers via mobile devices. As long as these businesses take part in the framework parties, they could benefit from the automatic discovery of business partners. Also, it is possible for businesses, especially for retailers, to sell to potential buyers their products in an extra advertising way. For online retailers, they can expand their retail business onto the phone so that increase sales. Shopping through the phone becomes very alive and becomes an attractive new medium for retailers. In such a sense, this framework is an integration model

of C2C, B2C and B2B e-commerce. Nonetheless, using mobile devices for complex tasks can be quite frustrating (e.g. difficult to enter data), so people will just not use it. An idea is to incorporate targeted messaging or advertising into our model, where businesses could send a message to users who are physically located in their vicinity. Agents could negotiate a transaction, and the buyer would already be located nearby to complete the purchase and pick up the item.

5.3.4 Scalability

To promote scalability, some measurements were taken: the personal agent was excluded from the MAS and distributed JADE agents in the different containers. It is unnecessary for the MAS to manage personal agents since they run on the each single mobile phone, respectively. Considering the large amount of mobile users involved in the system, the MAS benefits from lower workload without managing such personal agents. It can be noted that buying or selling agents will be linearly increased in scale upon the requests from the users. So, it is reasonable to assign them to different containers to balance the whole system workload. The lifecycle of the servlets is controlled by servlet containers which are built in the Web services server. The performance of Web services server would have an impact on the multi-threading capability of the servlets. In the experiments, we simply populated 30 mobile agents on each mediator-server, and the observation showed that they all ran in a good state. To some extent, scalability depends on the computing capability and available consuming resources of the server. Although there were not enough tests conducted in the experiment to verify the high scalability, some researches have attempted to evaluate it using JADE. Chmiel et al. [6] and Vitaglione et al. [54] presented the results of a high scalability and excellent performance in JADE through some scenario measurements. Based on the existing research, our intention in future is to benchmark both the software and the hardware and to demonstrate the scalability of the proposed framework.

5.3.5 Interoperability

We focused primarily on the interoperability between different systems for mobile agents. Interoperability is required in the systems where mobile agents can migrate into and leave off. Mobile agents in one system therefore can interact and communicate with other agents in the second system. In the framework, FIPA ACL message is used as the communication language during the interaction between agents. Adopting such a mechanism helps to improve the interoperability in mobile agent communities.

5.3.6 Security Issues

As for e-business and e-commerce applications, transaction security will remain a major issue for mobile commerce applications. When it comes to the online payment, people are concerned about the security of transactions, especially those that involve a great amount of money. In the proposed framework, however, both buyer and seller do not need to worry about their transactions. One of reasons is that no payment module was presented in the architecture. Admittedly, that does not mean that the payment module is unnecessary to consider. (This would be enhanced as a new feature in the system in the near future.)

However, an important security issue must be addressed. When mobile agents travel, their code and data are transmitted over the network. The users should be assured that they are secure enough. JADE supports weak agent migration from container to container within the inter-platform or intra-platform. In the design, very little was done to protect agents from being attacked once they run in a remote container. Thus further research in this area is needed, e.g., consideration of possible attacks. Some possible security problems are listed as follows:

(1) Malicious mobile agents can try to access services and resources without adequate permissions. On the other hand, a malicious agent can assume the identity of another agent in order to gain access to platform resources and services, or simply to cause mischief or even serious damage to the platform.

(2) Mobile agents may suffer eavesdropping attacks from other mobile agents. A malicious agent can sniff the conversations between other agents or monitor the behavior of a mobile agent in order to extract sensitive information from it.

(3) Mobile agents may also suffer alteration attack from malicious hosts. To execute the agent and update its state, the host must definitely be capable of reading and writing the agent. A malicious host might steal private information from the agent or modify the agent to compute the wrong result or to misbehave when it jumps to another site.

Current research efforts in the area of mobile agent security adopt two different points of view [24]: First, from the platform perspective, the hosts need to be protected from malicious mobile agents such as viruses and Trojan horses that are visiting it and consuming its resources. Second, from the mobile agent point of view, the agent needs to be protected from malicious hosts. There are many mechanisms to protect a host against malicious agents. Digital signatures and trust management approaches may help identify the agent and evaluate how much it should be trusted. The malicious host problem, in which a malicious host attacks a visiting mobile agent, is the most difficult problem. We found some works in the literature about powerful techniques such as Sandboxing and Proof-Carrying Code (PCC). Sandboxing [58] is a software technique used to protect mobile agent platform from malicious mobile agents. PCC [29] introduces the technique in which the code producer is required to provide a formal proof that the code complies with the security policy of the code consumer. Therefore, it is envisaged that the security of mobile agents is an important issue that will encourage techniques and mechanisms for e-commerce in the future.

5.4 Summary

This chapter presented the evaluation that the mobile agent-mediated framework was subjected to. A number of addressed issues were briefly discussed after describing the technical feasibility of the suggested solutions. As confirmed by the experiments, mobile users connect to their servers only when they need to add new items or to cancel their tasks. This obviously results in such benefits as reduced bandwidth utilization, increased battery

life for mobile devices, and no complicated computation conducted in mobile devices. Also, mobile agents can move to various servers to negotiate autonomously, and mediator servers can accept mobile agents from outside their systems. This feature enables users to participate in multiple markets on the Internet. Further research is needed in mobile agent and security issues, such as confidential information carried and communicated by agents.

Chapter 6 Conclusions

6.1 Summary

In summary, the conceptual framework proposed in this thesis provides a feasible e-market environment for mobile e-commerce applications. This was successfully demonstrated through the implementation of a prototype system.

This thesis work is driven by the mobile needs for new solutions providing trading, mobile access and user support in the mobile e-commerce. We propose in this thesis a framework that allows traders to do business in inter-connected e-marketplaces by means of mobile, intelligent agents. This framework, which adheres to standardization efforts in multi-agent field such as FIPA, paves a possible way towards a near future when mobile agents can smoothly travel among different agent-based marketplaces to carry out tasks on their users' behalf. Our purpose of presenting such an idea is to improve our understanding of the value of mobility and to encourage the conceptual construction of a global community. We do not claim that buyers and sellers around the world would have to buy into this to make it work, and worldwide C2B e-commerce would be revolutionized thereby. In practice, however, we hope that our work would be useful on a smaller scale and leads to new investigations that may result in new solutions to the problems we addressed.

The proposed framework assists users in C2C e-commerce and enriches the resources for users to perform comparison shopping activities at the point of purchase. It was discussed in the previous section that this model could also be applied to B2C and B2B applications. The proposed framework, aimed at providing new capabilities for advanced mobile e-commerce solutions, employs an approach that integrates the advantages of intelligent and mobile agents. Intelligent agents are aiming at making trading decisions on behalf of the users that emulate human traders' behavior, while mobile agents are used to extend that support by allowing users to remotely participate in several e-marketplaces.

In addition, we believe that intelligent and mobile agent technology is also a promising solution to the problems of low speed, high latency and limited computing ability that current wireless network is facing. It is envisaged that this approach can provide several benefits to industry including the reduced participation costs and easy access from different mobile devices.

6.2 Future Work

Currently, we propose an initial conceptual framework that can be improved. In [60] [61], we already discussed some concepts about agent migration and user's mobility (the ability to conduct e-business via mobile devices anyplace and anytime). This thesis further demonstrates the proposed framework fully. Using this work as a starting point, a number of future research directions are outlined as follows:

(1) Negotiation protocols do not have to be hard-coded into the agents. Instead, mobile agents can adapt to all kinds of negotiations when they arrive at a new remote location. Thus, the framework paves the way for future research where more general architectures can be explored to allow mobile agents to take part in various negotiation protocols, such as factor negotiating (price, quality, delivery time etc.), electronic contracting, and so on. Currently, the negotiation strategy module is only a purchase determined by price (agents seek a preferable price by a fixed amount). FIPA defines auction protocols (e.g. Dutch and English auctions) as well as simpler strategies such as: fixed pricing, fixed pricing with a discount, and so on. We plan to add them into negotiation protocols in future research.

(2) Items are described only by their names. Obviously, other attributes, such as color, age, terms of warranty and delivery should also be considered. We believe that semantic technology would enrich the description of items and can help to solve this problem. It should be noted that the small screen of mobile devices will bring inconvenience to users when they specify many attributes of an item. A possible solution is to make use of the persistent memory of mobile devices to store users' preferences.

(3) Mobile agent technology currently has some limitations, such as identity management, fault tolerance, protection of agents, and resource security. These limitations have brought up many concerns about the practical utilization of mobile agents. For example, primarily in the area of security, e-commerce applications involve dealing with money and thus users will hesitate to use mobile agents unless mobile agents are secure enough to be trusted. In the situation presented in this thesis, the mobile agents representing different buyers or sellers migrate over the Internet and then execute themselves on remote computers. These mobile agents are thus exposed to an open environment and may become vulnerable. Since the mobile agents execute on unknown computers and interact with unknown agents, a reliable security infrastructure is vitally needed for the future development of the intelligent mobile agent-based e-marketplaces.

(4) An additional component, a reputation system, will be necessary in our architecture. At this reputation system, agents could sign binding contracts and check user's credit histories and reputations. The trust problem will be further studied in the continuing research (e.g., Jøsang and Ismail [20] present a Beta Reputation System).

(5) In a real business situation, we would have to ensure that messages are reliably delivered to the mediator server from the personal agents. Although this communication protocol's reliability is not detailed in our architecture currently, we could use a reliable transport at the very least, such as Reliable HTTP (HTTPR) [51], for the communication between the personal agents and the mediator server. Another consideration is to encrypt the communication. Encryption technologies can also help ensure that even intercepted transmissions cannot be read easily.

(6) The argument might be made that agents in the system should be able to send SMS text messages to users in different languages. Obviously, the traders who come from different countries may only read their local languages. From this perspective, the system should provide an additional translation service for agents. An issue that should be taken into consideration is the possibility of a mistranslated text causing embarrassment to the communicating parties. In the meantime, users ought to be made aware that what they receive is a translated text.

(7) Currently, the experiments have been running on three computers, where all buyers and sellers are located. More generally, experiments with a large number of computers should be performed in the near future for the purpose of establishing scalability of the system.

References

[1] Bellifemine, F., Caire, G., Trucco, T., & Rimassa,G. (2006). JADE Programmer's Guide. Retrieved November 12, 2006, from http://jade.cselt.it/docs.

[2] Bluetooth from Wikipedia, the free encyclopedia, Retrieved July 7, 2006, from http://en.wikipedia.org/wiki/Bluetooth

[3] Braun, P., & Rossak, W. (2005). Mobile Agents: Basic concepts, mobility models, & the Tracy toolkit (pp. 8-31). San Francisco: Elsevier Inc.

[4] Chavez, A., & Maes, P. (1996). Kasbah: an agent marketplace for buying and selling goods. In *Proceedings of the first international Conference on the practical Application of Intelligent Agents and Multi-Agent Technology.* London, U.K.

[5] Chiang, H., & Liao, Y. (2004). An agent-based framework for impulse-induced mobile shopping. In *Computer and Information Technology, 2004, CIT 04*, The Fourth International Conference, pp. 509 – 514.

[6] Chmiel, K., et al. (2004b). Testing the Efficiency of JADE Agent Platform. In *Proceedings of the 3rd International Symposium on Parallel and Distributed Computing*, pp. 49-57. Los Alamitos: IEEE Computer Society Press

[7] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., & Weerawarana, S. (2002). Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI, *IEEE Internet Computing Volume 6*, Issue 2, pp.86 – 93.

[8] Dasgupta, P., Narasimhan, N., Moser, L.E., & Melliar, P. (1999). MAgNET: Mobile agents for networked electronic trading. *IEEE Transactions on Knowledge and Data Engineering*, 11(4): 509–525.

[9] eBay Auction Software Snipe Bidding program Sniper, Retrieved July 6, 2006, from http://cgi.ebay.com

[10] The Evolution of UDDI, UDDI.org White Paper, Retrieved Nov.12, 2006, from http://www.uddi.org

[11] Fonseca, S., Griss, M., & Letsinger, R. (2001). An Agent-Mediator E-Commerce Environment for the Mobile Shopper. HP Technical Report HPL-20010157.

[12] The Foundation for Intelligent Physical Agents, Retrieved October 1, 2006, from http://www.fipa.org

[13] FIPA abstract architecture specification, Retrieved October 21, 2006, from http://www.fipa.org/specs/fipa00001/SC00001L.html

[14] FIPA ACL message structure specification, Retrieved October 21, 2006, from http://www.fipa.org/specs/fipa00061/SC00061G.html

[15] FIPA Interaction Protocol Library specification, Retrieved October 21, 2006, from http://www.fipa.org/specs/fipa00025/

[16] Garcia, A., Kulesza, U., & Lucena, C. (2004). Aspectizing Multi-Agent Systems: From Architecture to Implementation. *Software Engineering for Multi-Agent Systems III*, Springer-Verlag, LNCS 3390, pp. 121-143.

[17] Impulse, Retrieved July 21, 2006, from http://agents.media.mit.edu/projects/impulse/

[18] Jade Agent Development Framework, an Open Source platform for peer-to-peer based applications, Retrieved July 21, 2006, from http://jade.cselt.it/

[19] The Java ME platform- the Most Ubiquitous Application Platform for Mobile Devices, Retrieved June 2, 2006, from http://java.sun.com/javame/

[20] Jøsang, A. & Ismail, R. (2002). The Beta Reputation System. In *Proceedings of the 15th Bled Electronic Commerce Conference*, Bled, Slovenia.

[21] Jupiter Research, Retrieved October 11, 2006, from http://www.jupiterresearch.com

[22] Keegan, S., & O'Hare, G. (2002). EasiShop: context sensitive shopping for the mobile user through mobile agent technology. *Personal Indoor and Mobile Radio Communications, the 13th IEEE International Symposium*, Vol. 4, 15-18, pp. 1962 – 1966.

[23] Kotz, D., & Gray, R. (1999). Mobile Code: The Future of the Internet. In *Workshop Mobile Agents in the Context of Competition and Cooperation at Autonomous Agents'99*.

[24] Kotz, D., Gray, R., & Rus, D. (2002). Future directions for mobile agent research. *IEEE Distrib. Syst.* Aug. 2002, Retrieved May 20, 2006, from http://dsonline.computer.org/0208/f/kot_print.htm.

[25] Kowalczyk, R., Franczyk, B., Speck, A., Braun, P., Eismann, J., & Rossak, W. (2002). InterMarket, - towards intelligent mobile agent e-marketplaces. *Engineering of Computer-Based Systems, 2002 Proceedings*. Ninth Annual IEEE International Conference and Workshop. pp. 268 – 275.

[26] Kowalczyk, R., Ulieru, M., & Unland, R. (2002). Integrating Mobile and Intelligent Agents in Advanced E-commerce: A Survey. *Agent Technologies, Infrastructures, Tools, and Applications for E-Services: NODe 2002 Agent-Related Workshops*, pp. 295-313.

[27] Lange, B.D., & Oshima, M. (1998). Programming and Deploying Java Mobile Agents with Aglets. Addison-Wesley Longman, Reading, Mass.

[28] Lange, D.B., & Oshima, M. (1999). Seven good reasons for mobile agents. *Communications of the ACM*, 42(3), 88-89.

[29] Lee, P., & Necula, G. (1997). Research on Proof-Carrying Code on Mobile-Code Security. In *Proceedings of the Workshop on Foundations of Mobile Code Security.*

[30] Leung, K., & Antypas, J. (2001). Improving Returns on M-Commerce Investments. Journal of Business Strategy, 22 (5), 12-13.

[31] Lu, L., & Zhang, Y. (2003). Intelligent mobile agents for efficient and inexpensive e-shopping. *Computer Software and Applications Conference, COMPSAC 2003 Proceedings*, 27th Annual International, pp. 607 – 609.

[32] Mennecke, B.E., & Strader, T.J. (2003). Mobile Commerce Technology, Theory and Applications. PA: Idea Group Publishing.

[33] Mihailescu, P., & Binder, W. (2005). A Mobile Agent Framework for M-Commerce. GI Jahrestagung (2), pp. 959-967.

[34] Milojicic, D., Breugst, M., Busse, I.,Campbell, J., Covaci, S.,Freidman, B., Kosaka, K., Lange, D., Ono, K., Oshima, M., Tham, C., Virdhagriswaran, S., & White, J. (1999). MASIF: The OMG mobile agent system interoperability facility. *Personal Technologies*, 2 (3).

[35] Mobile Commerce, Retrieved November 11, 2006, from http://en.wikipedia.org/wiki/Mobile_Commerce.

[36] Mobile Commerce Description, Retrieved November 11, 2006, from http://searchmobilecomputing.techtarget.com/sDefinition/0,,sid40_gci214590,00.html.

[37] Moreno, A., Valls, A., & Viejo, A. (2005). Using JADE-LEAP to implement agents in mobile devices. Retrieved November 11, 2006, from http://www.zdnet.de/itmanager/whitepapers.

[38] Object Management Group Common Facilities RFP3, Retrieved Oct, 20, 2006, from www.omg.org/docs/1995/95-11-03.

[39] Weiss, G. (Ed.) (2001). Multiagent Systems: A modern approach to distributed artificial intelligence, pp.79-120. The MIT Press.

[40] Sandholm, T. & Huai, Q. (2000). Nomad: Mobile Agent System for an Internet-Based Auction House. *IEEE Internet Computing*, Special issue on Agent Technology and the Internet, Vol. 4, No.2, pp. 80-86.

[41] Satoh, I. (2004). Configurable network processing for mobile agents on the Internet. *Cluster Computing*, 7(1), pp. 73-83.

[42] Satoh, I. (2004). Selection of mobile agents. In *24th IEEE International Conference on Distributed Computing Systems (ICDCS'2004)*, pp. 484-493.

[43] Satoh, I. (2004). Software testing for wireless mobile computing. *IEEE Wireless Communications*, 11(5), pp.58-64.

[44] Shafiq, M.O., Ali, A., Ahmad, H.F., & Suguri, H. (2005). A middleware based approach for integration of Software Agents and Web Services. In *14th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2005)*, Linkoping University, Sweden.

[45] Shi, N.S. (2004). Mobile Commerce Applications. PA: Idea Group Publishing.

[46] Shoham, Y. (1993). Agent-oriented programming. Artificial Intelligence, 60(1), 51-92.

[47] Shoham, Y. (1998). An Overview of Agent-oriented Programming. In *Software Agents,* J. M. Bradshaw (Ed.), 271-290. Cal: AAAI Press/The MIT Press.

[48] Su, Z., & Postel, J. (1982). The Domain Naming Convention for Internet User Applications. ARPANET Request for Comment No. 819.

[49] Suri, N., et al. (2000). NOMADS: Toward a Strong and Safe Mobile System. *Proc. 4th Int'l Conf.* Autonomous Agents, pp. 163–164. NY: ACM Press.

[50] Sycara, K.P. (1998). Multiagent Systems. AI Magazine 19(2): 79-92.

[51] Todd, S., Parr, F., & Conner, M. (2005). An overview of the reliable HTTP protocol. Retrieved Jan 11, 2007, from http://www-128.ibm.com/developerworks/webservices/library/ws-phtt/.

[52] Tu, M., Seebode, C., & Lamersdorf, W. (2001). Dynamics: An actor-based framework for negotiating mobile agents. *Electronic Commerce Research Journal*, 1(1/2).

[53] Varshney, U., Mallow, A., Jain, R., & Ahluwalia, P. (2002). Wireless in the Enterprise: Requirements and Possible Solutions. *Proceedings of the Workshop on Wireless*

Strategy in the Enterprise: An International Research Perspective, University of CA, Berkeley.

[54] Vitaglione, G., Quarta, F., Cortese, E. (2002). AAMAS Workshop on AgentCities, Bologna.

[55] W3C Cooperation White Paper, NOTE-WAP-19981030. Retrieved Jan 11, 2007, from http://www.w3.org/TR/NOTE-WAP.

[56] W3C Web Services Architecture. Retrieved Jan 11, 2007, from http://www.w3.org/TR/ws-arch/.

[57] W3C XML. Retrieved Jan 11, 2007, from http://www.w3.org/XML/.

[58] Wahbe, R., Lucco, S., Anderson, T.E., & Graham, S.L. (1993). Efficient software-based fault isolation. *In Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pp. 203-216.

[59] Wang, A.I., Sørensen, C.F., & Indal, E. (2003). A Mobile Agent Architecture for Heterogeneous Devices. *IASTED International Conference on Wireless and Optical Networks 2003,* IASTED.

[60] Weng, Z., & Tran, T. (2007). A Mobile, Intelligent Agent-Based Architecture for E-Business. Accepted, to appear in *International Journal of Information Technology and Web Engineering, Special Issue on Business Agents and the Semantic Web*, 2007.

[61] Weng, Z., & Tran, T. (2007). An Intelligent Agent-Based Framework for Mobile Business. Accepted, to appear in *Proceedings of the Sixth International Conference on Mobile Business (ICMB-07).*

[62] White, J.E. (1999) Telescript technology: mobile agents. *Mobility: processes, computers, and agents*. 460-493. New York: ACM Press/Addison-Wesley Publishing Co.

[63] Wooldridge, M., & Jennings, N. (1995). Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, 10(2): 115-152.

APPENDICES

Appendix A: Ontology

This appendix attempts to introduce an ontology into the proposed framework. However, in this thesis, only a simplified trading ontology (that has to be refined in the near future) is shown in Figure A-1, based on the UML class diagram.

An ontology is a set of concepts and predicates referring to a given domain. A class expresses each ontological template, and a slot of an ontological template whose type is primitive (e.g., *String*) is expressed as an attribute of the corresponding class. Those entity templates are referred to as concepts, for example, *item* and *agent*. A concept can also be used to define an attribute of the corresponding class. For instance, both *brand* and *price* are two attributes of *item*. Both *email* and *telephone number* are two attributes of *agent*. A concept can be defined as the superclass of other concepts. For example, both *buyer* and *seller* are derived from *agent*. Those relation templates are referred to as predicates, for example, *isOrdered* and *has* that can be either true or false. A relationship can be described as, for example, *who has what* or *what isOrdered* by *who*. The instances of concepts and predicates in the ontology are encoded inside the content of ACL messages exchanged by two or more agents in the system. As we introduced in Subsection 2.4.2, an ontology can be used to construct the content expression of an ACL message. In an information retrieval scenario, a buyer may use an ACL message including an expression (e.g. query :sender buyer1 :receiver seller1 :content trading (has, item1) :ontology trading) to query the product information of one seller. In this example, *buyer1* is an instance of the concept *buyer*, *seller1* is an instance of the concept *seller*, *item1* is an instance of the concept *item*, and *has* is an instance of the predicate *has*.

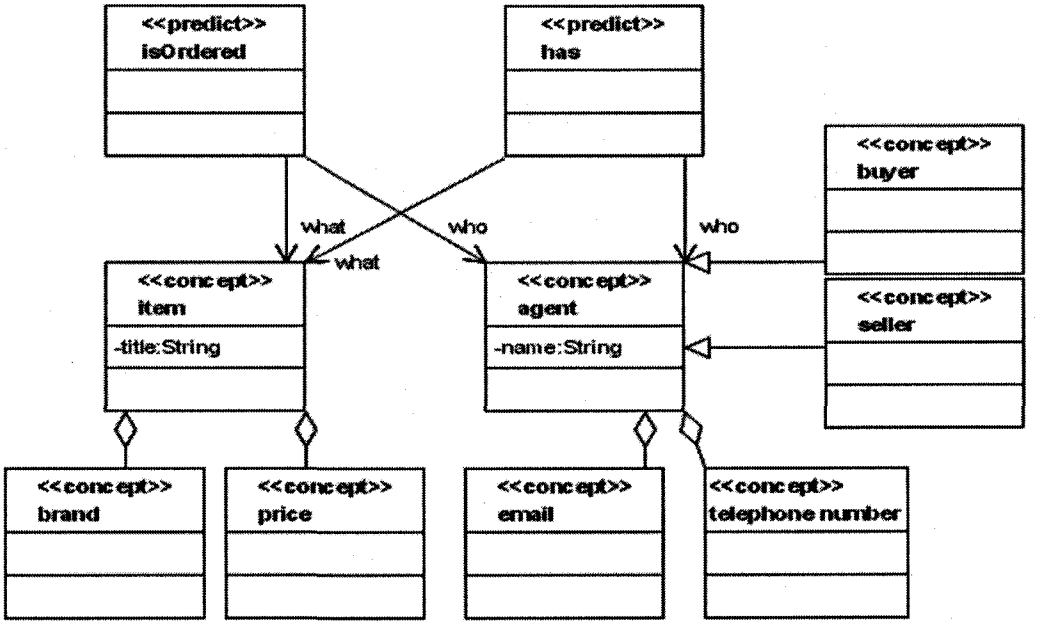


Figure A-1: Simplified Ontology of Trading Example in UML Class Diagram

Appendix B: Java Documentation

This appendix presents a summary of the Java documentation for all the classes of the system. Since the prototype of our system was fully implemented in the programming language Java, Java documentation helps us to analyze the structure of our application. The Java documentation was generated using the Javadoc generation tool. In our project, there are two primary packages, *examples.mobiletrading* and *serverscript*. The package is the top-level element in the documentation structure. The sub-level element, a Class Summary, lists all classes in the entire package. For example, we have three classes, *BuyerAgent*, *SellerAgent* and *WMABridgeAPI*, in the package *examples.mobiletrading*. Every class is demonstrated by respective sub-level elements, i.e., Nested Class Summary that summarizes nested classes, Field Summary that describes all fields defined in the class and Method Summary that presents all methods provided by the class.

Package examples.mobiletrading

Class Summary

BuyerAgent	Describes a buying agent
SellerAgent	Describes a selling agent
WMABridgeAPI	Describes a sms communication

examples.mobiletrading

Class BuyerAgent

```
java.lang.Object
  └jade.core.Agent
      └examples.mobiletrading.BuyerAgent
```

All Implemented Interfaces:

jade.core.TimerListener, java.io.Serializable, java.lang.Runnable

---

```
public class BuyerAgent
extends jade.core.Agent
```

Nested Class Summary

Nested classes/interfaces inherited from class jade.core.Agent
jade.core.Agent.Interrupted

Field Summary

Fields inherited from class jade.core.Agent
AP_ACTIVE, AP_DELETED, AP_IDLE, AP_INITIATED, AP_MAX, AP_MIN, AP_SUSPENDED, AP_WAITING, D_ACTIVE, D_MAX, D_MIN, D_RETIRED, D_SUSPENDED, D_UNKNOWN

Method Summary

Methods inherited from class jade.core.Agent
addBehaviour, blockingReceive, blockingReceive, blockingReceive, blockingReceive, changeStateTo, clean, doActivate, doClone, doDelete, doMove, doSuspend, doTimeOut, doWait, doWait, doWake, getAgentState, getAID, getAMS, getArguments, getContainerController, getContentManager, getCurQueueSize, getDefaultDF, getHap, getHelper, getLocalName, getName, getO2AObject, getProperty, getQueueSize, getState, here, notifyChangeBehaviourState, notifyRestarted, postMessage, putBack, putO2AObject, receive, receive, removeBehaviour, restartLater, restore, restoreBufferedState, run, send, setArguments, setEnabledO2ACommunication, setGenerateBehaviourEvents, setQueueSize, waitUntilStarted, write

examples.mobiletrading

Class SellerAgent

java.lang.Object
  └ jade.core.Agent
    └ **examples.mobiletrading.SellerAgent**

All Implemented Interfaces:

jade.core.TimerListener, java.io.Serializable, java.lang.Runnable

---

public class **SellerAgent**
extends jade.core.Agent

Nested Class Summary

Nested classes/interfaces inherited from class jade.core.Agent
jade.core.Agent.Interrupted

Field Summary

Fields inherited from class jade.core.Agent
AP_ACTIVE, AP_DELETED, AP_IDLE, AP_INITIATED, AP_MAX, AP_MIN, AP_SUSPENDED, AP_WAITING, D_ACTIVE, D_MAX, D_MIN, D_RETIRED, D_SUSPENDED, D_UNKNOWN

Method Summary

void	**updateCatalogue**(java.lang.String title, int price) This is invoked by the GUI when the user adds a new book for sale

Methods inherited from class jade.core.Agent
addBehaviour, blockingReceive, blockingReceive, blockingReceive, blockingReceive, changeStateTo, clean, doActivate, doClone, doDelete, doMove, doSuspend, doTimeOut, doWait, doWait, doWake, getAgentState, getAID, getAMS, getArguments, getContainerController, getContentManager, getCurQueueSize, getDefaultDF, getHap, getHelper, getLocalName, getName, getO2AObject, getProperty, getQueueSize, getState, here, notifyChangeBehaviourState, notifyRestarted, postMessage, putBack, putO2AObject, receive, receive, removeBehaviour, restartLater, restore, restoreBufferedState, run, send, setArguments, setEnabledO2ACommunication, setGenerateBehaviourEvents, setQueueSize, waitUntilStarted, write

examples.mobiletrading

Class WMABridgeAPI

java.lang.Object
  └ **examples.mobiletrading.WMABridgeAPI**

---

public class **WMABridgeAPI**
extends java.lang.Object

Field Summary

static int	**DEFAULT_CBS_MESS_ID**
static int	**DEFAULT_SMS_PORT**

Method Summary	
void	**handleSMS**(java.lang.String toPhoneNumber, java.lang.String messageStr) handle the sms command
void	**quit**()
void	**setPort**(java.lang.String argString) handle the setPort command

Package serverscript

Interface Summary	
Interface	Interface file with functions that client can call on this service.

Class Summary	
Implementation	Server functionality implementation.
Interaction	Interaction description Object used between the Servlet and the Proxy Agent to exchange an HTTP request and its response
ProxyAgent	Proxy Agent Agent linking the multi-agent system to the servlet by providing HTML interfaces.
Synchronizer	

serverscript

Interface Interface

All Superinterfaces:

java.rmi.Remote

```
public interface Interface
extends java.rmi.Remote
```

Interface file with functions that client can call on this service.

Method Summary	
java.lang.String	**request**(java.lang.String info, java.lang.String command) Request screen.

Method Detail

request

```
java.lang.String request(java.lang.String info,
                         java.lang.String command)
                         throws java.rmi.RemoteException
```

Request screen. Client state is defined by function parameters. Result of the function is string containing xml data describing next screen to be displayed on client.

Parameters:

`id` - Client or client state id. Interpreted only by server. Initial value of client id is 0. Server in xml data returned can request client to change his id.

`info` - Data entered or selected on client. In List it is name of selected item. In TextBox it is a value entered by user.

`command` - Command name pressed by user.

Returns:

xml string representing next screen to be displayed on client.

Throws:

`java.rmi.RemoteException`

serverscript

Class Implementation

```
java.lang.Object
  └ serverscript.Implementation
```

All Implemented Interfaces:

java.rmi.Remote, javax.xml.rpc.server.ServiceLifecycle, Interface

---

```
public class Implementation
extends java.lang.Object
implements Interface, javax.xml.rpc.server.ServiceLifecycle
```

Server functionality implementation.

Method Summary

void	**destroy**() Destroy service instance.
void	**init**(java.lang.Object context) Initialize service instance.
java.lang.String	**request**(java.lang.String info, java.lang.String command) Process server request.

Methods inherited from class java.lang.Object

```
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Method Detail

init

```
public void init(java.lang.Object context)
```

Initialize service instance.

Specified by:

`init` in interface `javax.xml.rpc.server.ServiceLifecycle`

---

destroy

```
public void destroy()
```

Destroy service instance.

Specified by:

`destroy` in interface `javax.xml.rpc.server.ServiceLifecycle`

---

request

```
public java.lang.String request(java.lang.String info,
                                java.lang.String command)
                         throws java.rmi.RemoteException
```

Process server request.

Specified by:

`request` in interface `Interface`

Parameters:

`info` - Data entered or selected on client. In List it is name of selected item. In TextBox it is a value entered by user.

`command` - Command name pressed by user.

Returns:

xml string representing next screen to be displayed on client.

Throws:

`java.rmi.RemoteException`

---

serverscript

Class Interaction

```
java.lang.Object
  └ serverscript.Interaction
```

```
public class Interaction
extends java.lang.Object
```

Interaction description Object used between the Servlet and the Proxy Agent to exchange an HTTP request and its response

---

Method Summary

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

serverscript

Class ProxyAgent

```
java.lang.Object
  └jade.core.Agent
      └serverscript.ProxyAgent
```

All Implemented Interfaces:

jade.core.TimerListener, java.io.Serializable, java.lang.Runnable

---

```
public class ProxyAgent
extends jade.core.Agent
```

Proxy Agent Agent linking the multi-agent system to the servlet providing HTML interfaces.

Nested Class Summary

Nested classes/interfaces inherited from class jade.core.Agent
jade.core.Agent.Interrupted

Field Summary

Fields inherited from class jade.core.Agent
AP_ACTIVE, AP_DELETED, AP_IDLE, AP_INITIATED, AP_MAX, AP_MIN, AP_SUSPENDED, AP_WAITING, D_ACTIVE, D_MAX, D_MIN, D_RETIRED, D_SUSPENDED, D_UNKNOWN

Method Summary

void	**setup()** Starting method of the agent: enable object2agent and set a cyclic behavior to check the queue.

void	**takeDown**() Disables the object-to-agent communication channel, thus waking up all waiting threads

Methods inherited from class jade.core.Agent
addBehaviour, blockingReceive, blockingReceive, blockingReceive, blockingReceive, changeStateTo, clean, doActivate, doClone, doDelete, doMove, doSuspend, doTimeOut, doWait, doWait, doWake, getAgentState, getAID, getAMS, getArguments, getContainerController, getContentManager, getCurQueueSize, getDefaultDF, getHap, getHelper, getLocalName, getName, getO2AObject, getProperty, getQueueSize, getState, here, notifyChangeBehaviourState, notifyRestarted, postMessage, putBack, putO2AObject, receive, receive, removeBehaviour, restartLater, restore, restoreBufferedState, run, send, setArguments, setEnabledO2ACommunication, setGenerateBehaviourEvents, setQueueSize, waitUntilStarted, write

Method Detail

setup

public void **setup**()

Starting method of the agent: enable object2agent and set a cyclic behavior to check the queue.

Overrides:

setup in class jade.core.Agent

---

takeDown

public void **takeDown**()

Disables the object-to-agent communication channel, thus waking up all waiting threads

Overrides:

takeDown in class jade.core.Agent

serverscript

Class Synchronizer

java.lang.Object
  └ **serverscript.Synchronizer**

---

public class **Synchronizer**
extends java.lang.Object

Method Summary

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Packages	
example.serverscript.connector	
example.serverscript.demo	

Package example.serverscript.connector

Interface Summary	
Interface	

Class Summary	
Interface_Stub	
Request	
RequestResponse	

example.serverscript.connector

Interface Interface

All Superinterfaces:

java.rmi.Remote

All Known Implementing Classes:

Interface_Stub

```
public interface Interface
extends java.rmi.Remote
```

Method Summary	
`java.lang.String`	`request(java.lang.String string_1, java.lang.String string_2)`

Method Detail

request

```
java.lang.String request(java.lang.String string_1,
                         java.lang.String string_2)
                         throws java.rmi.RemoteException
```

Throws:

java.rmi.RemoteException

example.serverscript.connector

Class Interface_Stub

java.lang.Object
  └ **example.serverscript.connector.Interface_Stub**

All Implemented Interfaces:

Interface, java.rmi.Remote

```
public class Interface_Stub
extends java.lang.Object
implements Interface
```

Constructor Summary

Interface_Stub(java.lang.String serverURL)

Method Summary

java.lang.Object	**_getProperty**(java.lang.String name)
void	**_setProperty**(java.lang.String name, java.lang.Object value)
java.lang.String	**request**(java.lang.String string_1, java.lang.String string_2)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

_setProperty

```
public void _setProperty(java.lang.String name,
                         java.lang.Object value)
```

_getProperty

```
public java.lang.Object _getProperty(java.lang.String name)
```

---

request

```
public java.lang.String request(java.lang.String string_1,
                                java.lang.String string_2)
                         throws java.rmi.RemoteException
```

Specified by:

`request` in interface `Interface`

Throws:

`java.rmi.RemoteException`

example.serverscript.connector

Class Request

`java.lang.Object`
  └ **`example.serverscript.connector.Request`**

`public class` **`Request`**
`extends java.lang.Object`

Constructor Summary

`Request``()`
`Request``(java.lang.String string_1, java.lang.String string_2)`

Method Summary

`java.lang.String`	**`getString_1`**`()`
`void`	**`setString_1`**`(java.lang.String string_1)`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Method Detail

getString_1

`public java.lang.String` **`getString_1`**`()`

---

setString_1

`public void` **`setString_1`**`(java.lang.String string_1)`

---

example.serverscript.connector

Class RequestResponse

`java.lang.Object`
  └ **`example.serverscript.connector.RequestResponse`**

`public class` **`RequestResponse`**

extends java.lang.Object

Constructor Summary

RequestResponse()
RequestResponse(java.lang.String result)

Method Summary

java.lang.String	**getResult**()
void	**setResult**(java.lang.String result)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

getResult

public java.lang.String **getResult**()

setResult

public void **setResult**(java.lang.String result)

Appendix C: Sample Code

A sample class code of a buying agent, which is a very important role in the system, is shown as follows. An instance of this class *BuyerAgent* can be initiated with some attributes such as the title of the targeted item, maximum server activity time, total lifetime, and preferred price. The multiple behaviors of a buying agent, such as migration and negotiation, are modeled as the inner classes that are implemented in a multi-threaded manner. The class provides some methods, for example, *setup()* for agent's initialization, *takedown()* for sending a SMS message before agent's termination. Note that a *behavior* of an agent is not the same concept as a *method* of a class object. In the object-oriented context, a class object's method is called a behavior of this object. In the agent context, a behavior of an agent means a task performed by this agent.

```
public class BuyerAgent extends Agent {
 // The title of the item to buy
 String targetTitle;
 String toPhoneNumber;
 int maxprice;
 // The list of known seller agents
 AID[] sellerAgents;
 int maxServertime;
 int maxTotaltime;
 String isMobile;
 //String fproperties = "agent.properties";
 Vector itinerari;
 final int MAXPRICE=99999;
 final int INTERVAL=45;
 int _bestprice;
 AID _bestSeller;
 int _besthop;
 int step;
 boolean found;
 // Put agent initializations here
 protected void setup() {
        // Printout a welcome message
        System.out.println("Hallo! Buyer-agent "+getAID().getName()+" is ready.");
        _bestprice=MAXPRICE;
        _besthop=0;
        found=false;
        // Get the title of the item to buy as a start-up argument
```

```
    Object[] args = getArguments();
    if (args != null && args.length > 0) {
                ArrayList al=(ArrayList)args[0];
                 targetBookTitle = (String) al.get(0);
                 maxprice=Integer.parseInt((String)al.get(1));
                 toPhoneNumber=(String) al.get(2);
                 isMobile=al.get(3).toString();
                 maxServertime=Integer.parseInt((String)al.get(4));
                 maxTotaltime=Integer.parseInt((String)al.get(5));
                 itinerari=(Vector)al.get(6);
                 System.out.println("Target good is "+targetTitle+"mobile?"+isMobile);
                 if (isMobile.equals("Y")){
              addBehaviour(new MyB(this,itinerari,targetBookTitle,maxServertime,maxTotaltime));
                      System.out.println("gointo migration program");
                 }else{
                      System.out.println("gointo stationary program");
               // Add a TickerBehaviour that schedules a request to seller agents every minute
addBehaviour(new TickerBehaviour(this, INTERVAL*1000) {
       protected void onTick() {
        System.out.println("Trying to buy "+targetBookTitle);
        // Update the list of seller agents
        DFAgentDescription template = new DFAgentDescription();
        ServiceDescription sd = new ServiceDescription();
        sd.setType("book-selling");
        template.addServices(sd);
        try {
             DFAgentDescription[] result = DFService.search(myAgent, template);
             System.out.println("Found the following seller agents:");
            sellerAgents = new AID[result.length];
            for (int i = 0; i < result.length; ++i) {
             sellerAgents[i] = result[i].getName();
              System.out.println(sellerAgents[i].getName());
            }
}
catch (FIPAException fe) {
    fe.printStackTrace();
}
// Perform the request
        step=0;
         myAgent.addBehaviour(new RequestPerformer(sellerAgents));
        }
      } );
        addBehaviour(new  TickerBehaviour(this, maxServertime*1000)
                    {
                          protected void onTick()
                          {
                                        System.out.println("Time?s up!");
                                        if(found)
                                        {
                                              step=2;
                                                          myAgent.addBehaviour(new
                              RequestPerformer(sellerAgents));
```

```
                                        }
                                        else
                                                myAgent.doDelete();
                                }
                        });
          }
         }
         else {
          // Make the agent terminate
          System.out.println("No target commodity title specified");
          doDelete();
         }
        }

 // Put agent clean-up operations here
 protected void takeDown() {
  // Printout a dismissal message
        String msg="Buyer-agent "+getAID().getName()+" terminating.";
    if (found==false) msg=msg+ "\nThere is no suitable ITEM ("+targetBookTitle+") and price
    ("+maxprice+") to buy";
        else msg=msg+"Finally found best price ("+_bestprice+") for ITEM ("+targetBookTitle+") from
"+_bestSeller.getName()+"at server "+itinerari.get(_besthop);
         System.out.println(msg);
   try{
          WMABridgeAPI wma=new WMABridgeAPI();
          wma.handleSMS(toPhoneNumber,msg);
          wma.quit();
    }
    catch (Exception e)
    {
          e.printStackTrace();
    }
 }
        /**
          Inner class RequestPerformer.
          This is the behaviour used by Book-buyer agents to request seller
          agents the target item.
         */
        private class RequestPerformer extends Behaviour {
          private AID bestSeller; // The agent who provides the best offer
          private int bestPrice;  // The best offered price
          private int repliesCnt = 0; // The counter of replies from seller agents
          private MessageTemplate mt; // The template to receive replies
         // private int step = 0;
          private AID[] sellerAgents;
          public RequestPerformer(AID[] sellerAgents)
          {
                  //step=0;
                  this.sellerAgents=sellerAgents;
          }
          public void action() {
          switch (step) {
```

```
            case 0:
             // Send the cfp to all sellers
                 found=false;
             ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
             for (int i = 0; i < sellerAgents.length; ++i) {
              cfp.addReceiver(sellerAgents[i]);
             }
             cfp.setContent(targetBookTitle);
             cfp.setConversationId("book-trade");
             cfp.setReplyWith("cfp"+System.currentTimeMillis()); // Unique value
             myAgent.send(cfp);
             // Prepare the template to get proposals
             mt = MessageTemplate.and(MessageTemplate.MatchConversationId("book-trade"),
                                 MessageTemplate.MatchInReplyTo(cfp.getReplyWith()));
             step = 1;
             break;
            case 1:
             // Receive all proposals/refusals from seller agents
             ACLMessage reply = myAgent.receive(mt);
             if (reply != null) {
              // Reply received
              if (reply.getPerformative() == ACLMessage.PROPOSE) {
               // This is an offer
               int price = Integer.parseInt(reply.getContent());
               if (bestSeller == null || price < bestPrice) {
                // This is the best offer at present
                bestPrice = price;
                bestSeller = reply.getSender();
                found=true;
               }
              }
              repliesCnt++;
              if (repliesCnt >= sellerAgents.length) {
               // We received all replies
                 if (found)
                  {_bestprice=bestPrice;
                   _bestSeller=bestSeller;

                 System.out.println("at present,found best price ("+bestPrice+") for item
("+targetBookTitle+") from "+bestSeller.getName());
                  }
                 if (maxprice>=_bestprice)
                  step=2;   //reached target price
                 else
                  step = 4;  //go on search best price

              }
             }
             else {
              block();
             }
             break;
```

```
            case 2:
              // Send the purchase order to the seller that provided the best offer
              ACLMessage order = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
      order.addReceiver(_bestSeller);
              order.setContent(targetBookTitle);
              order.setConversationId("book-trade");
              order.setReplyWith("order"+System.currentTimeMillis());
              myAgent.send(order);
              // Prepare the template to get the purchase order reply
              mt = MessageTemplate.and(MessageTemplate.MatchConversationId("book-trade"),
                                       MessageTemplate.MatchInReplyTo(order.getReplyWith()));
              step = 3;
              break;
            case 3:
              // Receive the purchase order reply
              reply = myAgent.receive(mt);
              if (reply != null) {
                // Purchase order reply received
                if (reply.getPerformative() == ACLMessage.INFORM) {
                  // Purchase successful. We can terminate
                    String msgtxt=targetBookTitle+" successfully purchased from agent
"+reply.getSender().getName();
                  System.out.println(toPhoneNumber+msgtxt);
                  System.out.println("Price = "+_bestprice);
                  try{
                      WMABridgeAPI wma=new WMABridgeAPI();
                      wma.handleSMS(toPhoneNumber,msgtxt + "Price = "+_bestprice);
                      wma.quit();
                  }
                  catch (Exception e)
                  {
                      e.printStackTrace();
                  }
                  myAgent.doDelete();
                }
                else {
                  System.out.println("Attempt failed: requested book already sold.");
                }
                 step = 4;
              }
              else {
                block();
              }
              break;
            }
          }

          public boolean done() {
                if (step == 2 && bestSeller == null) {
                        System.out.println("Attempt failed: "+targetBookTitle+" not available for sale");
                        //myAgent.doDelete();
                }
```

```
    return ((step == 2 && bestSeller == null) || step == 4);
  }
} // End of inner class RequestPerformer
private class MyB extends SimpleBehaviour{
        private String _msg;
        private boolean _done;
        private int _state;
        private int _hopcounter;
        private Vector _itinerari;
        private Agent _a;
        private int _maxServertime;
        private int _maxTotaltime;
        private AID[] _sellerAgents;
        private boolean _purchased;
        private String msgtxt;
         //private int _step;
        public MyB(Agent a, Vector itinerari, String item,int maxServertime,int maxTotaltime){
          super(a);
    // System.out.println("after supera");
          _msg = item;
          _a = a;
          _state = 0;
          _done = false;
          _purchased=false;
          _hopcounter = 1; //_hop0 is host
          _maxServertime=maxServertime;
          _maxTotaltime=maxTotaltime;
         // Properties p = new Properties();
          _itinerari = itinerari;
          step=0;
         /*  try {
                p.load(new FileInputStream(fproperties));
                parseItinerary(p);
           } catch(Exception e){
                e.printStackTrace();
                System.out.println("Error reading agent file properties.");
                myAgent.doDelete();
           }
          */
           addBehaviour(new  TickerBehaviour(_a, _maxServertime*1000)
                        {
                                protected void onTick()
                                {
                                        System.out.println("Time?s up in server!");
                                        System.out.println(_msg);
                                  System.out.println("\n\nMoving to " +
                                    _itinerari.get(_hopcounter));
                                AID a = new AID("ams@" +
                                  (String)_itinerari.get(_hopcounter) + ":1099/JADE",true);
                          a.addAddresses("http://" + (String)_itinerari.get(_hopcounter)
                                                                + ":7778/acc");
                                  PlatformID dest = new PlatformID(a);
```

```
                                _hopcounter++;
                                        _maxTotaltime-=_maxServertime;
                                        if(_itinerari.size() == _hopcounter)
                                                _state=1;
                                        myAgent.doMove(dest);
                                }
                        });
        addBehaviour(new TickerBehaviour(_a, INTERVAL*1000) {
                protected void onTick() {
                                System.out.println("Trying to buy "+targetBookTitle);
            // Update the list of seller agents
            DFAgentDescription template = new DFAgentDescription();
            ServiceDescription sd = new ServiceDescription();
            sd.setType("book-selling");
            template.addServices(sd);
            try {
              DFAgentDescription[] result = DFService.search(myAgent, template);
                        System.out.println("Found the following seller agents:");
                       _sellerAgents = new AID[result.length];
                       for (int i = 0; i < result.length; ++i) {
                        _sellerAgents[i] = result[i].getName();
                                System.out.println(_sellerAgents[i].getName());
                       }
            }
            catch (FIPAException fe) {
                fe.printStackTrace();
            }
            // Perform the request
                    step=0;
                    myAgent.addBehaviour(new RequestPerformer2(_sellerAgents));
                  }
                 } );
          }
        public void action(){
          switch(_state){
          case 0:

                if (_maxTotaltime<=0)
                {
                        System.out.println("\n\nReturning to host " +
                            _itinerari.get(0));
                        _state=2;
                if (_itinerari.get(0).toString().equals(_itinerari.get(_hopcounter-1).toString()))
                        {
                                System.out.println("already at home");
                        }
                        else
                        {
                                AID a = new AID("ams@" +
                                  (String)_itinerari.get(0) + ":1099/JADE",true);
                                        a.addAddresses("http://" + (String)_itinerari.get(0)
                                                + ":7778/acc");
```

```
                                        PlatformID dest = new PlatformID(a);
                                    myAgent.doMove(dest);
                                }
                            }
                            break;
                        case 1:
                            _state=0;
                            _hopcounter=1;
                            break;
                        case 2:
                            _done = true;
                            if (_purchased)
                            {
                                    try{
                              WMABridgeAPI wma=new WMABridgeAPI();
                              wma.handleSMS(toPhoneNumber,msgtxt + "Price = "+_bestprice);
                              wma.quit();
                           }
                           catch (Exception e)
                           {
                              e.printStackTrace();
                           }
                            }
                            //System.out.println(_msg);
                            System.out.println("Agent " + _a.getName() + " has ended his itinerary.");
                            myAgent.doDelete();
                        }
                    }
                    public boolean done(){
                        return _done;
                    }
                    private void parseItinerary(Properties p){
                        String prop = "";
                        int i = 0;
                        while(!(prop = p.getProperty("hop" + i, "")).equals("")) {
                            _itinerari.add(prop);
                            i++;
                        }
                    }
                }
}
```