A Proxy-Based Communication Scheme for Mobile Agents

by

Xiao Yan Zhou

A Thesis
Submitted to the Faculty of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree

Master of Science

Department of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES
*****
COPYRIGHT PERMISSION

A Proxy-Based Communication Scheme for Mobile Agents

BY

Xiao Yan Zhou

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

Of

MASTER OF SCIENCE

Abstract

Although the mobile agent paradigm provides great potential advantages over traditional approaches in distributed computing applications, there are still several issues to be addressed before the technology can be widely accepted. The performance of the communication protocol is one of the critical issues in mobile agent systems. A practical communication protocol for mobile agents must satisfy three basic requirements: location transparency, reliability and efficiency. Although many communication protocols have been proposed for mobile agent system and most of them are location transparent, these protocols usually compromise some aspects of reliability and efficiency.

The goal of this research is to develop a communication scheme for efficient location tracking of agents and reliable message delivery in mobile agent systems. In this thesis, a proxy-based communication protocol is developed to solve the reliability and efficiency problems in current schemes. The purpose of the proposed scheme is to provide reliable message delivery with minimum cost. The basic idea of the scheme is that a proxy agent acts as the message service center for all the other agents in the domain. The proxy agent can obtain location information from incoming messages and share this information among a group of agents. A simulation model is developed to estimate the performance of the proposed protocol. The result shows that the proxy-based protocol not only can guarantee reliable message delivery but also can decrease communication cost in most cases.

Acknowledgements

This thesis would have not been completed without the guidance and supports of my two advisors: Dr. Sylvanus Ehikioya and Dr. Neil Arnason. I would like to thank Dr. Ehikioya for motivating the initial ideas for this thesis and guiding me through the whole research process. I would like to thank Dr. Arnason for his financial assistance and his support on the simulation design.

I would also like to thank Dr. Peter Graham and Dr. Michel Toulouse for providing useful comments of the initial thesis proposal.

Finally, a special thanks to my mother and my husband for their constant support and encouragement throughout the years.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The increase in size and performance of computer networks has led to the rise of the decentralized architecture. As a result, the concept of distributed computing has become increasingly important in the last decade. Although current communication hardware and the underlying protocols, such as OSI and TCP/IP, enable some degree of distributed computing on the network, the performance of these protocols on distributed computing is far from satisfactory, especially in large scale distributed environments (e.g., the Internet). In the traditional client/server architecture, the client needs to have a permanent connection to a server, even if the connection is idle most of the time. Furthermore, the client process has to suspend while waiting for a response from the server. Moreover, when the waiting time exceeds the expected time, the client has to terminate the process and fail the whole task. As the size of the network increases rapidly, the inflexibility and inefficiency of the client/server architecture raises problems of scalability, heterogeneity, and efficiency.

Over the years, various innovative approaches [4] have been proposed for large-scale distributed computing. A mobile agent approach is one of the most promising for creating distributed systems. The idea is that a computer program can

transport itself from host to host in a network, and act autonomously to fulfill a user-assigned task. By moving the code to the data, mobile agent technology offers a number of advantages. These include the reduction of network load and network latency, increasing the overall performance, allowing the application to process data in the presence of network disconnection, working autonomously and asynchronously, and adapting to changing environments.

Since agents are mobile and must share information, a standard infrastructure that provides tracking and support for effective communication between agents must be provided. The need for this infrastructure is common among many multi-agent systems. In a multi-agent system, a server program usually runs on each node. The server program is responsible for hosting agents, allocating resources to the agents to run, administering the migration and communication of agents, and providing requested local services to the agents. In recent years, different agent systems have been developed and used for for distributed information retrieval [13], network management [10], data minding [6], distributed simulation [55], and electronic commerce [20].

1.1 Motivation and Problem Statement

Although the mobile agent paradigm provides great potential advantages over traditional approaches in distributed computing applications, there are still several issues to be addressed before the technology can be widely accepted. The performance of the communication protocol is one of the critical issues in mobile agent systems. Coordination between agents and the resources on the hosting execution environment is one of the fundamental activities. However, mobility has added complexity to the design and implied different problems in wide area computing environments. In general, a successful communication protocol should provide the

desired degree of location transparency, efficiency, reliability, scalability, security, and asynchrony.

Early research attempts to solve these problems used agent remote procedure call (ARPC) [43], which is analogous to the traditional RPC. With ARPC, programmers have to explicitly handle agent allocation and message delivery. Recently, the focus of research has shifted towards supporting location-transparent communication between agents, and a wide range of schemes, such as home-server [2, 22, 36, 38, 47], email [37], forwarding-pointers [21, 39, 52], broadcast [35, 42, 46], mailbox-based [14], and blackboard [15], have been proposed. However, each protocol has its drawbacks and limitations. As discussed more fully later, the home-server and email protocols have the triangular routing, central server constraints as well as the single-point failure problem; the forwarding-pointer protocol is vulnerable to host failures and the storage cost of forwarding information becomes too high in widely-distributed and highly-dynamic agent systems; the broadcast protocol can cause heavy network traffic when the number of agents in the system is large; and so on.

This thesis focuses on the reliability and efficiency issues of the communication protocol for multi-agent systems and proposes a proxy-based communication scheme to solve these problems in current schemes. The goal of the proposed scheme is to provide reliable message delivery with minimum communication cost. The problem addressed in this thesis is twofold:

1. How to minimize the communication cost, including both location update and message delivery cost.

2. How to guarantee that a message is delivered reliably to its intended receiver, especially during the receiver's migration.

1.2 Organization

The remainder of this thesis is divided into five chapters that provide further details on the problem area and related work. They describe the system architecture, simulation design and data analysis in depth.

Chapter 2 presents the background information for this research. The chapter begins with an overview of the mobile agent and agent systems. Furthermore, the design space of the communication protocols for a multi-agent system is described in detail as the foundation for the related work and motivation for the research. Current research in communication protocols for multi-agent systems is presented. This chapter concludes with a description of the design methodologies used in the thesis.

In Chapter 3, the proxy-based protocol developed in this thesis is introduced and described in detail. The system architecture and the functionality of the major components are presented. The system's operation and algorithms are discussed in depth. A limited-forwarding algorithm is developed to further improve the performance of the protocol. At the end of the chapter, a verification of the reliability of the protocols is provided.

A simulation model that simulates a proxy-based protocol and a home-server protocol is presented in Chapter 4. The purpose of the simulation is to compare the difference between the communication cost of these two protocols. The reasons for using simulation instead of a prototype are presented. The supported framework and the concept of discrete-event stochastic simulation are introduced. After that, different aspects of the simulation model, such as performance metrics, control parameters, conceptual model, initialization, and event generation, are discussed in detail. This chapter also provides a brief description of the verification and validation of the simulation model. The implementation details of the simulation model are presented in the last section.

Chapter 5 presents the analysis of simulation results. The basic proxy-based protocol and proxy-based protocol with limited-forwarding algorithm are analyzed independently. A sensitivity analysis is performed for the basic proxy-based protocol.

Chapter 6 outlines the contributions of this thesis and future research directions. The chapter summarizes the research, lists the contributions, and provides some suggestions for future research directions.

Chapter 2

Background

This chapter presents a broad overview of background material for mobile agent communication protocols. The first section describes the state of the art in mobile agent technologies and agent systems. The general concepts and design space for the communication protocols in multi-agent systems are discussed in the second section. The third section presents related work on communication protocols for multi-agent systems. The design methods, object-oriented and UML, are introduced in the last section.

2.1 Mobile Agents and Agent Systems

Mobile agents are a powerful programming paradigm that provides highly efficient and scalable solutions in complex distributed applications. The benefits of mobile agent technology include reducing network load, overcoming network latency, encapsulating proprietary protocols, executing autonomously and asynchronously, and adapting dynamically [31]. A mobile agent is an intelligent software entity that can migrate across a heterogeneous network system and accomplish a specific task on behalf of a user [16]. In various situations, agents at different locations must collaborate with one another to accomplish a task. The communication scheme

is responsible for ensuring effective interagent communication. To understand the research problem, a thorough knowledge of the characteristics of mobile agents is essential.

A key feature of a mobile agent is that its code is mobile [25]. In a classic client/server architecture, both clients and servers are stationary. The client requests some services offered by the server, and the server provides the services as well as the necessary resources. Mobile agent technology enhances the traditional client/server paradigm by exploiting locality in the access to distributed resources and by performing distributed operations in an asynchronous way. Mobility permits an agent to move to a destination where the required resources are available, and then work locally at the site. Thus, a mobile agent is neither bound to the originating host nor dependent on a continuous connection to the destination.

Agents are often described as intelligent and autonomous [44], which indicates that the agent can act independently without intervention from any other agents or users. After being dispatched by a user, a mobile agent can decide on its own where to go and what to do based on a predefined schedule, the network load, a host's computational load, or requests from external sources. A mobile agent may also spawn subagents and assign subtasks to the subagents to execute a distributed task concurrently when multiple resources are available. The ability to autonomously execute a task ensures that mobile agents react appropriately to unforseen events and can easily adapt to a changing environment.

Autonomy, however, does not mean isolation nor complete freedom. The real strength of agents is based on the large community of agents and the negotiation mechanisms and coordination facilities [50]. In various situations, agents at different hosts must cooperate with one another to accomplish complex tasks efficiently. Collaborative interactions can prevent hostile competition for limited resources between agents. Such interaction also allows agents to resolve conflicts and incon-

sistencies in information, current tasks, and world models, thus improving their decision-support capabilities in a dynamic environment. For example, in the Information Gathering (IG) area, due to the amount of available information, a task is usually decomposed into subtasks that entail sending agents to local data resources for information retrieval. Detecting interactions between agents, exploiting relevant information and resolving inconsistences in the acquired data are important aspects of these systems [45].

An agent system is another fundamental component involved in the mobile agents paradigm. An agent system is a distributed environment that supports creation, execution, migration and termination of mobile agents [30]. Mobile agents cannot exist outside an agent system. A multi-agent system offers flexibility to the programmer by hiding the underlying network architecture. Therefore, agent systems enable programmers to write distributed programs without knowing the detail of network structure. A mobile agent system may also provide support services to access other mobile agent systems, and provide open access to non-agent based software environments. To this end, many mobile multi-agent systems, such as Aglet [1, 30], Emerald [27], Voyager [5], JADE [9] and Concordia [58] have been developed.

As multi-agent systems become more pervasive, the collaboration between agents increases, which in turn raises the need for an efficient location-management and communication mechanism for mobile agents. The next section explores the common requirements of a communication protocol for multi-agent systems, which provides a foundation for the related work in the third section.

2.2 Design Space

A communication scheme for mobile agents is a mechanism that tracks the locations of mobile agents and facilitates interagent communication. A practical scheme should support two fundamental operations: location update and message delivery [57]. Location update is the action the scheme should take to maintain up-to-date location information after an agent moves to a new network location. Message delivery is the process of conveying a message from the sender to the receiver.

In addition to these two operations, the design of communication protocols for mobile agents is rather application-specific – different multi-agent systems impose different requirements on communication protocols. Therefore, one feasible solution for a particular system may not be applicable to another system with different migration and communication patterns. A number of requirements, however, are essential to most multi-agent systems. These common requirements of a communication protocol for multi-agent systems are summarized below.

2.2.1 Location Transparency

In a static network infrastructure, it is easy for a sender to know the address of a receiver since the location never changes, and thus, a sender can deliver a message straight to the intended receiver. However, in the mobile agent paradigm, message delivery becomes more complicated because a receiver's address can change frequently. To use direct communication protocols, such as TCP/IP [23, 24], RPC [54] and RMI [56], a programmer must explicitly handle agent migration and communication, which not only increases the development and maintenance cost but also restricts the flexibility and extensibility of an application. Therefore, it is important for a communication protocol to handle the location tracking of agents in

multi-agent systems transparently. This means a sender should be able to deliver a message to the receiver without knowing the receiver's current location and the process of locating the receiver should be completely hidden from the sender. By doing this, the multi-agent system offers the programmer more flexibility while making the underlying network architecture transparent.

2.2.2 Efficiency

The two tasks of minimizing the communication overhead for location update and for message delivery often appear to be in conflict. Strategies for locating a mobile agent in a distributed network range between two extremes: *full-information* and *no-information* [7, 51]. With the full-information approach, every site in the network has a database which maintains up-to-date information on the current location of all agents. This method minimizes the message delivery cost while location update cost is expensive. The full-information strategy is appropriate for a relatively static environment, where agents communicate frequently but move rarely. On the other hand, the no-information approach requires no updates for migration, but a search over the whole network is necessary for every message delivery. This method is appropriate for small-scale and highly-dynamic environments, where agents move frequently but communicate rarely. An efficient protocol should strike a balance between these two extremes to meet the requirements of some specific communication and migration patterns while minimizing the total costs of location update and message delivery.

2.2.3 Reliability

The reliability of a communication protocol for multi-agent systems involves two issues:

1. The vulnerability of the protocol to the failures in the distributed environment

2. Message losses or chasing problems

The first issue, also referred as the fault-tolerance problem, reflects the level of failure the communication protocol can accept. Some protocols are sensitive to any kind of failure in the network; one component failure may crash the whole communication system. Some protocols have higher tolerance and can continue to operate properly even after a certain number of nodes and links fail. The second problem can occur even in a fault-free network due to the asynchronous message passing and agent migration model. A message can be lost if it is delivered during the receiver's migration or the message can chase the receiver forever if the target agent moves frequently. A reliable communication protocol should address both issues to guarantee that a message can be delivered successfully to the receiver in any situation.

2.2.4 Scalability

Scalability is one of the key elements that measure the successfulness of a communication protocol for multi-agent systems. The scalability issue can be addressed from four aspects: (1) the size of the network; (2) the total number of the mobile agents; (3) the migration frequency of agents; (4) the size of the communication data. A scalable communication protocol should be able to preserve the quality of the services as size of any or all of the above parameters increase.

2.2.5 Security

In recent years, users' demands have raised the priority of security issues for mobile agent communication. In existing protocols, message passing between communi-

cating partners generally involves using various sites to process the message. At the same time, many agents operate autonomously which present a significant obstacle to agent activity management. The network complexity and agent mobility provide numerous opportunities for a malicious agent to intercept or overwrite the message during the transmission process. A secure communication protocol should be able to prevent an adversary compromising message exchange between partners.

2.2.6 Asynchrony

The decision of whether to use a synchronous or an asynchronous communication approach is application-dependent. In synchronous protocols, such as TCP [24] and RPC [54], the client and the server open a communication channel between both ends. The client application maintains the entire communication process until it receives the response message from the server. In asynchronous protocols, such as messaging [19], once the client application composes and hands off a message to the messaging system, the application continues execution.

Synchronous communication protocols have been standard networking protocols for decades and have proven capable of providing reliable delivery of packets. Unfortunately, synchronous communication requires that synchronization is established for a long duration, which is contradictory to the dynamic nature of multi-agent systems. In addition, disconnection usually is allowed in a mobile agent system, which also makes synchronous communication an unsuitable approach. On the other hand, although asynchronous communication supports agent mobility and disconnection, it cannot guarantee reliable message delivery. A designer should weigh the benefits of both methods and the requirements of the application before choosing the communication model that is best suited for the application.

2.3 Related Work

In multi-agent systems, a critical performance issue is the agent communication protocol. Much effort [2, 21, 37, 38, 42] has been devoted to ensure efficient location tracking of agents and reliable message delivery in mobile agent systems. To be able to send messages in a location-transparent fashion, researchers have proposed two different approaches:

1. Direct agent-to-agent communication, such as home-server [2, 22, 36, 38, 47], email [37], forwarding-pointers [21, 39, 52], broadcast [35, 42, 46], and mailbox-based [14].

2. Indirect interactions using shared places, such as blackboard [15].

The following sections presents the advantages and disadvantages of each kind of protocol in terms of the design space discussed in the previous section, where it is possible.

2.3.1 Home-Server

Many mobile agent platforms, for example Aglets [1, 30] and JADE [9] use the home-server protocol proposed in Sprite [22], Mobile Internet Protocol [2], and Mobile Agent System Interoperability Facility [38] to solve agent communication's problem. The home-server protocol works as follows:

- Each mobile agent associates with a stationary agent, which is called the mobile agent's home agent.

- Each home agent has a database to store the addresses of all agents that use this host as home agent.

- The sending agent knows the name of the receiving agent.

- A central naming server, called home server, maintains a binding between mobile agents' names and their home agents' addresses.

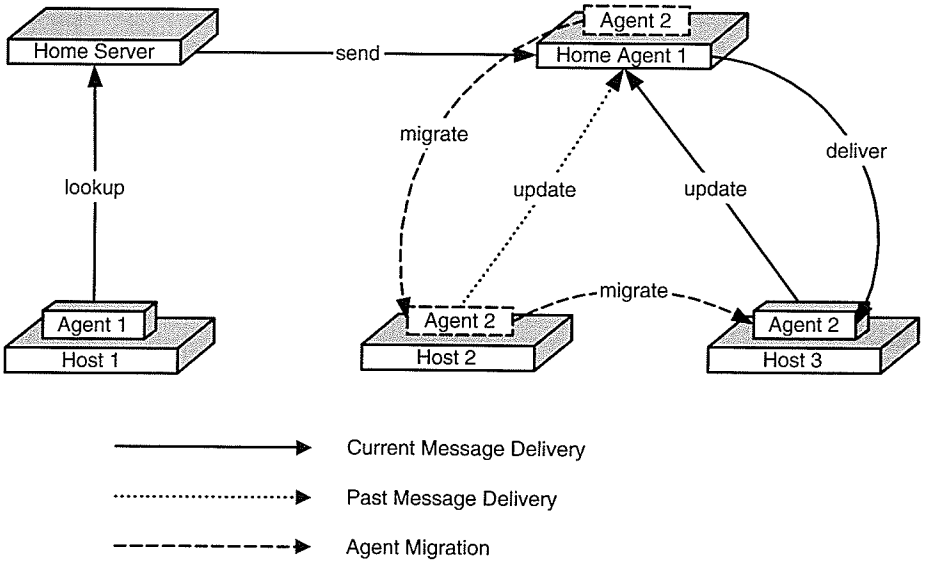


Figure 2.1: Home-Server Model

In the home-server approach, shown in Figure 2.1, a mobile agent must inform the home agent of its new location after each migration. To contact a mobile agent, the sending agent first delivers the message to the home server. Second, the home server routes the message to the receiving agent's home agent. Third, the home agent forwards the message to the receiving agent's actual location.

A basic motivation for using the home-server protocol is to support mobile IP addresses [2] while preserving compatibility with the current IP protocol, where hosts are unaware of agent mobility. However, the home-server protocol raises concerns for efficiency, reliability and scalability of the system even though the model is simple to implement and works well for small-to-medium distributed systems. Messages sent to a mobile agent always pass through the home server and a home agent, which causes a triangular routing problem. Both location update

and message delivery can incur a significant delay if the mobile agent is far from its home agent. Various cache-based strategies [36, 47] have been proposed to avoid the triangular routing problem. Nevertheless, these protocols cannot guarantee reliable message delivery because a message would be lost if the delivery happens during the receiver's migration. Furthermore, the central naming server puts a burden on the global infrastructure. The home server may become a performance bottleneck and a single-point of failure in a large-scale network, and therefore, prevents the scalability of this approach.

2.3.2 Email

The ffMain [37] system uses an email infrastructure for agent communication, which is similar to the home-server protocol. The only difference, as shown in Figure 2.2, is a home agent stores an incoming message in the receiver's message queue instead of forwarding it to the target agent. The mobile agent checks its home agent periodically for incoming messages. If the the agent's message queue is not empty, it can either pull the messages from its home agent to its current location or move back to its home and retrieve the messages locally.

Since it is the mobile agent that initiates the communication with its home agent, the email protocol can guarantee reliable message delivery and reduce the location update cost. However, this approach encounters the same problems as in the home-server protocol, such as triangular routing, central server constraints as well as the single-point-failure problem. Also, if home agents are not distributed equally, one host could be overloaded with the vast storage demands. Furthermore, the time intervals for a mobile agent to check its home place has a big impact on the system performance. If the time interval is too large, an agent might not be able to get its messages on time which would cause serious problems in systems that require prompt response. On the other hand, if the agent checks its home

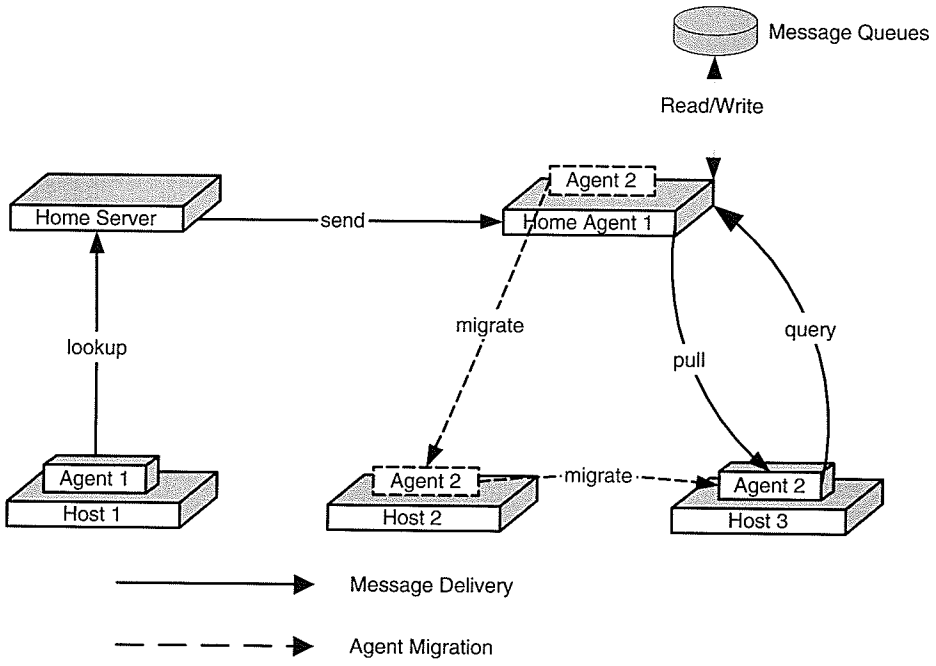


Figure 2.2: Email Model

place frequently, a large number of query and response message will induce high communication overhead. Therefore, the email protocol is only suitable for systems in which agents do not require instant messages.

2.3.3 Forwarding-Pointer

Emerald [27] and Voyager [5] systems use the forwarding-pointer protocols [21, 39, 52] for interagent communication. These protocols work as follows:

- Each mobile agent associates with a stationary agent, which is called the mobile agent's home agent.
- Each home agent has a database to store the initial addresses of all agents that use this host as home agent.
- Each host has a database to store the cached addresses of all agents that

migrated from this host.

- The sending agent knows the name of the receiving agent.
- A central naming server, called home server, maintains a binding between mobile agents' names and their home agents' addresses.

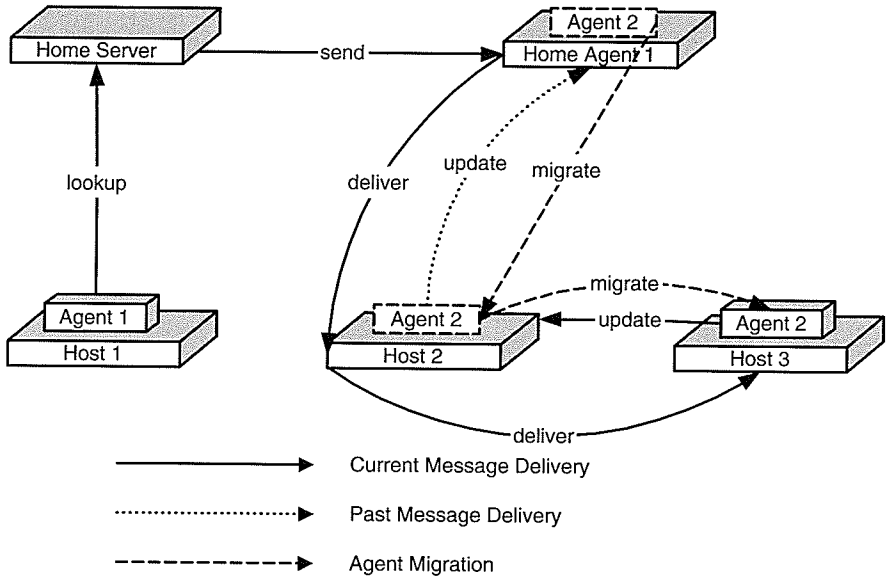


Figure 2.3: Forwarding-Pointer Model

In the forwarding-pointer approach, shown in Figure 2.3, after each movement, a mobile agent must inform the last host it visited of its current location. Accordingly, every host on the agent's migration path keeps a forwarding pointer to the next host on the path. Similar to the home-server method, a message is first delivered to the home server, then to the receiver's home agent. After that, the home agent sends the message to the last known location of the target agent. If the receiver is not at the last known location, the messages would be forwarded along the chain of pointers. Upon receiving a message, the target agent sends back

an invalid-cache message to the original sending agent to update the outdated address.

The dependency on the home agents and home server, as well as the location update cost are reduced in the forwarding-pointer scheme. On the other hand, the message delivery cost can be very high due to the redundant hosts on the migration path. Some path compression algorithms, such as lazy updates and back propagating information [39] along the chain, can be used to collapse the chain [41]. A serious drawback of this approach is its vulnerability to the failures of hosts, one failure on the migration path results in an unreachable target agent. A solution is to update its N previously visited hosts after a mobile agent reaches a new location, which can improve the system's tolerance of a failure up to N [40]. However, the chasing problem remains unsolved in forwarding-pointer protocols, that is, a message can follow an agent forever if the agent migrates frequently. In addition, the forwarding-pointer scheme is not practical for widely-distributed and highly-dynamic agent systems since the storage cost of forwarding information on every host increases as the chain grows.

2.3.4 Mailbox

Cao et al. [14] propose a mailbox-based scheme, which is a three-dimensional model for designing flexible and adaptive message delivery protocols in mobile agent systems. The protocol works as follows:

- Each mobile agent owns a mailbox which buffers all the incoming messages for its owner agent.
- Each mailbox associates with a stationary agent, which is called the mailbox's home agent.
- Each home agent has a database to store the initial addresses of all mailboxes

that use this host as home agent.

- The sending agent knows the name of the receiving agent.
- A central naming server, called the home server, maintains a binding between mailbox names and their home agents' addresses.

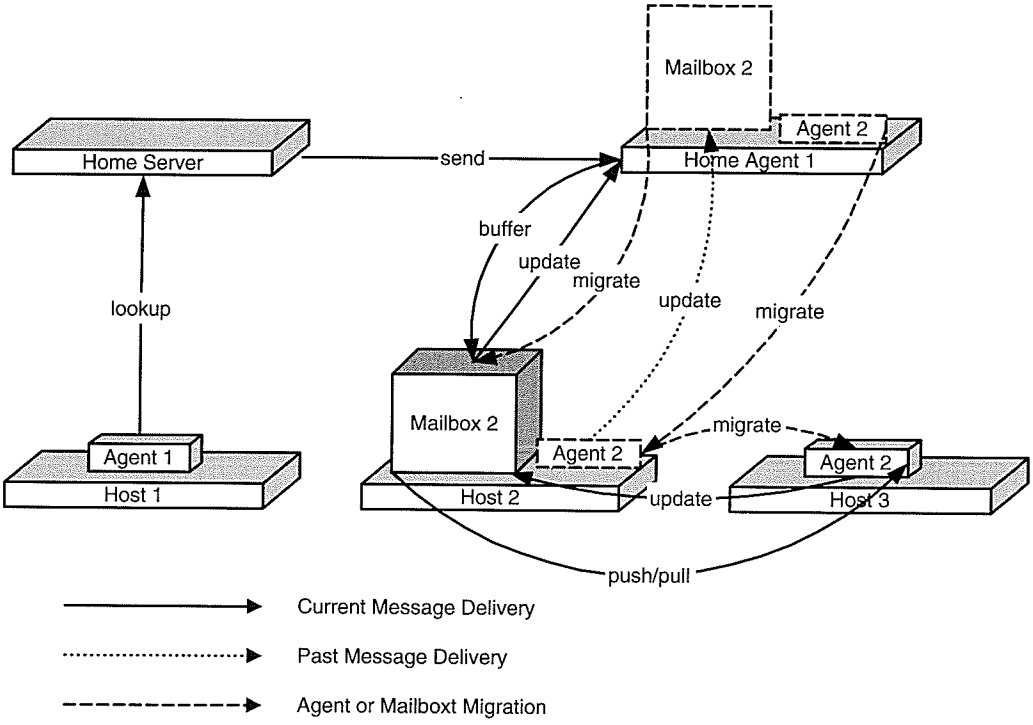


Figure 2.4: Mailbox-Based Model

In the mailbox-based protocol [14], shown in Figure 2.4, a mailbox may be detached from its owner agent and migrate to a different place. A mailbox must inform its last visited host and its owner agent of its current location after each migration. A mobile agent also has to report the new address to its mailbox after the agent reaches a new location. Messages sent to an agent are all buffered to its mailbox, and the agent later receives the messages by either a push or pull operation.

Users can customize the protocol to meet particular requirements by defining three parameters: *mailbox migration frequency, mailbox-to-agent message delivery,* and *migration-delivery synchronization* [14]. Mailbox migration frequency is the number of mailbox migrations within its mobile agent's migration interval. *No migration* means the mailbox never moves and the protocol acts as a home-server protocol. *Full migration* means the mailbox moves synchronously with its mobile agent and the protocol acts as a forwarding-pointer protocol. *Jump migration* means the mobile box can decide its mailbox migration frequency dynamically based on the number of incoming messages. *Mailbox-to-agent delivery* defines how a mobile agent receives messages from its mailbox. A user can decide whether mailboxes *push* messages to mobile agents or mobile agents *pull* messages from mailboxes. *Migration-delivery synchronization* defines different levels of reliability in message delivery. A user can choose to synchronize the host's message forwarding and the mailbox's migration, or the mailbox's message forwarding and the mobile agent's migration, or both of them.

By decoupling the mailbox from its owner agent, the mailbox-based model separates the location tracking and message delivering, which introduces great flexibility into the design space. The three-dimensional model provides users the potential to develop a new protocol that is best suited to a specific agent migration and communication pattern. However, mailbox's mobility also increases system's complexity, which in turn lessens the fault-tolerance and raises the location update cost of the system.

2.3.5 Broadcast

In the flooding broadcast approach [35, 46], shown in Figure 2.5, the sending agent knows the name of the receiving agent but does not use a central naming server to resolve the name. To send a message to a mobile agent, a source agent simply

broadcasts location queries, location notifications or pending messages to all of its neighbors. These neighbors in turn rebroadcast the message and this operation continues until all the hosts have received the message. Only the corresponding agent would process the receiving message.

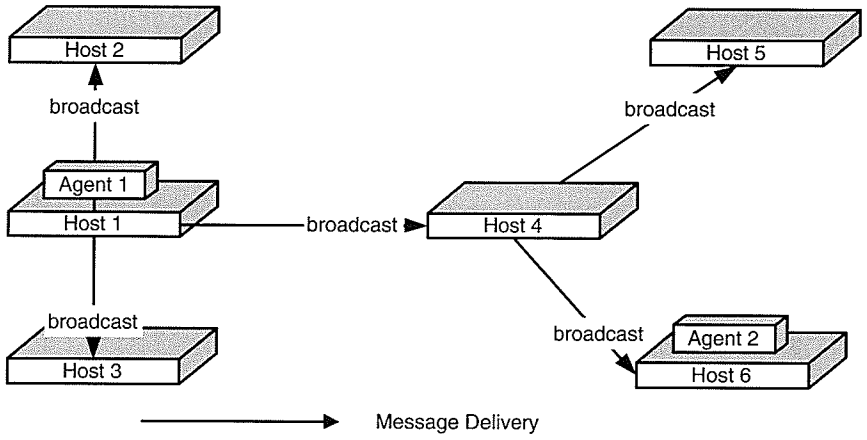


Figure 2.5: Broadcast Model

The broadcast protocol is simple to implement and has no location update cost. However, the protocol does not handle message losses due to agent mobility. A snapshot broadcast strategy [42] can guarantee message delivery to a specific agent as well as for group communications. However, the network could be overwhelmed by the enormous number of unnecessary messages when the number of agents in the system is large. Therefore, the broadcast protocol is impractical in a wide-area network and should only be used when all other methods fail.

2.3.6 Blackboard

The Ambit [15] system employs a blackboard protocol, in which the sender does not know the name of the receiver. To send a message to other agents, as shown in Figure 2.6, an agent simply writes the message to the local storage of the host

at which it is currently residing. After the agent migrates to another location, it may repeat the same action if necessary. For a receiving agent to read a message, it must move to the related host and retrieve the message locally.

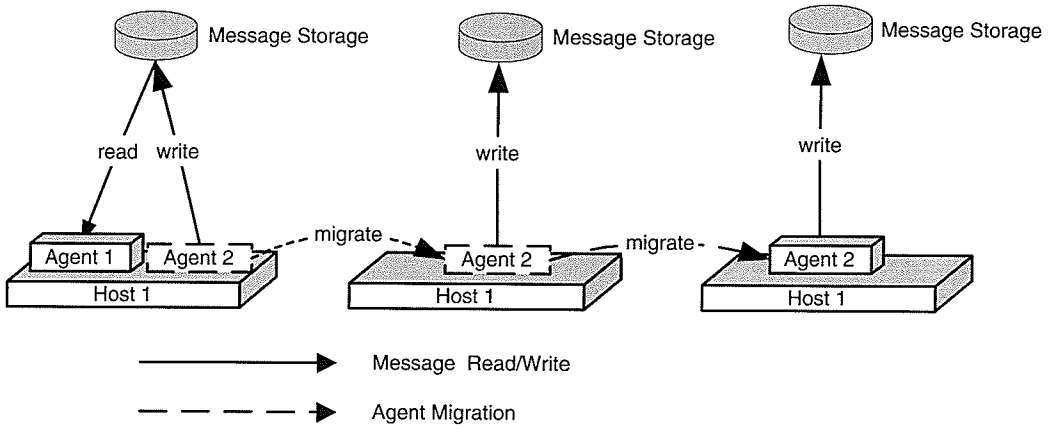


Figure 2.6: Blackboard Model

The blackboard protocol is the simplest among all agent communication models; there is no central server, no home agents, and no location tracking and update. Agents communicate with one another by simply writing and retrieving message locally. However, the blackboard model requires every host to maintain a message shared storage which agents can use to leave messages for others to read. Besides, it becomes the responsibility of a developer to ensure the receiving agent goes to the right locations to obtain the messages that are intended for the agent, which in turn increases the development and maintenance costs of the application. In addition, the message removal strategy also needs to be considered, since the storage capacity is finite, which puts additional complexity on the application.

2.4 Design Methodologies

In this research, the object-oriented design (OOD) method is used for the development of the simulation model. In contrast to the separation of data and function in a procedural development, the object-oriented method integrates data and functions into a whole [28]. In the object-oriented approach, a software system is described as a collection of objects. An object is a software entity that can perform a set of tasks. Each object is responsible for its own data and behavior, and details of implementation are hidden from the rest of the system. System functions are accomplished by object cooperation.

Object-oriented development methods have many advantages over structured methods. The organization of an OO system is closer to that of human activities and makes the system easier to understand. With proper design, the encapsulation and inheritance mechanism can increase reusability significantly. Furthermore, since all attributes and functions are encapsulated in objects, changes inside an object will not affect other objects as long as the interaction interface remains the same. The encapsulation allows programmers to work at a higher level and hide implementation details behind a message-passing interface. As a result, the system can be easily adapted to changing requirements and is easier to maintain.

Object-oriented modeling is a formal way of describing an existing domain as an assembly of objects. The use of modeling is essential for the creation of well-designed, robust and quality object-oriented software that meets the needs of its users. Since late 1980s, numerous object-oriented modeling languages, such as Booch [11], Coad-Yourdon [17], Fusion [18], OMT (Object Modeling Technique) [48], OOSE (Object-Oriented Software Engineering) [28] and Shlaer-Mellor [53], have been proposed. In mid 1990s, Grady Booch, James Rumbaugh and Ivar Jacobson created the Unified Modeling Language (UML) [12] based on the semantics and notation from Booch, OMT, OOSE and other prominent methods. Since then,

the UML has been widely adopted by many software development organizations and the notation it uses is becoming a worldwide standard for object-oriented modeling.

The UML is a graphical modeling langauge for visualizing, specifying, constructing and documenting software systems. It is process independent, and therefore can be used with most existing object-oriented development processes. The UML notations include a large set of graphical symbols, which are supported by well-defined semantics. From these symbols, various diagrams can be constructed to capture the information about the static structure and dynamic behavior of a system. These diagrams include use case diagrams, class diagrams, statechart diagrams, activity diagrams, sequence diagrams, collaboration diagrams, component diagrams and deployment diagrams. Use case diagrams are used for requirements analysis by modeling the interactions between actors and the system. Class diagrams are used to model the static aspects by showing a set of classes and their relationships. Activity diagrams support the functional perspective by showing control flow in business processes and internal operations. Sequence and collaboration diagrams are used to display messages passing between objects and entities within the system. Finally, component and deployment diagrams capture the physical aspect by showing how components are packaged and deployed.

In this thesis, the activity diagrams are used to describe the control flow in system operations and the class diagrams are used to present the conceptual model of the simulation.

Chapter 3

Proxy-based Protocol

This chapter introduces a proxy-based communication protocol that is capable of supporting efficient and reliable message delivery in mobile agent systems. The goal of the proxy-based solution is to reduce the communication overhead and solve the message loss problem in the home-server schemes.

To this end, the home-server schemes [2, 22, 36, 38, 47] are the most popular communication protocols for multi-agent systems because they are compatible with the current Internet Protocol. However, the triangular routing, central server constraints as well as the message loss problems in the home-server protocol affect the system performance.

The proxy-based scheme improves on the home-server schemes by incorporating an additional type of agent, a proxy agent, in the system. In this thesis, a domain is a group of connected computers that share a common central Directory Services Database that contains user account and security information. In each domain, at least one proxy agent is allocated to provide communication services to mobile agents in the domain. Messages sent to mobile agents pass through these proxies before they are dispatched over the network. The proxy agent decides where the message should go based on the most recent knowledge of the receiver.

There are several reasons to use a proxy agent as a message service center for the mobile agents in the domain. First, the proxy agent can obtain location information from incoming messages and share this information among a group of agents. Second, message passing between a proxy agent and the mobile agents within its control is fast because they are geographically close. Third, a proxy agent can buffer the messages for a mobile agent during the mobile agent's migration.

The proxy-based communication protocol is developed under the following assumptions:

- An agent server must be running on all the nodes where data can potentially reside.
- Every agent server has a unique address, which is accessible by all other nodes in the network.
- All agents in the system are trustworthy and available (security and authentication issues are not considered).
- All agents in the system are reliable (agent operation and migration failure are not considered).
- There is no loss or corruption in message passing (reliable delivery network).

In this chapter, the first section presents the system architecture and the functionality of the major components of the proposed protocol. The second section describes the system operation and algorithms in details. On top of the basic proxy-based protocol, a limited-forwarding algorithm is developed to further improve the protocol's performance. The last section verifies the reliability of the protocol.

3.1 System Architecture

Figure 3.1 provides a simple view of the system. All messages passing between two domains are under the control of proxy agents. If *Agent 1* in *Domain 1* wants to send a message to *Agent 2* in *Domain 2*, it must first deliver the message to *Proxy 1*. Then *Proxy 1* will forward the message to *Proxy 2*, and it is the responsibility of *Proxy 2* to send the message to the final receiver *Agent 2*. There is no direct communication between sender *Agent 1* and receiver *Agent 2*.

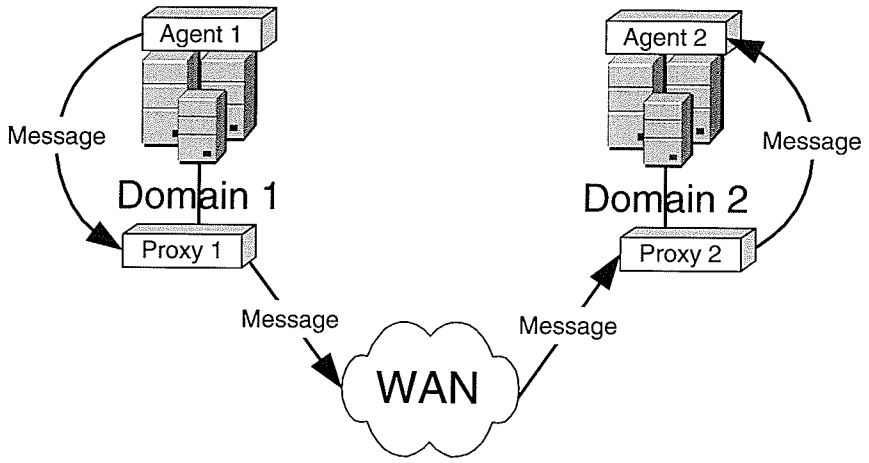


Figure 3.1: System Overview

The basic architecture of proxy-based system is shown in Figure 3.2. The system consists of several components that communicate with one another to provide location management and message delivery services. The roles and relationships of these components are defined as follow:

- *Master agent* (MA) is a stationary agent. This agent is responsible for creating a number of child agents and dispatching them to the respective destinations to perform some specific tasks. The master agent maintains the current location information of each of its child agents. The master agent

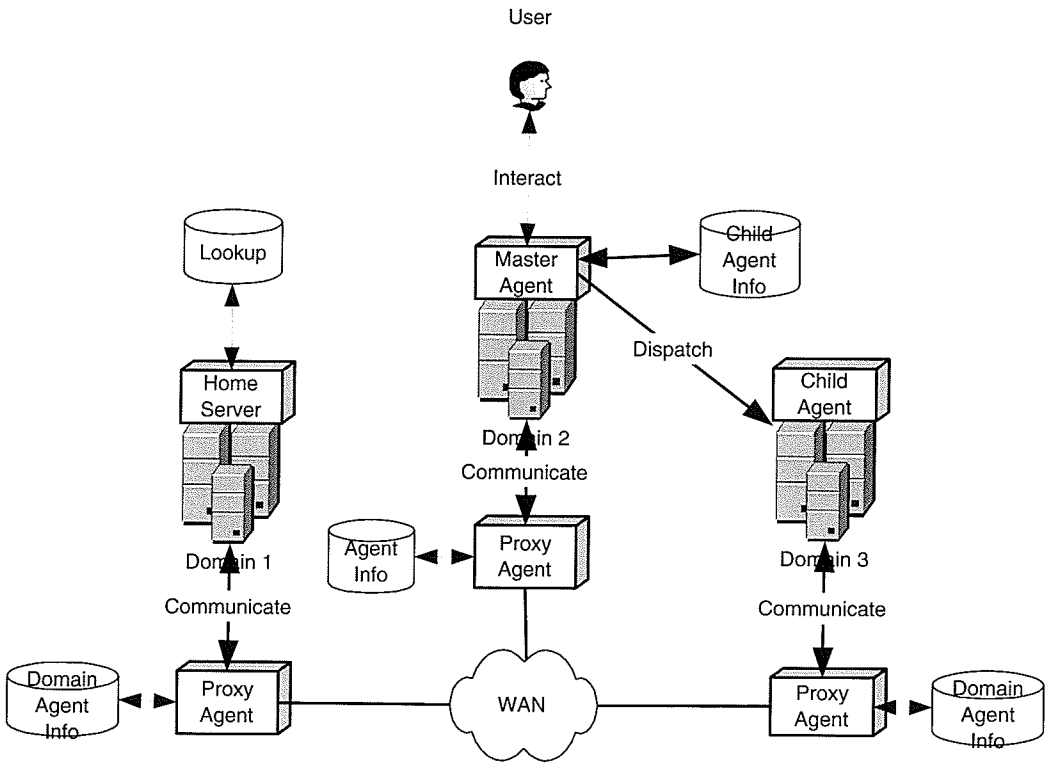


Figure 3.2: Detailed System Model

also provides mechanisms to interact with the user that issues the task request.

- *Child agent* (CA) is a mobile agent. This agent moves to the remote resource, performs tasks on behalf of the user and returns the results to the master agent. Each child agent has a unique ID to identify itself.

- *Home server* (HS) is a stationary agent. This agent provides the one-to-one mapping information between master agents and child agents, and routes a message to the corresponding master agent according to the child agent's ID.

- *Proxy agent* (PA) is a stationary agent. This agent collects agent location information from inbound messages, and builds up a local agent resource

table and a remote agent resource table gradually. The local agent resource table contains location information for all the mobile agents within the domain. The remote agent resource table includes some child agents' IDs and their corresponding proxy agent addresses. The proxy agent is responsible for handling messages for master agents and child agents. There is at least one proxy agent in each domain.

In the proxy-based communication protocol, every message contains the sender's ID, master agent's address and proxy agent's address. Except for these basic information fields, different types of messages contain different information fields. The functionality and field details for different messages are discussed in the next section.

3.2 System Operations

The following sections describe the processes of agent creation and dispatch, registration and deregistration, termination, migration, and message delivery.

3.2.1 Agent Creation and Dispatch

When a new child agent is created, it obtains a unique ID and the address of its master agent. After the creation, the master agent must inform the home server by sending a CREATION message to the home server. The CREATION message contains the ID of the child agent and the address of the master agent. The home server maintains a lookup table which records the mapping between child agents and master agent. Figure 3.3 shows an example of the lookup table. Every time the home server receives a CREATION message, it inserts a new entry in the lookup table and sends an ACK_CREATION message back to the master agent.

ChildId	MasterAddr
Child_01	128.1.2.10
Child_02	128.1.2.11
Child_03	128.1.2.12

Figure 3.3: Lookup Table of the Home Server

Upon receiving the ACK_CREATION message from the home server, the master agent prepares to dispatch the child agent by first assigning the destination address to the child agent. The master agent then inserts a new entry into its child agent information table and dispatches the child agent. Figure 3.9 shows an example of the child information table of the master agent. When the status of a child agent is set to INACTIVE, the master agent holds the messages for the child agent until the status is changed to ACTIVE. The status of a child agent is INACTIVE when the agent is first dispatched from its master agent. After the child agent reaches its destination, the proxy agent in the destination domain sends an UPDATE message to the master agent. The master agent then sets the child agent's status as ACTIVE in the child agent information table.

ChildId	ProxyAddr	Status
Child_01	216.239.36.27	ACTIVE
Child_02	216.109.118.40	ACTIVE
Child_03	202.108.36.78	INACTIVE

Figure 3.4: Child Information Table of the Master Agent

Figure 3.5 illustrates the whole process of agent creation and dispatch.

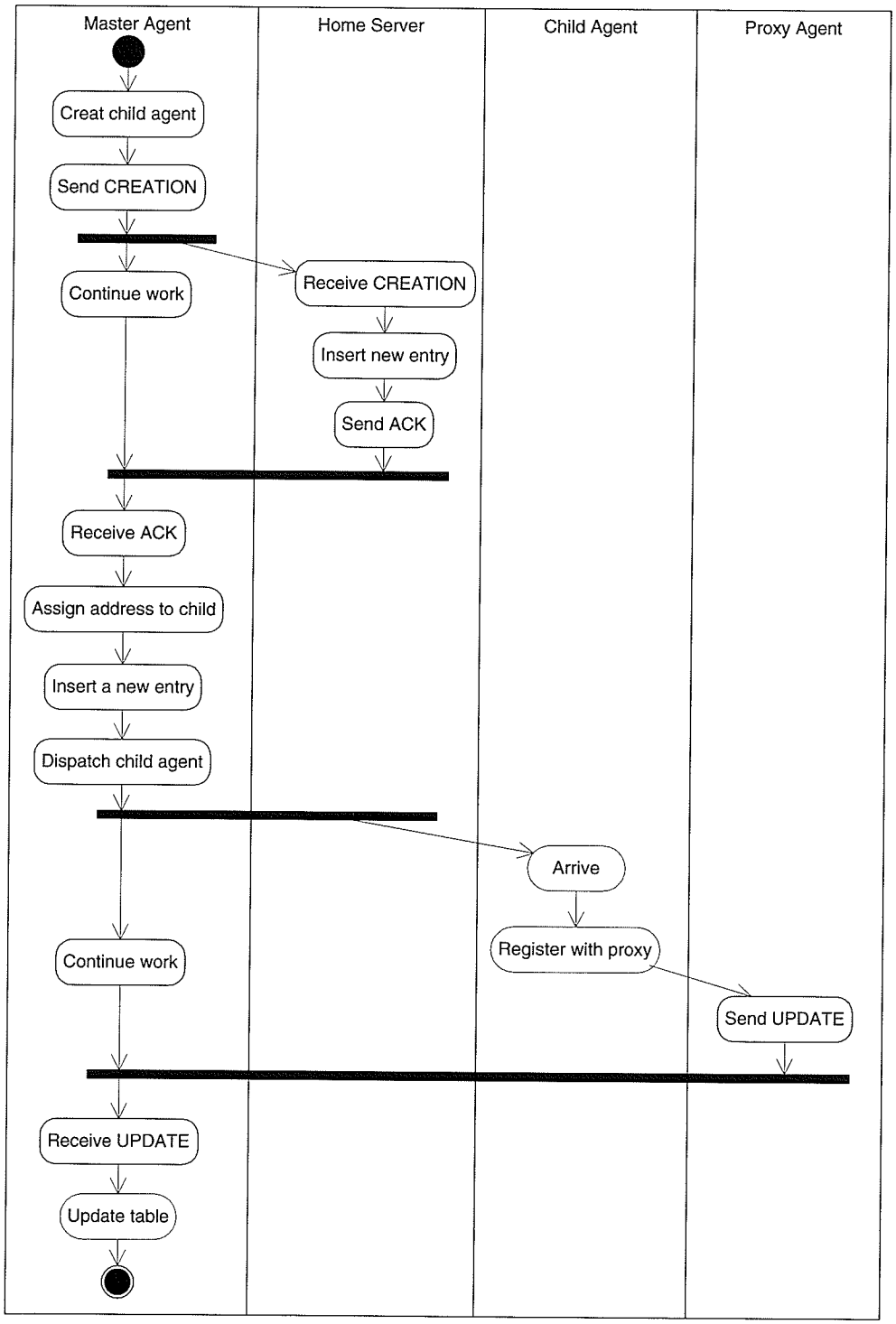


Figure 3.5: Activity Diagram for Agent Creation and Dispatch

3.2.2 Agent Registration and Deregistration

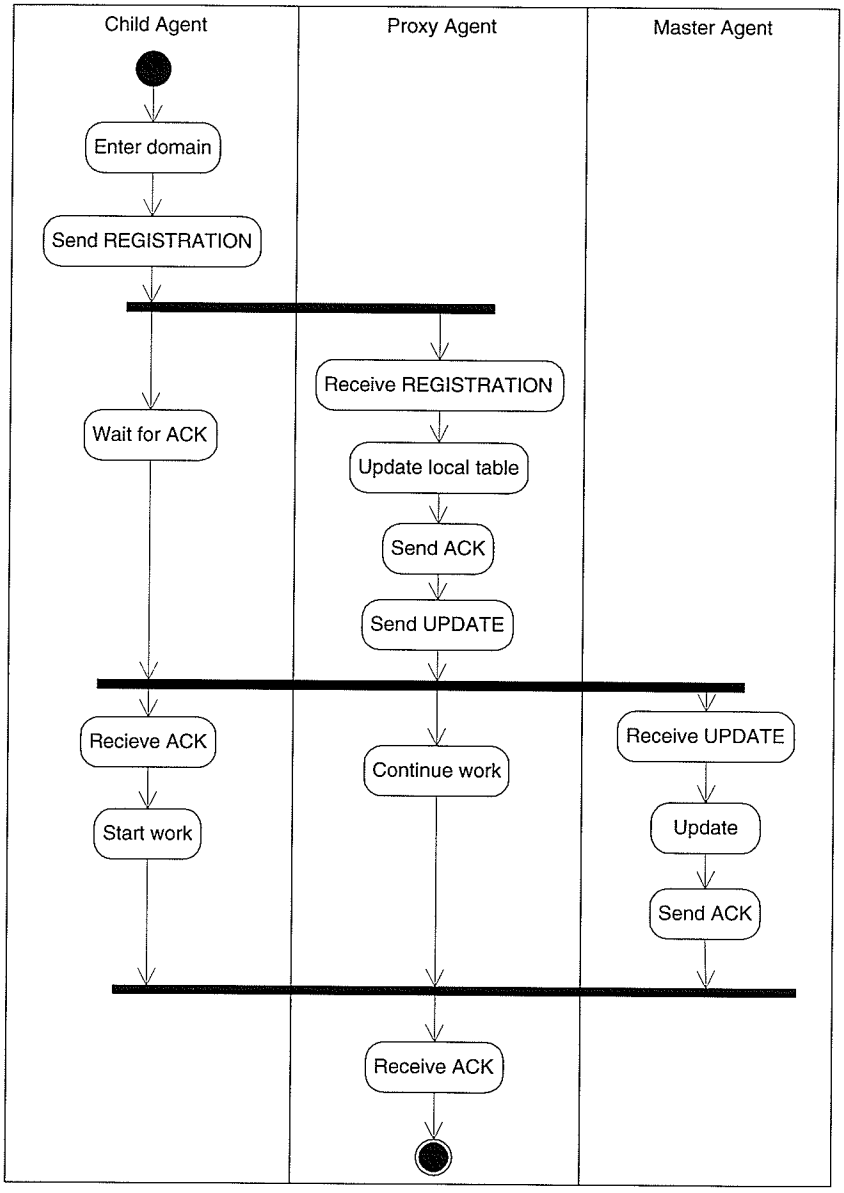


Figure 3.6: Activity Diagram for Agent Registration

Figure 3.6 illustrates the process of agent registration. Every time a child agent enters a new domain, it must register with the local proxy agent. Every proxy agent has a local agent table which records the location information of all the agents in

the domain. Figure 3.9 shows an example of the local agent table. When a child agent's status is set to MIGRATING, the proxy agent holds the incoming messages for the child agent until the status is set back to ACTIVE.

ChildId	Addr	Status
Child_01	216.239.36.27	ACTIVE
Child_11	216.239.36.28	ACTIVE
Child_21	216.239.36.29	MIGRATING

Figure 3.7: Local Agent Table of the Proxy Agent

The child agent first sends a REGISTRATION message to the proxy agent. The proxy agent inserts a new entry to its local agent table according to the message and sets the child agent's status to ACTIVE. The proxy agent then sends back an ACK_REGISTRATION message to the child agent. The proxy agent is also responsible for informing the child agent's master agent of the new location information by an UPDATE information. The master agent sends back an ACK_UPDATE message to the proxy agent after updating its child agent information table. The child agent then starts performing its tasks after receiving the ACK_REGISTRATION message.

Figure 3.8 illustrates the process of agent deregistration. During the registration process, the proxy agent can obtain the child agent's previous proxy agent ID from the REGISTRATION message. The current proxy agent sends a DEREGISTRATION message, which contains the child agent's ID and current proxy agent's address, to the child agent's previous proxy agent. The previous proxy agent deletes the child agent's record in its local agent table and inserts a new entry in the remote agent table. Figure 3.9 shows an example of the remote agent table. After updating both tables, the previous proxy agent sends back an ACK_DEREGISTRATION message. All the incoming messages for the child agent during its migration are also forwarded to its destination proxy agent.

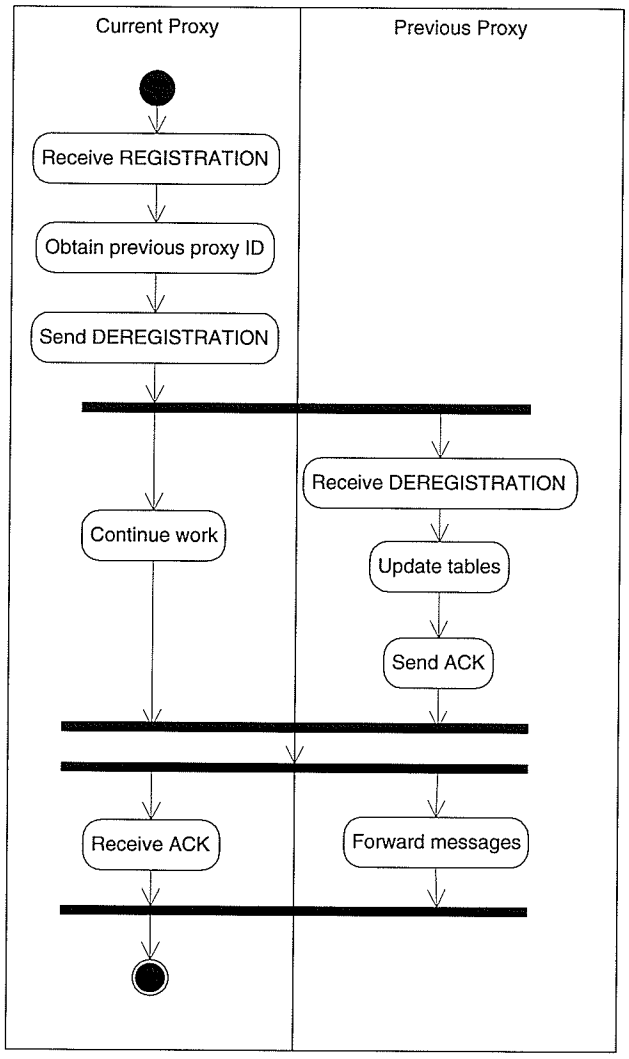


Figure 3.8: Activity Diagram for Agent Deregisteration

ChildId	ProxyAddr
Child_01	216.239.36.27
Child_31	240.29.55.90
Child_45	202.239.21.29

Figure 3.9: Remote Agent Table of the Proxy Agent

From the above description, it is obvious that part of the deregistration process is handled during agent registration. Figure 3.10 shows the registration algorithms

for the proxy agents.

```
/*  Variable
    localAgentTable: The local agent information table
    regMsg: The REGISTRATION message the proxy agent received
    MSG_UPDATE: Constant for update message type
    MSG_DEREGISTRATION: Constant for deregistration message type

    updateMsg: The UPDATE message
    deMsg: The DEREGISTRATION message
    */

        Registration() operation:

        InsertNewRec(regMsg, localAgentTable);
        updateMsg = creageMsg();
        updateMsg.type = MSG_UPDATE;
        updateMsg.endAgentId = regMsg.masterAgentId;
        sendMsg(updateMsg,regMsg.masterAgentId)

        IF regMsg.preProxyId Exists
            deMsg = createMsg();
            deMsg.type = MSG_DEREGISTRATION;
            deMsg.endAgentId = regMsg.preProxyId;
            sendMsg(deMsg,preProxyId);
        End IF

        End Registration()
```

Figure 3.10: Agent Registration Algorithm

3.2.3 Agent Termination

Figure 3.11 illustrates the process of agent termination. The child agent must inform its proxy agent before its termination by sending a TERMINATION message. The proxy agent forwards the TERMINATION message to the child agent's master agent and the home server. The home server deletes the child agent's record in the lookup table and sends an ACK_TERMINATION message to the proxy agent. The master agent deletes the child agent's record in the child agent information table and sends an ACK_TERMINATION message to the proxy agent. The proxy agent then deletes the child agent's record in its local agent table and

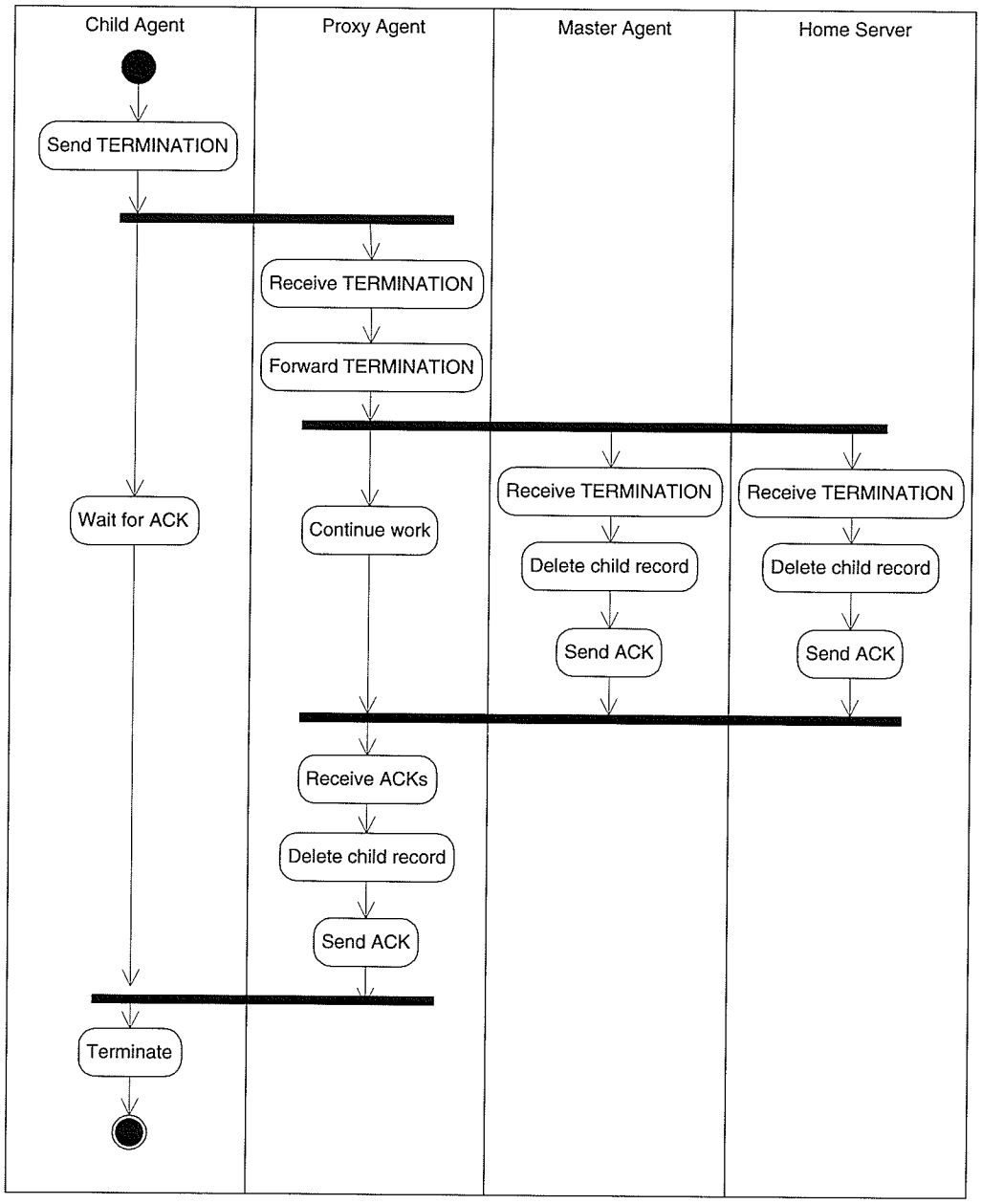


Figure 3.11: Activity Diagram for Agent Termination

sends an ACK_TERMINATION message to the child agents. On receiving the ACK_TERMINATION message, the child can terminate all its processes.

3.2.4 Agent Migration

Figure 3.12 illustrates the process of agent migration. The child agent must inform its proxy agent before its migration by sending a MIGRATION message. The proxy agent sets the child agent's status to MIGRATING in its local agent table and sends an ACK_MIGRATION message to the child agent. The child agent cannot start the migration until it receives the ACK_MIGRATION message. During the child agent's migration, the proxy agent puts all the child agent's incoming messages in a message queue.

If the child agent migrates within the same domain, it simply informs its proxy agent of its current location by an UPDATE message and waits for the ACK_UPDATE. The proxy agent updates the child agent's address and sets the status back to ACTIVE in its local agent table. The proxy agent then sends an ACK_UPDATE message and forwards all the holding messages to the child agent's new location. If a child agent moves to another domain, it must perform the registration with the proxy agent in the destination domain.

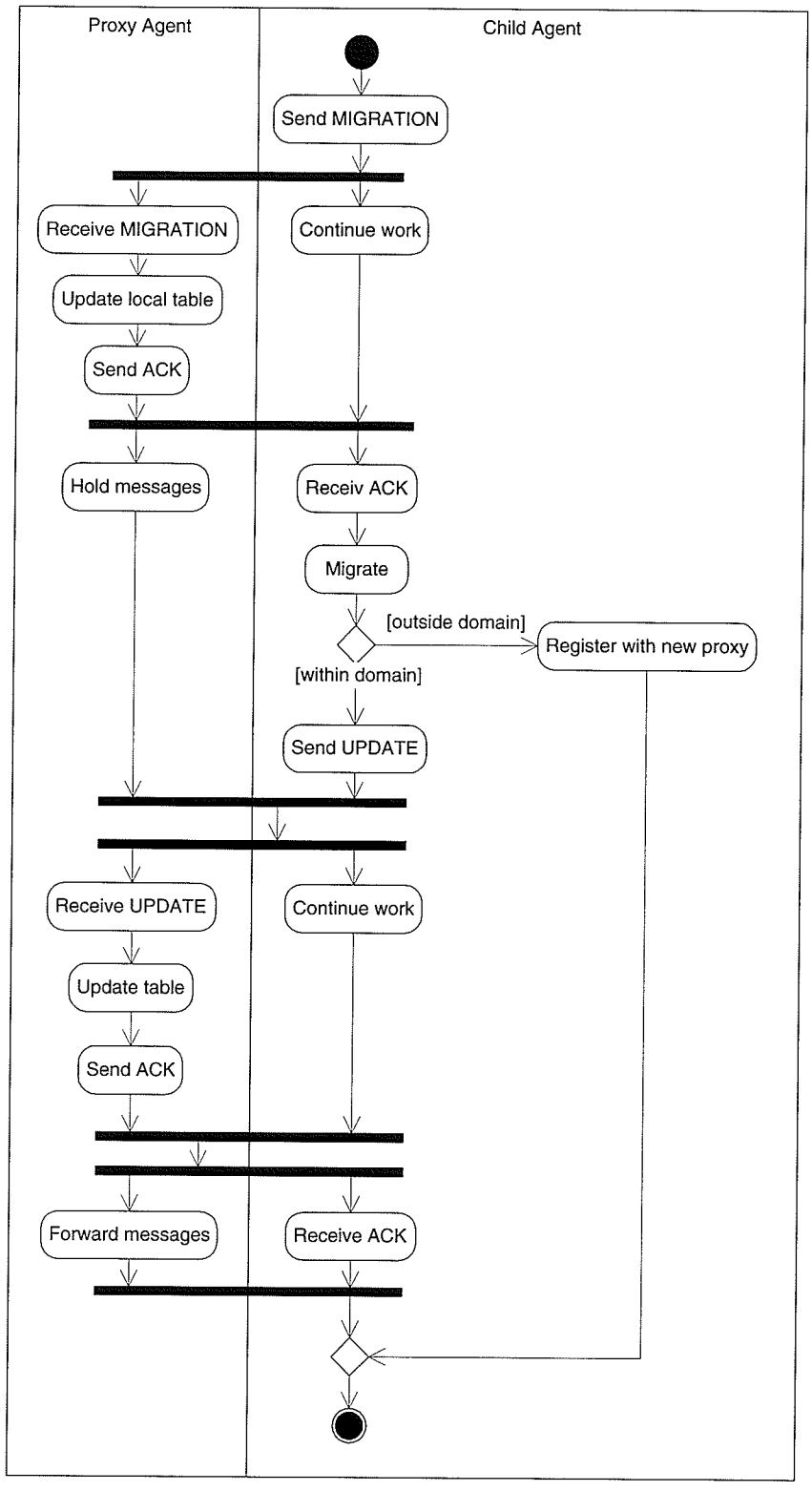


Figure 3.12: Activity Diagram for Agent Migration

3.2.5 Message Delivery

The basic idea of the proxy-based protocol is that all messages must be processed by proxy agents. When a proxy agent receives a message, it processes the message according to the message type. If the message is directed to another agent, the proxy agent processes the message differently for the message originating within or outside the domain.

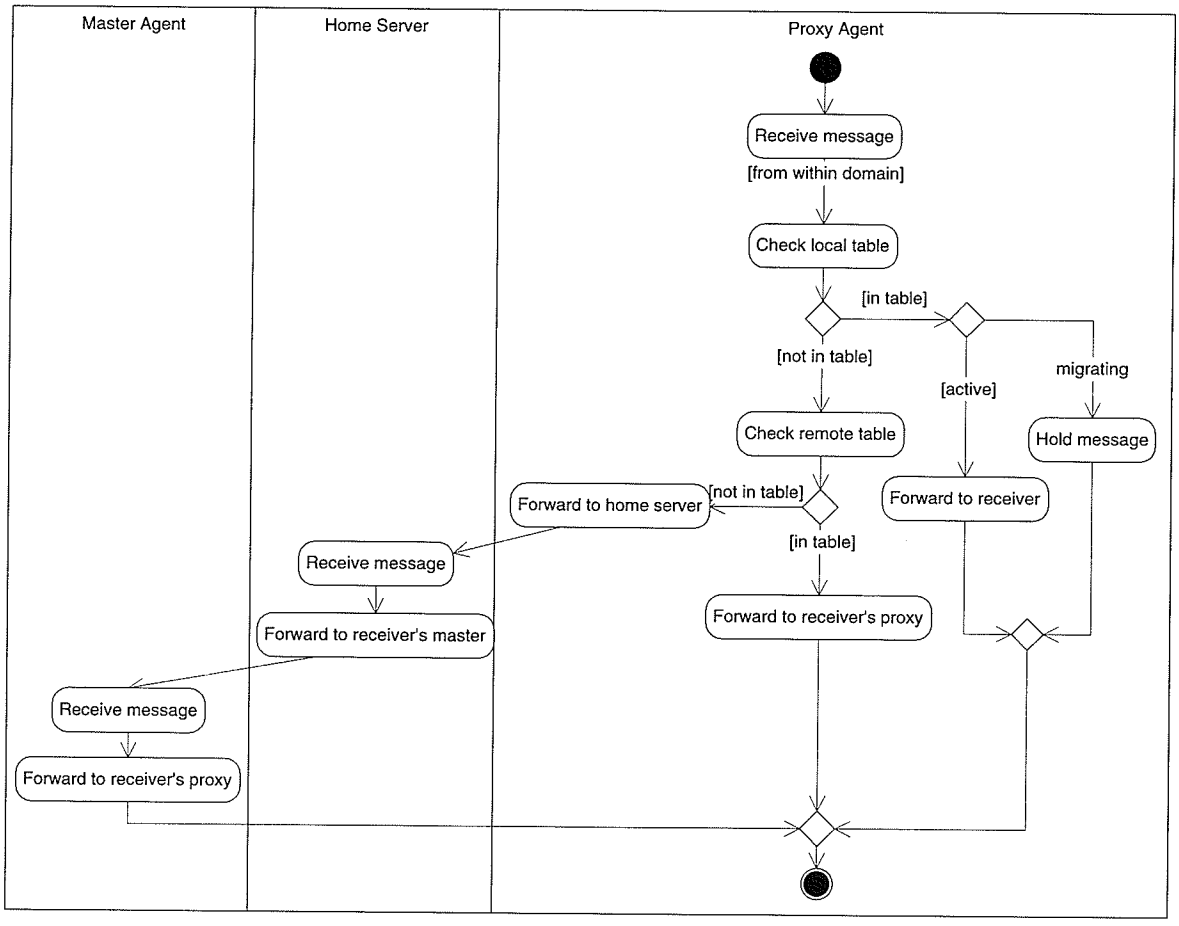


Figure 3.13: Activity Diagram for Message Delivery (from the same domain)

Figure 3.13 illustrates how a proxy agent processes a message coming from an agent within the same domain. The proxy agent first checks the local agent resource table. If the receiver is active in the domain, the proxy agent forwards

the message directly to the receiver. The proxy agent holds the message for the receiver when the receiver is migrating. If the receiver is not in the domain, the proxy agent searches for the receiver's ID in the remote agent resource table. The message is routed to the receiver's proxy agent if its information is in the table. If the receiver's ID is not in the remote agent resource table, the message is routed to the home server. The home server forwards the message to the receiver's master agent. The master agent then delivers the message to the receiver's current proxy agent.

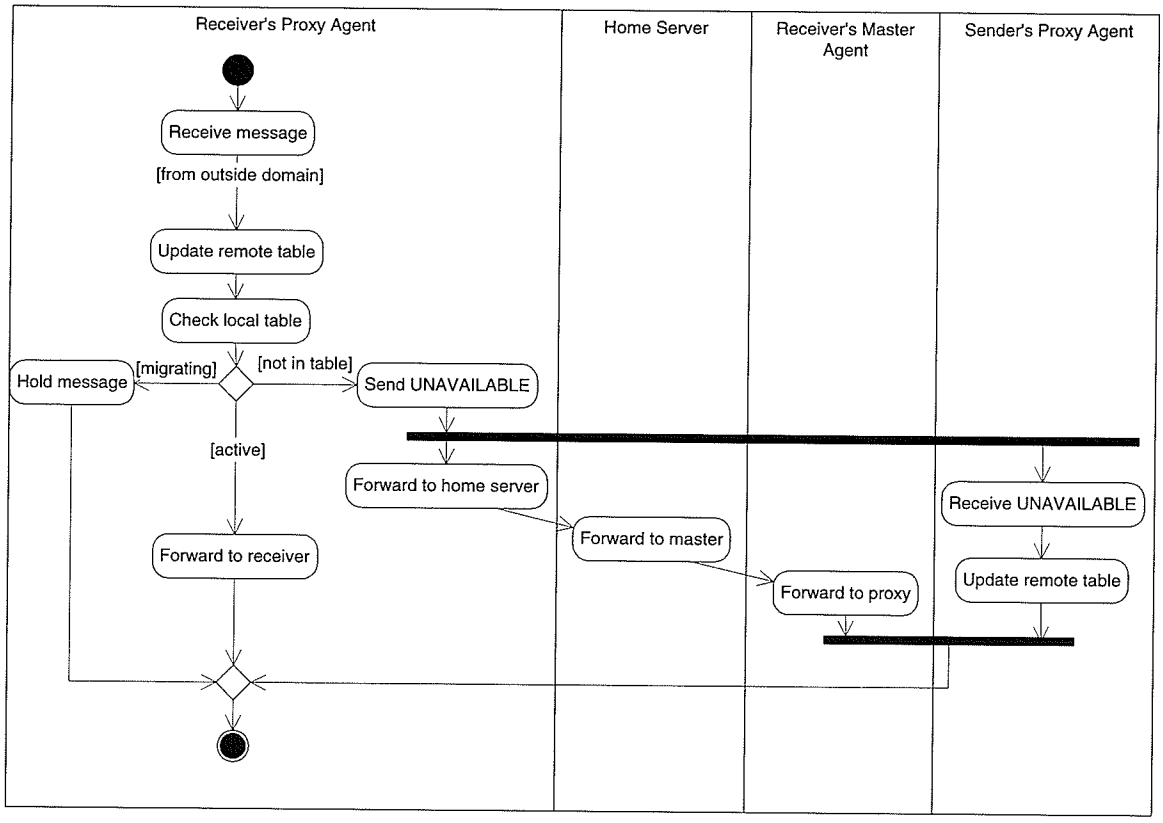


Figure 3.14: Activity Diagram for Message Delivery (from outside the domain)

Figure 3.14 illustrates how a proxy agent processes a message from an agent outside the domain. The proxy agent first adds/updates the sender's proxy agent's address in the remote agent resource table. The proxy agent then checks the local

agent resource table. If the receiver is active in the domain, the proxy agent forwards the message directly to the receiver. The proxy agent holds the message for the receiver when the receiver is migrating. If the receiver is not in the domain, an UNAVAILABLE message is sent to the sender's proxy agent. Upon receiving an UNAVAILABLE message, the sender's proxy agent deletes the outdated location information in its remote agent table. At the same time, the message is delivered to the home server. The home server forwards the message to the receiver's master agent and the master agent delivers the message to the receiver's proxy agent.

3.2.6 Message

In the proxy-based protocol, for the communication purpose, agents provide specific services according to the messages they receive. This section summarizes the description of different message types and their relation to the agents.

Table 3.1 outlines the functionality of different message types

Message Type	Description
COMMUNICATION	Normal communication message from agent to agent
CREATION	A master agent informs the home server of the creation of a new child
ACK_CREATION	The home server acknowledges the new child agent's creation.
REGISTRATION	A child agent registers with the local proxy agent upon entering a new domain.
ACK_REGISTRATION	A proxy agent acknowledges the newly entered child agent.
UPDATE	A child agent updates its location with the proxy agent. Or a proxy agent updates its newly registered child agent's location with the child agent's master agent.
ACK_UPDATE	A proxy agent or master agent acknowledges a child agent's location update.
DEREGISTRATION	A proxy agent deregisters its newly registered child agent with the child agent's previous proxy agent.
ACK_DEREGISTRATION	A child agent's previous proxy agent acknowledges the child agent's deregistration.
MIGRATION	A child agent notifies its proxy agent of an upcoming migration.
ACK_MIGRATION	A proxy agent acknowledges a child agent's migration.
TERMINATION	A child agent notifies its proxy agent, master agent and the home server of the upcoming termination.
ACK_TERMINATION	A proxy agent, master agent or home server acknowledges a child agent's termination.
UNAVAILABLE	A proxy agent informs another proxy agent that the intended receiver is no long within the domain.

Table 3.1: Message Description

Table 3.2 shows the relationship between agents and different message types.

Message Type	Sender	Receiver
COMMUNICATION	Agent	Agent
CREATION	Master	Home Server
ACK_CREATION	Home Server	Master
REGISTRATION	Child	Proxy
ACK_REGISTRATION	Proxy	Child
UPDATE	Child or Proxy	Proxy or Master
ACK_UPDATE	Proxy or Master	Child or Proxy
DEREGISTRATION	Proxy	Proxy
ACK_DEREGISTRATION	Proxy	Proxy
MIGRATION	Child	Proxy
ACK_MIGRATION	Proxy	Child
TERMINATION	Child or Proxy	Proxy or Master or Home Server
ACK_TERMINATION	Proxy or Master or Home Server	Child or Proxy
UNAVAILABLE	Proxy	Proxy

Table 3.2: Relationship between Agent and Message Type

3.2.7 Information Table

In the proxy-based protocol, the stationary agents, such as the home server, master agents and proxy agents, use different tables to record the location information of other agents. Agents perform various actions on these information tables after receiving different messages during system operations. This section summarizes the relationships between information tables and messages.

Table 3.3 shows the relationships between the lookup table of the home server and different messages.

System Operation	Message Type	Table Action
Agent Creation	CREATION	Insert (ID,ADDR)
Agent Termination	TERMINATION	Delete (ID=ChildId)
Message Delivery	COMMUNICATION	Search (ID=ReceiverID)

Table 3.3: Relationships between Lookup Table and Messages

Table 3.4 shows the relationships between the child agent information table of the master agent and different messages.

System Operation	Message Type	Table Action
Agent Creation	ACK_CREATION	Insert (ID,ADDR,STATUS=INACTIVE)
Agent Registration	UPDATE	Update (ADDR,STATUS=ACTIVE)
Agent Termination	TERMINATION	Delete(ID=ChildId)
Message Delivery	COMMUNICATION	Search(ID=ReceiverID)

Table 3.4: Relationships between Child Agent Information Table and Messages

Table 3.5 shows the relationships between the local agent table of the proxy agent and different messages.

System Operation	Message Type	Table Action
Agent Registration	REGISTRATION	Insert (ID,ADDR,STATUS=ACTIVE)
Agent Deregistration	DEREGISTRATION	Delete (ID=ChildId)
Agent Termination	ACK_TERMINATION	Delete (ID=ChildId)
Agent Migration	MIGRATION	Update (STATUS=MIGRATING)
Agent Migration	UPDATE	Update (ADDR, STATUS=ACTIVE)
Message Delivery	COMMUNICATION	Search (ID=ReceiverID)

Table 3.5: Relationships between Local Agent Table and Messages

Table 3.5 shows the relationships between the remote agent table of the proxy agent and different messages.

System Operation	Message Type	Table Action
Agent Deregistration	DEREGISTRATION	Insert (ID,ADDR)
Message Delivery	COMMUNICATION	Search (ID=ReceiverID)
Message Delivery (from outside the domain)	COMMUNICATION	Insert (ID,ADDR) or Update(ADDR)
Message Delivery	UNAVAILABLE	Delete (ID=ReceiverId)

Table 3.6: Relationships between Remote Agent Table and Messages

3.3 The Limited-Forwarding Scheme

The efficiency of the proxy-based scheme depends on the message exchange rate and the mobile agents' migration frequency. The scheme may perform poorly in a highly dynamic system with relative low frequency of interagent communication because the cache information is likely to be obsolete when it is needed. To improve the performance of the system, the forwarding-pointer strategy described in section 2.3.3 can be integrated with the proxy-based protocol.

In the basic proxy-based scheme, if a proxy agent receives a message from outside the domain and the message is directed to a mobile agent that is no longer in the domain, the proxy agent sends an UNAVAILABLE message to the sender's proxy agent and redirects the incoming message to the home server. According to the home-server protocol discussed in section 2.3.1, the message has to be delivered at least three times before it can reach the final proxy agent. In some cases, simply forwarding a message is cheaper than using the home-server scheme.

However, as noted before, the forwarding-pointer strategy has one serious drawback: the chasing problem. A message can follow an agent forever if the agent migrates frequently. In the proxy-based protocol, if a message is forwarded more than three times, the communication cost is likely higher than the simple home-server protocol since the home-server protocol delivers a message by forwarding

the message three times in most cases. To solve this problem, an algorithm that can guarantee that the cost of using forwarding-pointer will be lower than the home-server protocol in most cases is developed.

The notion of limited-forwarding algorithm has been used for PCS (personal communications services) [36]. The idea is when a cache miss occurs, the message will be forwarded one step. If the cache miss occurs again, the IS-41 (the equivalent of home-server in PCS) protocol is used to deliver the message. The drawback of the one-step forwarding algorithm is the location information in the proxy agent's remote agent source table may be obsolete due to the mobility of agents. If the obsolete cache information is used to forward a message, the next proxy agent still has to use the home-server scheme to deliver the message. In such a case, the overhead of using one-step forwarding strategy to deliver the message is higher than the home-server scheme. Therefore, it is desirable to use some strategies to predict whether the cache is valid to improve the cache hit ratio.

The idea is, a mobile agent records its average inter-domain migration interval M, and embeds this information into the outgoing message. The proxy agent at the receiving end extracts the M from the incoming message. As a result, the proxy agent can determine whether to forward the message to the cached location or to the home server based on the receiver's migration frequency. In order to use the limited-forwarding scheme, a few extra bytes are needed in a message. In addition, the record in the proxy agent's remote agent resource table must include two fields: FT the time the proxy agent receives the incoming message, and M the average inter-domain migration interval of the sender. Figure 3.15 shows the algorithm of the calculation of average inter-domain migration interval for a child agent. The calculation occurs after the child agent moves to a new location.

To make the algorithm more flexible, a forwarding factor FF is defined to control the average maximum steps a message can be forwarded in the system.

```
/*  Variable
    meanMoveTm(=M): Average cross-domain migration interval, it is assigned a
    maximum value during agent creation.
    interDomainMv: Number of inter domain migrations(base figure for M), initial value is 0
    curMoveTm: Current move time, initial value is 0
    lastMoveTm: Last move time, initial value is 0

    totalTm: The total time since the child agent left its master agent
    */

        Recalculate() operation:

            curMoveTm = Time();
            IF (the agent moves to a new domain)
                totalTm = interDomainMv*meanMoveTm+(curMoveTm - lastMoveTm);
                meanMoveTm = totalTm/(interDomainMv+1);
                interDomainMv++;
                lastMoveTm = curMoveTm;
                Register();
            End IF
        End Recalculate()
```

Figure 3.15: Recalculation Algorithm

Generally, the forwarding factor should not be larger than 3 because the home-server protocol can deliver a message by forwarding the message three times. Let CT represent the current time and FT represent the time the proxy agent receives the incoming message, the proxy agent compares the $(CT - FT)$ to $FF * M$ before it delivers the message. If $(CT - FT) \leq FF * M$, the cache information is considered valid and the message is forwarded to the receiver's proxy agent. If $(CT - FT) > FF * M$, the cache information is considered obsolete. In such a case, the message is directly delivered to the home server. Using this algorithm, most of the message forwarding can be limited to within predefined steps, and therefore, prevents message chasing and long chain problems in the forwarding-pointer algorithm [21, 39, 52]. Figure 3.16 illustrates the process of message delivery in proxy-based protocol with the limited-forwarding algorithm.

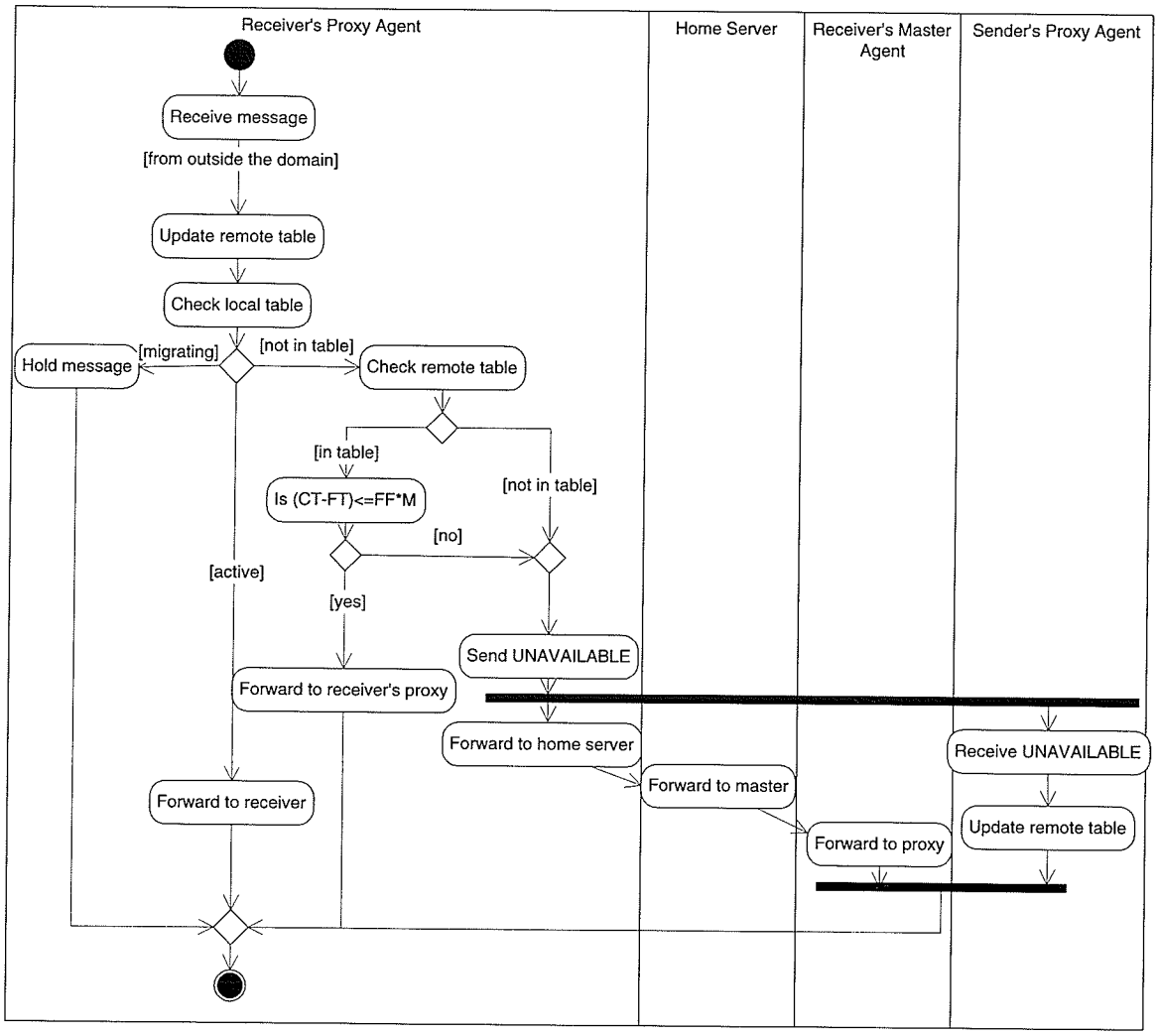


Figure 3.16: Activity Diagram for Message Delivery (from outside the domain)

3.4 Protocol Reliability

One of the goals of this thesis is to develop a reliable protocol for mobile agent communication. This section presents the mechanisms in the proxy-based protocol that can guarantee reliable message delivery. As discussed in section 2.2.3 , the reliability of a communication protocol for multi-agent systems involves two issues: (1) the vulnerability of the protocol to the failures in the distributed environment;

and (2) message losses or chasing problems. A reliable communication protocol should address both issues to guarantee that a message can be delivered successfully to the receiver in any situation.

For the first issue, the proxy-based protocol is a little more sensitive to the failure in the network than the home-server protocol. In addition to home-server and master agents, the failure of proxy agents also causes communication problems in the system. However, since the number of proxy agents is much smaller than the total number of network nodes, the problem can be solved by using replication to backup information for the proxy agents.

For the second issue, the home-server protocol cannot guarantee the delivery of messages to the receiving agent even in a fault-free network environment. If a mobile agent migrates during message forwarding, the message would be lost forever. The proxy-based protocol solves this problem by using synchronization messages. Let's consider the worst case: mobile agent A sends a message m to mobile agent B, agent B migrates to another location during the message delivery. However, before B's migration, B must inform its current proxy agent PB of its intended movement and waits for the ACK_MIGRATION message. Once the proxy agent receives the MIGRATION message, it sets B's status to MOVING and will not forward any message to B. There are three possible situations for m when it arrives at PB: (1) B is moving; (2) B has reached a new location and PB holds valid address information for B's new location; (3) B has reached a new location and PB no long holds cache information for B. In the first case, PB holds m until PB receives UPDATE or DEREGISTRATION message from B. Then PB can forward m to B's new location. In the second situation, PB simply forwards m to B's new location. If B is in a new domain, the new proxy agent processes m based on the same principles. In the third situation, PB employs the home-server protocol and directs m to the home server. Therefore, no matter which situation

holds, the proxy-based protocol can deliver a message to its destination agent in a bounded number of hops.

Chapter 4

Simulation Model

This chapter describes a simulation model that simulates a proxy-based protocol and a home-server protocol. The purpose of the simulation is to compare the difference in communication costs of these two protocols. The first section explains why simulations, instead of a prototype, were used to evaluate the performance of the proposed protocol. The supported framework for the simulation model, SSJ, and the concept of discrete-event stochastic simulation are also introduced. The development of the simulation model is described in depth. After that, the verification and validation of the simulation model are discussed. The implementation details of the simulation model are presented in the last section.

4.1 Introduction

Many mobile agent platforms, such as Aglets [1, 30], ffMain [37], Emerald [27] and Voyager [5], have been proposed in recent years. However, these agent platforms all use one specific protocol for agent communication. This makes the implementation of the proposed scheme infeasible on these agent platforms. The only way to implement the proxy-based protocol is to modify one of the agent platforms or create a new agent platform, both of which are very time-consuming. In addition,

due to access limitations, the implementation can only take place in a LAN setting, which makes the measurement of large scale wide-area network communication impossible. As a consequence, the result of such an implementation would be far from accurate. Furthermore, implementation cannot provide easy insight into the effects of various parameters and their interactions. First, changing system configuration for every alternative is costly in terms of time and effort. Second, it is impossible to create identical events for every alternative in the real world. Therefore, it is hard to tell whether a performance change is a result of some random effects in the environment or due to the particular configuration.

On the other hand, simulation provides a more flexible and accurate technique for analyzing the performance of the proxy-based communication protocol for mobile agents. First of all, it takes less time and effort to construct a simulation model than to implement the protocol on an agent platform. Second, a simulation model allows the performance of the proposed protocol to be measured under a wider variety of workloads, network diameters, and behavior patterns of mobile agents. Third, simulation is preferred over implementation due to its ease of changing configurations. Fourth, a simulation model using control variables can eliminate the random effect in the environment in comparison between runs involving different configurations.

Discrete-event stochastic simulation methods [8, 32] are used to developed the simulation model for the proposed protocol because it is hard to obtain the real trace of a multi-agent system. The Stochastic Simulation in Java (SSJ) [33] framework is used to create a home-server and a proxy-based protocol simulation model.

4.2 The SSJ Framework

SSJ is a general-purpose framework for simulation programming. It is implemented as a library of classes in the Java programming language. These classes provide tools for generating random numbers, collecting statistics, managing a simulation clock and event list, synchronizing concurrent processes, etc. SSJ is primarily designed for discrete-event stochastic simulations, but it also supports continuous simulation and arbitrary mixtures of these simulations.

For the discrete-event stochastic simulations, SSJ supports both event-oriented and process-oriented programming. An event is an incident which occurs instantaneously at a point in time and changes the state of the system. A subclass of class *Event* is defined for different types of events that can occur in the simulation, e.g., message delivery and agent migration. Each event is created with a scheduled time of occurrence and is inserted into the event list automatically. An event is executed when the simulation clock reaches the event's pre-schedule time. The *Sim* class is responsible for maintaining the simulation clock and the event list.

Process-oriented programming provides higher-level tools for discrete-event stochastic simulations than event-oriented programming. A process is an active object that has methods describing the succession of states of an object and its effect on the system state over a period of time. Processes are typically implemented as threads that can be suspended and resumed to represent the stop-start nature of the work of a process [32]. Processes are particularly suitable for describing autonomous entities, such as agent or robots, within an environment. However, event-oriented programming provides better performance because it avoids the process-synchronization overhead.

4.3 Simulation Model

4.3.1 The Performance Metrics

The goal of the simulation is to provide estimation for the efficiency of the proposed protocol. Since the communication overhead in a multi-agent system includes both location updates and message delivery, the simulation should provide performance metrics for both. Interest is not in absolute performance value, but rather in relative difference in performance of the proxy-based protocol and the home-server protocol.

The performance metrics of the simulation model are defined as follow:

- **Number of the Communication Messages:** the total number of message exchanges among mobile agents during the simulation time. This metric is used to verify whether the home-server and proxy-based protocol model generate the same number of communication message.

- **Average Delivery Time of Communication Messages:** the interval between the time the sender sends a message and the time the receiver receives the message. The execution time of a message is ignored because the execution time is negligible compare to the transit time.

- **Number of the Update Messages:** the total number of update messages. The results from home-server and proxy-based protocol are expected to be different due to their respective update policies.

- **Average Delivery Time of Update Messages:** the interval between the time a sender sends out the message and the time the receiver receives the message. The update message is always sent to a stationary agent, which does not involve location tracking, therefore, the results from both protocols are expected to be similar.

- **Total Cost:** the sum of communication cost and the update cost. Communication cost is equal to the number of communication messages times the average delivery time of communication messages while the the cost of update is equal to the number of update messages times the average delivery time of update messages.

4.3.2 The Control Parameters

The simulation parameters include the number of computers, the number of domains, the number of master agents, the number of child agents, average time interval between message delivery, average time interval between agent migrations, probability of an agent move within the same domain, forwarding factors, and simulation stop time.

- **Number of Computers:** The number of computers represents the total number of computers that the child agents can visit in the network. In this simulation, the value of this parameter ranges from 100 to $1,000,000$.

- **Number of Domains:** The number of domains represents the total number of domains in the network. In this simulation, the value of this parameter ranges from 2 to $1,000$.

- **Number of Master Agents:** The number of master agents represents the total number of master agents in the network. In this simulation, the value of this parameter ranges from 2 to $1,000$.

- **Number of Child Agents:** The number of child agents represents the total number child agents in the network. In this simulation, the value of this parameter ranges from 10 to $3,000$.

- **Average Time Interval between Message Delivery:** The average time

interval between message delivery represents the average message generation rate of each child agent. In this simulation, the value of this parameter ranges from 5ms to 3,600,000ms (1 hour).

- **Average Time Interval between Agent Migrations:** The average time interval between agent migrations represents the average migration rate of each child agent. In this simulation, the value of this parameter ranges from 60,000ms (1 minute) to 21,600,600 (6 hours).

- **Probability of an Agent Move within the Same Domain:** The probability of an agent move within the same domain defines the average number of the consecutive moves of an agent in the same domain before it moves to another domain. Let p represent the probability, $1-p$ can be viewed as the parameter for a geometric distribution with state space $\{1, 2, \ldots\}$. Therefore, the mean number of consecutive moves in the same domain for a child agent is $1/(1-p)$.

- **Forwarding Factor:** The forwarding factor is used in the limited-forwarding algorithm to define the cache available time. In this simulation, the value of this parameter ranges from 0 to 3. When the forwarding factor is 0, there is no forwarding strategy in the system, therefore, the system is the proxy-based protocol without the limited-forwarding algorithm.

- **Simulation Stop Time:** The simulation stop time represents the length of time that the simulation runs. To get an accurate result, the value of this parameter should be much larger than the interval time between message exchange and agent migration.

4.3.3 Conceptual Model

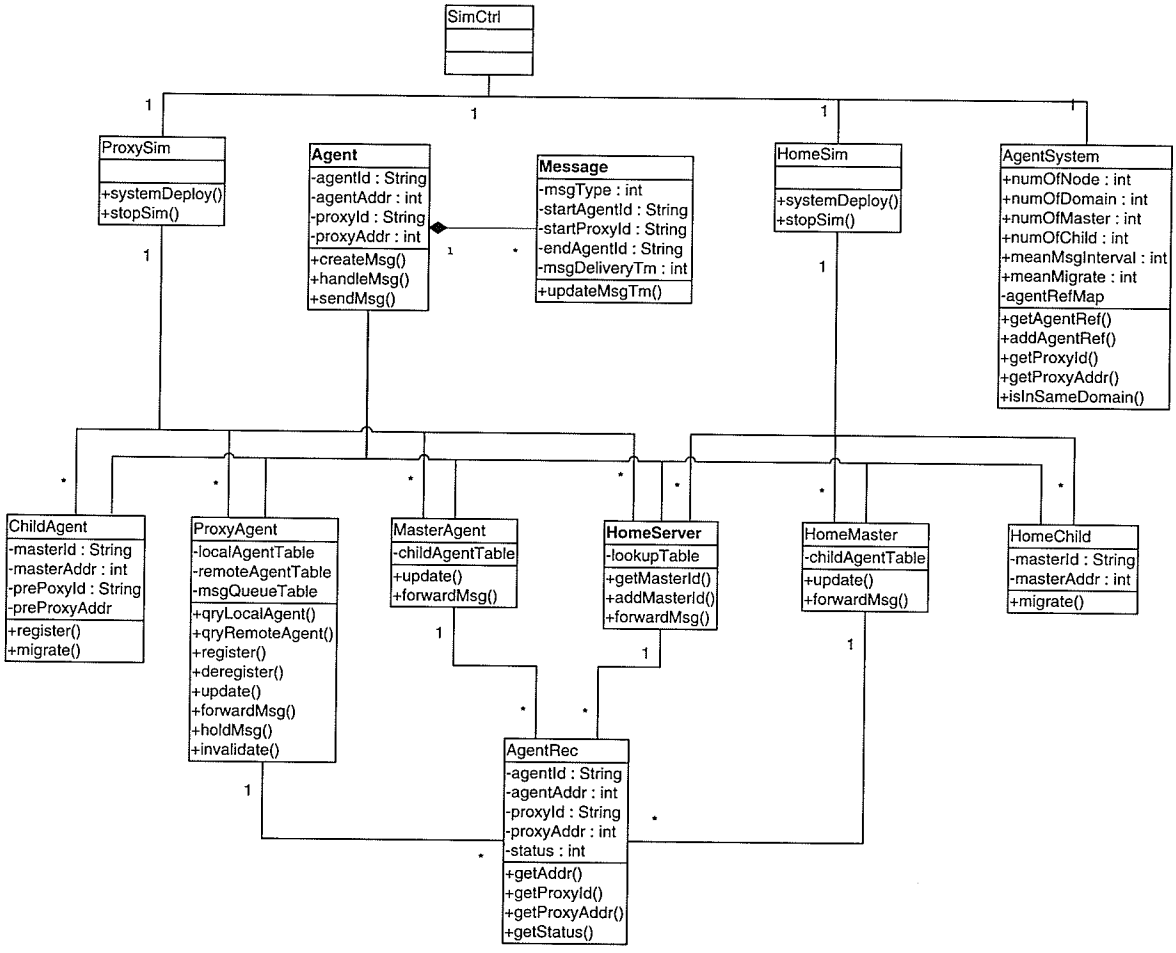


Figure 4.1: Class Diagram for Simulation Model

Figure 4.1 represents the class model for the simulation. At the root is the *SimCtrl* class where the user specifies the control parameters and receives the result for the simulation run. For each simulation run, a home-server and a proxy-based protocol are simulated under the same control parameters. *HomeSim* and *ProxySim* classes represent the simulation control for these two protocols, respectively. The *Agent* class is the generic class for all different types of agents in the system. Each agent has a unique identifier and current address. An agent can

create certain types of messages, send messages to other agents, and process messages from other agents. Although the interfaces are similar, the master agent and child agents in home-server and proxy-based protocol have quite different actions. Therefore, *HomeMaster* class and *HomeChild* class are used to described these agents in the home-server protocol while *MasterAgent* and *ChildAgent* are used in the proxy-based protocol. Since the home server performs the same tasks in both protocols, the *HomeServer* class can be used in both simulations. The *ProxyAgent* can only be used in the proxy-based protocol. The main task of a proxy agent is to maintain the local agent table, the remote agent table and hold message table, and to process and forward messages according to the table information and message type. The *AgentRec* class describes the structure of a record that can be used by master and proxy agents.

Figure 4.2 shows the control flow of the simulation. After the user specifies the parameters, the system initiates the proxy-based simulation. A multi-agent system is constructed and the initial events are defined during the child agent's creation. When the simulation starts, more events are added to the event list. The *Sim* class manages the event list and processes events when the simulation clock reach the schedule time. The simulation stops when the clock reaches the pre-defined stop time. The system records the result for the proxy-based protocol simulation and initiates the home-server simulation. The simulation goes through the same procedure and at the end displays the results of both protocols to the user. The user can either change the parameters and start another simulation or exit the system.

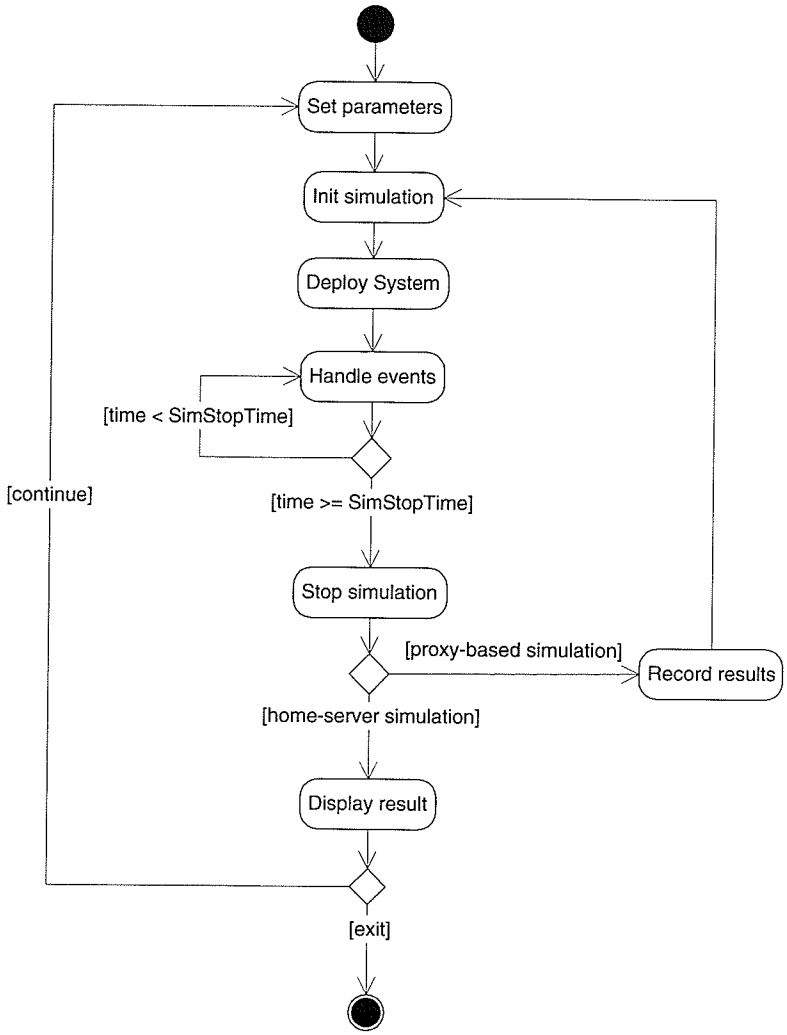


Figure 4.2: Simulation Control Flow

4.3.4 Initialization of the Simulation

Since the goal the simulation is to measure the communication cost, some irrelevant activities such as agent creation and termination are not simulated in the model. To make sure the performance metrics are collected after the system reach a steady state, one tenth of the total simulation time is defined as the run-up period of the simulation. All agent objects are stored in a hash map *agentRefMap*, the key is

the agent's ID. An agent object is added to the *agentRefMap* upon its creation. The underlying network model is constructed according to the number of nodes and the number of domains specified by the user. The nodes are evenly distributed over the whole network. For example, if the total number of nodes is 100 and the number of domains is 10, then node0 to node9 are in domain0, and node10 to node19 are in domain1, etc.

For each system, node0 is the default residing node for the home server. The number of proxy agents is equal to the number of domains, which means for every domain there is exactly one proxy agent. The first node in the domain is the default residing node for the proxy agent. The user can define the number of master agents and child agents in the system. There is no limitation on which node a master agent or a child agent can reside at. For the initial deployment, master agents and child agents are randomly distributed across the network. The master agent for a specific child agent is also randomly chosen. The child agent must register with the assigned master agent and proxy agent upon its creation. Figure 4.3 shows the algorithm for the system deployment of the proxy-based protocol.

In the simulation model, all messages are considered to have equal length; thus, the speed of delivery is the same under the same network conditions. A domain is viewed as a local area network. Generally, the end-to-end delay in a local area network ranges from $1ms$ to $10ms$. Here, the average value $5ms$ is used as the intra-domain message delivery time. The inter-domain communication is considered as network traffic in a wide area network. The inter-domain message delivery time is defined as $150ms$, which is the maximum one-way delay acceptable for IP telephony applications [3].

```
/*  Variable
    numOfNode: The number of nodes in the network
    numOfDomain: The number of domain in the network
    numOfMaster: The number of master agents in the system
    numOfChild: The number of child agents in the system


    numPerDomain: The number of nodes in each domain
    HomeServer: Home server for the system
    Proxy_i: Proxy agent in domain i
    Master_i: The #i master agent
    Child_i: The #i child agent
    NewRec: A new record in table
    agentId, agentAddr: Attributes of Agents
    */
        SystemDeploy() operations:

        numPerDomain = ceil(numOfNode/numOfDomain);
        HomeServer = NEW HomeServer();
        HomeServer.agentId = "HOME";
        Homeserver.agentAddr = 0;

        FOR(i = 0;i < numOfDomain; i++)
            Proxy_i = NEW ProxyAgent();
            Proxy_i.agentId = "Proxy"+i;
            Proxy_i.agentAddr = i*numPerDomain+1;
        END{FOR}

        FOR(i = 0; i < numOfMaster; i++)
            Master_i = NEW MasterAgent();
            Master_i.agentId = "Master"+i;
            Master_i.agentAddr = genUniformRan(1,numOfNode);
        END{FOR}

        FOR(i = 0; i <, numOfChild; i++
            Child_i = NEW ChildAgent();
            Child_i.agentId = "Child"+i;
            Child_i.agentAddr = genUniformRan(1,numOfNode);
            Child_i.proxyId = "Proxy"+getDomainNum(Child_i.agentAddr);
            Child_i.proxyAddr = getDomainNum(Child_i.agentAddr)*numPerDomain+1;
            Child_i.masterId = "Master"+genUniformRan(0,numOfMaster);
            Child_i.masterAddr = getAgentRef(Child_i.masterId).agentAddr;
            HomeServer.addMasterId(Child_i.agentId, Child_i.masterId);
            NewRec = NEW AgentRec();
            NewRec.agentId = Child_i.agentId;
            NewRec.agentAddr = Child_i.agentAddr;
            NewRec.masterId = Child_i.masterId;
            NewRec.ProxyId = Child_i.proxyId;
            getAgentRef(Child_i.masterid).addChildAgent(Child_i.agentId,NewRec);
            getAgentRef(Child_i.proxyId).addLocalAgent(Child_i.agentId,NewRec);
        END{FOR}

        END SystemDeploy()
```

Figure 4.3: Algorithm for System Deployment

4.3.5 Generation of Events

At first sight, process-oriented programming seems to be the natural way for constructing such a complex simulation model with a large number of autonomous entities. However, the *Process* class in SSJ is implemented with the Java *Thread* class. In the current Java environment, the time required for the creation of a new thread is almost equivalent to that of creating 100 objects [33]. SSJ employs the thread pool [26] technique to improve efficiency. When a process finishes all its activities, its associated thread object is put on a stack of free threads. A new process can reuse one of these free threads when the stack is not empty. However, the simulation model for a communication protocol cannot gain any benefit from this technique because an agent will not end its life until the simulation stops. This means the thread pool is always empty and a new thread object has to be created for every new agent object. For a system with a large number of agents, the use of *Process* will cause a significant performance penalty.

On the other hand, only two types of events will change the state of the system. The first is the message delivery and the second is agent migration. It is fairly easy to construct *MessageEvent* and *MigrationEvent* classes for these two events by extending the *Event* class in SSJ. The random number streams *genMsg* and *genMove* are the random number generators used to generate the times between successive message delivery and agent migration, respectively. The average interval between 2 consecutive message deliveries originating from one agent and the average interval between 2 consecutive migrations of a child agent are defined by the users. These two random variables are considered exponentially distributed [32]. The first event of each type is scheduled during agent creation. The *action* method in each event schedules the next event. Figure 4.4 and Figure 4.5 show the algorithms for these two events.

```
/*  Variable
    meanMsgTm: The average interval between 2 consecutive message delivery
    MSG_COMMUNICATION: Constant for normal message type

    nextDeliverTm: The time for next Message delivery
    msg: Normal message;
    type, endAgentId: Attributes of message
    proxyId: Attributes of Agent
    */

        MessageEvent() operation:

        nextDeliverTm = genExponentialRan(meanMsgTm);
        schedulNextMessage(nextDeliverTm);
        msg = creageMsg();
        msg.type = MSG_COMMUNICATION;
        msg.endAgentId = "Child"+genNewId();
        sendMsg(mgrMsg,proxyId)

        End MessageEvent()
```

Figure 4.4: Algorithm for Message Delivery Event

```
/*  Variable
    meanMigrateTm: The average interval between 2 consecutive migration
    MSG_MIGRATION: Constant for migration message type

    nextMigrateTm: The time for next migration
    newAddr: The destination for migration
    migMsg: Migration message;
    type, endAgentId: Attributes of message
    proxyId: Attributes of Agent
    */

        MigrationEvent() operation:

        nextMigrateTm = genExponentialRan(meanMigrateTm);
        schedulNextMigrate(nextMigrateTm);
        newAddr = genNewAddr();
        migMsg = creageMsg();
        migMsg.type = MSG_MIGRATION;
        migMsg.endAgentId = proxyId;
        sendMsg(mgrMsg,proxyId)
        move(newAddr);

        End MigrationEvent()
```

Figure 4.5: Algorithm for Agent Migration Event

4.4 Verification and Validation

In this research, the major technique that is used to verify the simulation model is a debug trace. For the two major events *Message Delivery* and *Migration*, the state of the simulated system are printed out during debugging at each event occurrence. For agent migration, the agent's original address, new address, the updates of remote agent and local agent tables of relative proxy agents and the updates of the master agent are reported. For message delivery, the initial time, the delivery path, and the updates of remote agent and local agent tables are reported. Figure 4.6 shows part of a trace file. These traces indicate that the program is operating as intended.

```
ID=HOME ADDR=0
ID=Proxy0 ADDR=1
ID=Proxy1 ADDR=11
    .
    .
    .
ID=Master0 ADDR=13
ID=Master1 ADDR=32
    .
    .
    .
ID=Child0 ADDR=82 ProxyId=Proxy8 proxyAddr=81 masterId=Master0 masterAddr=13
Schedule at0.0
ID=Child1 ADDR=53 ProxyId=Proxy5 proxyAddr=51 masterId=Master1 masterAddr=32
Schedule at0.0
    .
    .
    .
Send message from Child0 to Proxy8  end agent=Child15 at time 0.0
Send message from Proxy8 to HOME  end agent=Child15
Send message from HOME to Master1 end agent=Child15
Send message from Master1 to Proxy7 end agent=Child15
Proxy7 receive message from Child0
Send message from Proxy7 to Child15  end agent=Child15
Child15 receive message distance= 460
    .
    .
    .
Child28 migrate from 63 to 38 at 57.55108137305862
Child28 enters new domain Proxy3
Child28 send REGISTRATION Proxy3
Send message from Child28 to Proxy3  end agent=Proxy3
Proxy3 updates local table Child28:38
Proxy3 sends UPDATE to Master1
Send message from Proxy3 to Master1  end agent=Master1
Master1 updates child table Child28:Proxy3
Send message from Proxy3 to Proxy6  end agent=Proxy6
Proxy6 del local table Child28
Proxy6 add remote table Child28:Proxy3
    .
    .
    .
```

Figure 4.6: Trace File Sample

Another issue in the simulation verification is to verify the seed independence. Seed independence means the seed value used to initialize the random-number generation should not affect the final results. Thus, the model should produce similar results for different seed values [29]. To verify this, three simulation cases

are run with four different seed values. In the first case, the value of control parameters are set to 5 percentile of the range. In the second case, the value of control parameters are set to 50 percentile of the range. In the third case, the value of control parameters are set to 95 percentile of the range.

Table 4.1 and Table 4.2 summaries the simulation results. The *MaxDiff* row represents the maximum difference between results of the same simulation case with different seeds. Since all the maximum difference is less than 1%, we can safely conclude that the selection of seeds does not affect the simulation result and the difference between simulation runs is due to different configurations.

	Case 1		Case 2		Case 2	
	Home	Proxy	Home	Proxy	Home	Proxy
Seed 1	446.99	454.86	449.80	454.64	449.88	456.03
Seed 2	446.58	456.07	449.84	454.85	449.83	456.27
Seed 3	447.35	455.85	449.92	454.13	449.94	456.33
Seed 4	446.92	456.57	449.45	454.38	449.78	455.88
MaxDiff	0.17%	0.38%	0.1%	0.16%	0.04%	0.01%

Table 4.1: Mean Message Delivery Cost for Simulations with Different Seeds

	Case 1		Case 2		Case 2	
	Home	Proxy	Home	Proxy	Home	Proxy
Seed 1	1.62E7	1.72E7	1.54E7	1.56E7	1.53E7	1.48E7
Seed 2	1.62E7	1.72E7	1.55E7	1.57E7	1.54E7	1.49E7
Seed 3	1.62E7	1.72E7	1.55E7	1.56E7	1.54E7	1.49E7
Seed 4	1.61E7	1.71E7	1.54E7	1.56E7	1.53E7	1.48E7
MaxDiff	0.62%	0.58%	0.64%	0.64%	0.66%	0.46%

Table 4.2: Total Communication Cost for Simulations with Different Seeds

It is relatively easy to validate the home-server protocol. As discussed in Section 2.3.1, the maximum time for message delivery time will not exceed $3 * 150ms = 450ms$ because of the triangular delivery path. And in most cases, the delivery time is close to $450ms$ because the home server, master agent and child agent usually do not reside in the same domain. The simulation results confirm this

theory by showing that none of the simulation runs generates average delivery time for communication messages that is greater than $450ms$ and in most of the runs it is close to $450ms$. Another result that confirms the validity of the model is the average delivery time of the update message which is close to $150ms$ in most of the cases.

On the other hand, there is no existing theory that can be used to validate the proxy-based protocol. The only validation we can have from the simulation results is that the number of communication messages is the same in both protocols and the average delivery time of the update message is close to $150ms$. Another proof is that, unlike the home-server protocol, the average message delivery time of communication messages of the proxy-based protocol is not stable. The value of the delivery time varies in different settings. This result indicates the influence of the proxy agents. Further data analysis is shown in Chapter 5.

4.5 Implementation

The simulation model is developed in Java 2 SDK and can be run on any Windows platform with Java 2 Runtime Environment installed. Figure 4.7 shows the graphic interface for the the simulation model. The interface allows users to define the control parameters, such as number of computers, number of child agents, average time interval between messages delivery and simulation stop time, etc. After defining all the parameters, the user can run the simulation by pressing the *Run Simulation* button.

Figure 4.8 is the report windows of the simulation which shows five different performance metrics for both protocols.

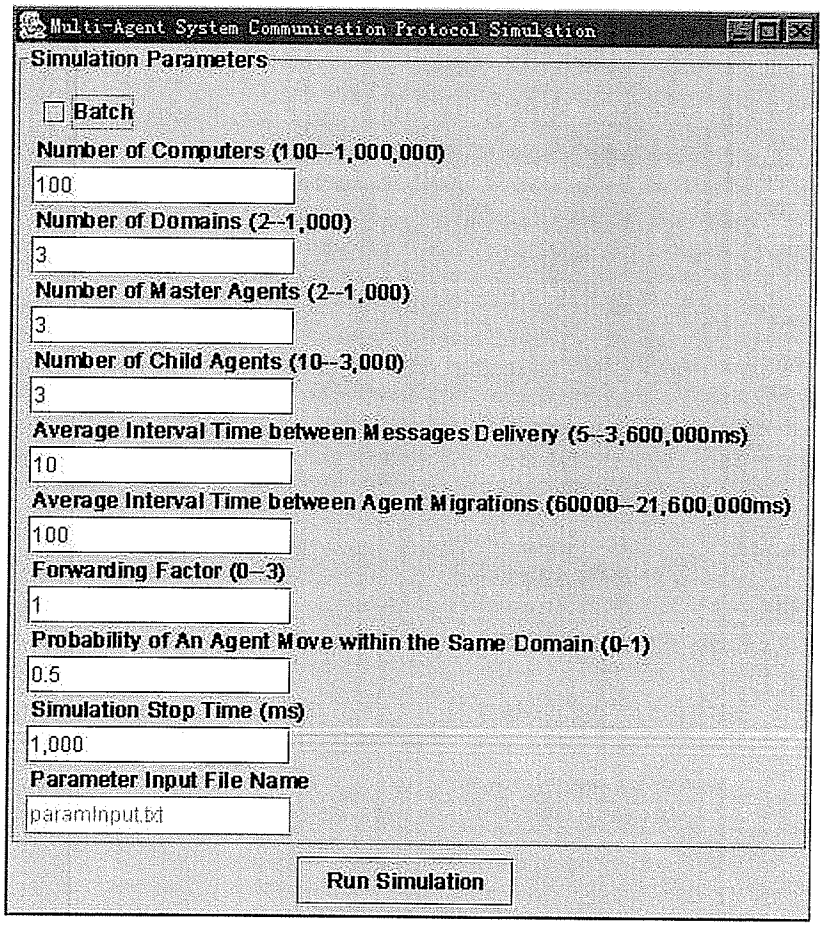
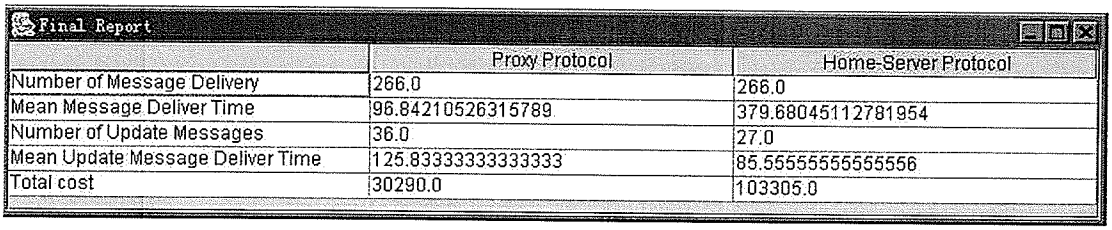


Figure 4.7: Simulation Input Window



	Proxy Protocol	Home-Server Protocol
Number of Message Delivery	266.0	266.0
Mean Message Deliver Time	96.84210526315789	379.68045112781954
Number of Update Messages	36.0	27.0
Mean Update Message Deliver Time	125.83333333333333	85.55555555555556
Total cost	30290.0	103305.0

Figure 4.8: Simulation Output Window

A user can also use the batch function of the simulation model. When the batch check box is selected, the user can only enter the input file name. Figure 4.9 shows the format of an input file for the simulation.

```
0.05 0.95 0.05 0.95 0.05 0.95 0 0.05
0.05 0.05 0.95 0.05 0.95 0.95 0 0.05
0.95 0.05 0.95 0.05 0.05 0.05 0 0.95
0.05 0.05 0.05 0.05 0.05 0.05 0 0.95
0.95 0.05 0.05 0.95 0.05 0.95 0 0.05
```

Figure 4.9: Simulation Input Quantile File

Each row in the file represents the parameters for one simulation run. The simulation stops on reaching the end of the file. The order of the parameter is the same as shown on the input screen, *number of computers, number of domains, number of master agents, number of child agents, average time interval between messages delivery, average time interval between agent migrations, forwarding factors* and *probability of an agent move within the same domain.* The number in the input file represents the quantile of the parameter distribution. All these parameters are considered uniformly distributed between their minimum possible value and maximum possible value. The simulation model translates the quantile to the actual value. For example, the value of *number of computers* is between 100 and $1,000,000$. If the value in the input file is 0.05 for the parameter, then the actual value is the 5 percentile of the range:

$$100 + (int)(0.05 * (1000000 - 100 + 1)) = 950005$$

For the batch function, the default simulation time is $43200000ms$ (12 hrs).

The output results are recorded in file *homeSim.out* and *prxoySim.out.* Figure 4.10 shows the format of the *proxySim.out* file. The format of the *homeSim.out* is similar except it does not include forwarding factor.

Each row represents the control parameters and five performance metrics for the proxy-based protocol in one simulation run. Notice the value of a control

```
________Control Parameters_____________  ________Performance Metrics________
950005 951 51 2851 180004  20523000 0 0.95  614342 412.76 552    150    2.54E+08
50095  51  51 2851 180004  20523000 0 0.95  614342 205.16 724    148.59 1.26E+08
950005 51  51 2851 3420001 1137000  0 0.05  32082  490.32 184674 148.56 4.32E+07
950005 951 51 2851 180004  1137000  0 0.05  615128 476.66 185362 149.92 3.21E+08
50095  951 51 159  180004  20523000 0 0.95  34422  302.85 38     150    1.04E+07
```

Figure 4.10: Output File

parameter in the output file is the actual value instead of the quantile value in the input file. The batch function allows the model to run multiple simulations automatically without user interference.

Chapter 5

Data Analysis

This chapter shows the analysis result of the output data from the simulation model. The basic proxy-based protocol and proxy-based protocol with limited-forwarding algorithm are analyzed independently. A sensitivity analysis is performed for the basic proxy-based protocol.

5.1 Basic Proxy-Based Protocol

5.1.1 Experimental Design

Experimental design provides a useful technique for measuring the effect of different factors on a system's performance. A well-designed experiment can offer maximum information with minimum cost. Also, experimental design helps in identifying factors that have significant effect on performance [29]. In this study, a two level full factorial design is used to determine the effect of 7 factors (forwarding factor is set to 0), each of which has two alternative levels. Therefore, the number of simulation runs is 128. This design can estimate not only additive effects for each factor but also the effects of pair-wise interactions between factors.

Traditionally, the 2 level in the factorial design are labelled -1 and 1. The

values of these two levels are decided by the parameter distribution. Table 5.1 summarizes the distribution information for the parameters.

Variable	Meaning	Distribution	Min	Max
NUMCMP	Number of computers	Uniform	100	1,000,000
NUMDMN	Number of domains	Uniform	2	1,000
NUMMST	Number of master agents	Uniform	2	1,000
NUMCHD	Number of child agents	Uniform	10	3,000
INTMSG	Average time interval between messages delivery	Uniform	5 ms	3,600,000 ms (=1 hr)
INTMIG	Average time interval between agent migrations	Uniform	60,000 ms (=1 min)	21,600,000 ms (=6 hrs)
PSTAY	Probability of an agent move within the same domain	Uniform	0	1

Table 5.1: Distribution for Factors

The simplest way is to assign the minimum value to the -1 level and the maximum value to the 1 level. However, by doing this, we only simulate some extreme cases that are very unlikely to happen in real world. For example, at the extremes for the PSTAY variable, we either have child agents that always move in the same domain (PSTAY=1) or child agents that always move to another domain (PSTAY=0). To solve this problem, 0.05 and 0.95 quantile values in the parameter distribution for the two levels are used. Table 5.2 shows the values of different factors.

Factor	Level -1	Level 1
NUMCMP (A)	50095	950005
NUMDMN (B)	51	951
NUMMST (C)	51	951
NUMCHD (D)	159	2851
INTMSG (E)	180004 ms ($\approx$ 3 mins)	3420001 ms ($\approx$ 57 mins)
INTMIG (F)	1137000 ms ($\approx$ 19 mins)	20523000 ms ($\approx$ 5.7 hrs)
PSTAY (G)	0.05	0.95

Table 5.2: Factors and Levels

5.1.2 Statistical Analysis

To give a more intuitive view, I transform the output metric. Instead of using the direct result of the average delivery time of a communication message, I calculate the relative difference between the two protocols. Let $C_p(D)$ represents the average delivery time of a communication message in the proxy-based protocol, and $C_h(D)$ represents the average delivery time of a communication message in the home-server protocol. I define the difference of the delivery time as:

$$D_d = \frac{C_h(D) - C_p(D)}{C_h(D)} * 100\%$$

D_d represents the relative difference between two protocol in each runs. $D_d > 0$ indicates that the proxy-based protocol performs better than home-server protocol.

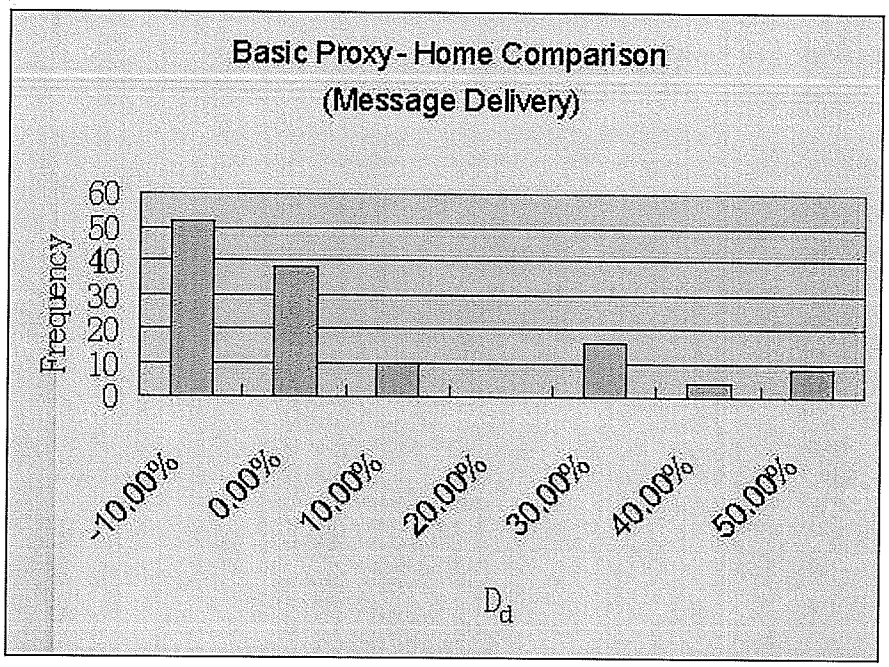


Figure 5.1: The Difference of Average Delivery Time of Communication Messages between Two Protocols

Figure 5.1 shows the statistical result of D_d for a total of 128 runs. 52 out of 128

runs are between -10% and 0%, 38 out of 128 are between 0% and 10%, and 38 out of 128 are greater than 10%. Therefore, in about 60% of the simulation runs, the proxy-based protocol decreases the message delivery time. On a closer look at the data, the worst case is $D_d = -9.71\%$ and the best case is $D_d = 55.32\%$. This shows that, although the proxy-based protocol does not always guarantee better performance for message delivery, it only slightly increases the cost in the worst case. On the other hand, in about 30% of the simulation runs, the proxy-based protocol significantly decreases the delivery cost ($D_d > 10\%$).

In a similar manner, I use the relative difference of the total cost between two protocols to report the result. Let $C_p(T)$ represent the total cost for the proxy-based protocol, and $C_h(T)$ represent total cost for the home-server protocol. I define the difference of the metric as:

$$D_t = \frac{C_h(T) - C_p(T)}{C_h(T)} * 100\%$$

Figure 5.2 shows the statistical result of D_t for a total of 128 runs. 20 results are between -50% and -30%, 26 are between -10% and 0%, 22 are between 0% and 10% , 16 are between 10% and 20%, and 44 are greater than 20%. Therefore, in about 64% of the simulation runs, the proxy-based protocol decreases the total cost for communication, which is almost the same as the result for the message delivery. However, comparing Figure 5.2 with Figure 5.1, we can see the distribution has changed. The difference in these two protocols has increased, both in a positive and negative way. The worst case now is $D_t = -51.06\%$ and 16 out of 128 runs are with $D_t \leq -30\%$. At the same time, 40 out of 128 runs are with $D_t \geq 30\%$. The larger differences indicate that the update cost has a big impact on the total communication cost. This also means we have to carefully choose the protocol to use in different conditions because the proxy-based protocol can increase the total communication cost significantly.

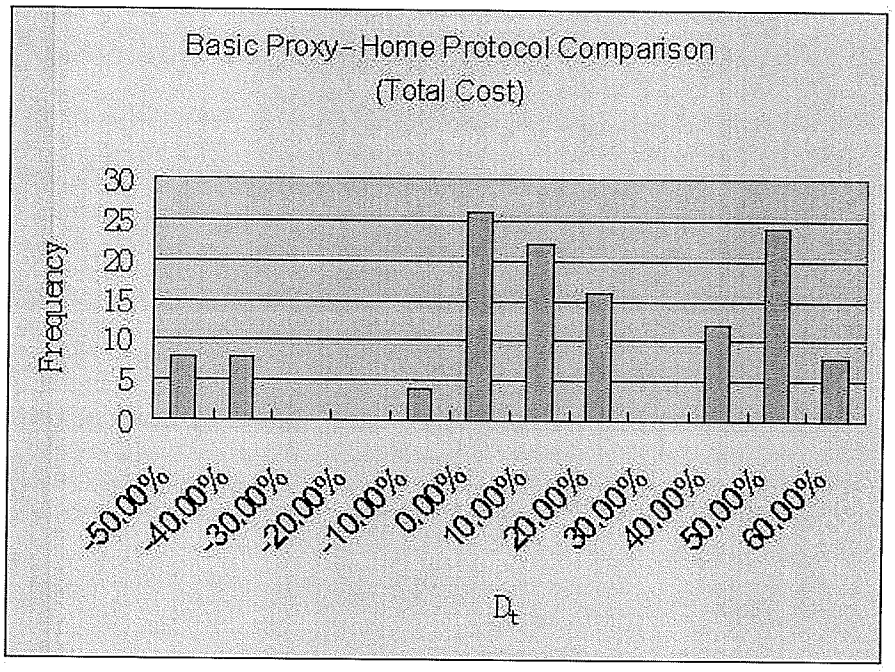


Figure 5.2: The Difference of Total Cost between Two Protocols

5.1.3 Sensitivity Analysis

To get further information on the protocol performance, I perform a sensitivity analysis on the simulation results. The goal of sensitivity analysis is to estimate the variation in the output of the simulation model with respect to different sources of variation [49]. The sensitivity analysis performed here is based on the parameter ranges defined in the previous section. If the ranges of the parameters change, the result of the sensitivity analysis might change as well. However, since the ranges of the simulation parameters are large enough to cover most situations in the real world, this sensitivity analysis can still provide useful information of the relationships between the input and output flow of the model.

A statistical procedure called Analysis of Variance (ANOVA) is used to measure the relative significance of various factors. The idea is to use variance as an indicator of importance for input factors. Table 5.3 and 5.4 are the results of the

most important factors (contribution > 10%) for D_d and D_t.

Factor	Coefficient	Contribution (percent)
INTMSG (E)	-9.83597E-008	29.59
NUMDMN (B)	-2.84115E-004	13.96
INTMIG (F)	+6.95185E-009	13.38
PSTAY (G))	+0.14085	11.84

Table 5.3: Analysis of Variance (ANOVA) Results for D_d

Factor	Coefficient	Contribution (percent)
PSTAY (G))	+0.57063	40.90
INTMIG × PSTAY (FG)	-2.93131E-008	17.95
INTMSG (E)	-1.09208E-007	10.50

Table 5.4: Analysis of Variance (ANOVA) Results for D_t

The Cost of Message Delivery

From the analysis result of the D_d, we can see the most important factor for the cost of message delivery is the the average time interval between message delivery. About 30% of the output variation is due to this factor. The coefficient of the factor shows that the smaller the value, the better the performance of the proxy-based protocol over the home-server protocol in terms of the cost of message delivery. This result agrees with the protocol design. Since the cache information is collected from the incoming messages, the more frequent the message exchanges occur, the more accurate the cache information is.

Variable NUMDMN, the number of domains, contributes about 14% in the variation of the output. The coefficient of the factor shows the smaller the value, the better the performance of the proxy-based protocol over the home-server protocol in terms of the cost of message delivery. This also agrees with the protocol design; when the number of domains is lower, it is more likely that the receiver's information can be found in the cache information.

The average time interval between agent migrations has almost the same contribution to the output variation as variable NUMDMN. The coefficient of the factor shows the larger the value, the better the performance of the proxy-based protocol over the home-server protocol. The result is reasonable because the cache information is valid until the receiver moves to a new location. The longer an agent stays in one location, the more efficient the cache information is.

Variable PSTAY, the probability of an agent move staying within the same domain, contributes about 12% to the variation of the output. The coefficient of the factor shows the larger the value, the better the performance of the proxy-based protocol over the home-server protocol in terms of the cost of message delivery. This result agrees with the protocol design because as long as an agent moves within the same domain, its address remains the same to other agents outside the domain which means other agents can still use the cache information to send messages to it.

The Total Cost

From the analysis result of the D_t, we can see the most important factor for the total communication cost is the probability of an agent move staying within the same domain. About 41% of the output uncertainty is due to this factor. The coefficient of the factor shows that the larger the value, the better the performance of the proxy-based protocol than the home-server protocol. This shows that the update cost plays an important role in the total communication cost. In the proxy-based protocol, a mobile agent need not send update messages to the master agent and home server until it moves to a different domain. However, when the mobile agent moves to a new domain, it also has to send an update message to its former proxy agent. Therefore, if an agent keeps moving to different domains, it will send more update messages in the proxy-based protocol than in the home-server

protocol. As a consequence, the proxy-based protocol can increase the update cost significantly, which in turn affects the total communication cost.

The interaction of variables INTMIG and PSTAY, designated FG, is the second most important factor. The interaction contributes about 18% to the output uncertainty. The coefficient of the factor shows that the smaller the value, the better the performance of the proxy-based protocol than the home-server protocol. This result means that effect of FG is less than the sum of F and G (i.e. F and G are not addition), which is probably due to the different update policy of these two protocols. When the PSTAY value is large, the more frequent an agent moves, the more efficient the proxy-based protocol than the home-server protocol in terms of the update cost. On the contrary, when the PSTAY value is small, the more frequent an agent moves, the higher the update cost of the proxy-based protocol than the home-server protocol. This result also explains why in some cases D_t is much higher than D_d while in some cases D_t is much lower than D_d.

The variable INTMSG, contributes about 11% to the output variation. The coefficient of the factor shows that the smaller the value, the better the performance of the proxy based protocol than the home-server protocol. This result is consistent with the analysis of D_d.

Summary

The sensitivity analysis shows some surprising results. The number of the computers, master agents and child agents are not important factors in the performance comparison between the two protocols. On the other hand, the communication and migration pattern of agents, and the number of domains are more important in the performance comparison. Both analyses show that the proxy-based protocol performs better than the home-server protocol in agent systems with frequent message exchanges. This is a good sign because one of the research goals is to

develop a communication protocol for multi-agent systems with frequent message exchanges.

The analysis of D_t shows the proxy-based protocol is more suitable for highly-dynamic agent systems as long as the value of PSTAY is high. This indicates that we should consider both migration rate and migration patterns of agents when deciding whether to use the proxy-based protocol. At first, it seems the high PSTAY value limits the use of the proxy-based protocol in the real world. However, in the simulation model, the movement of a child agent is totally random while in the real world a mobile agent usually moves within certain areas. Therefore, a high PSTAY value is not hard to achieved in a distributed application based on mobile agent technology.

The analysis of D_d shows when the number of domains is high, the proxy-based protocol might perform worse than the home-server protocol. This is a bad sign because it may indicate the proxy-based cannot perform well in large-scale network. To investigate the scalability of the protocol, a simulation was conducted to evaluate the performance of the proxy-based protocol in different network ranges. In this simulation, variables INTMIG and INTMSG are set to the low (0.05 quantile) value to simulate a highly-dynamic agent systems with frequent message exchanges. The unimportant factors are set to medium (0.5 quantile) values. The variable PSTAY is set to different values to simulate different migration patterns.

Figure 5.3 shows the results of the simulation with PSTAY= 0.05 which means that an agent almost always moves to a new domain. In this situation, no matter how small the number of domains is, the proxy-based protocol always performs worse than the home-server protocol. Surprisingly, the difference between the two protocols does not simply become larger as the number of domains increases. The worst performance happens when the domain number equals 251; the proxy-

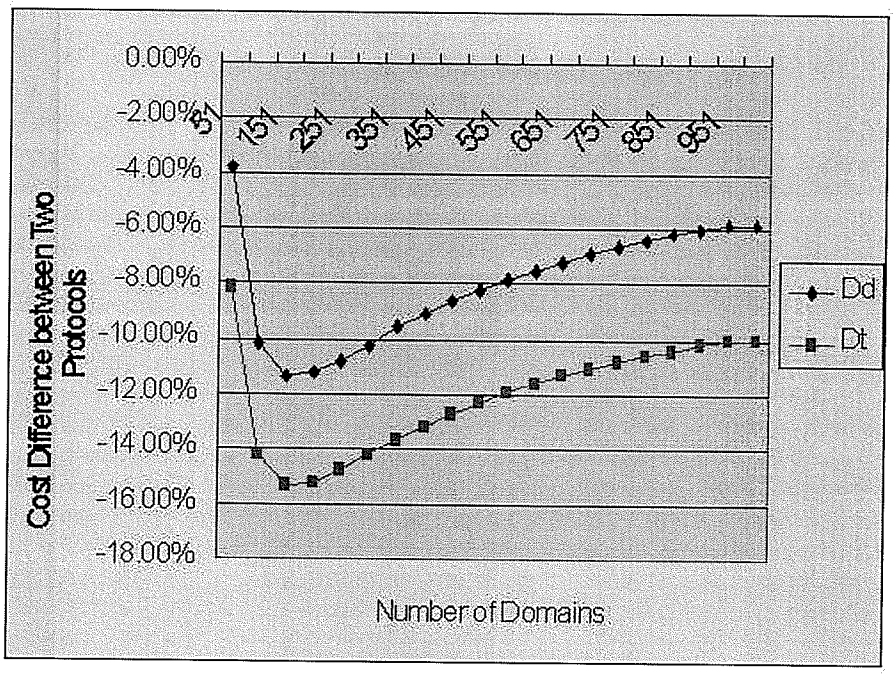


Figure 5.3: Cost Difference between Two Protocols versus Number of Domains. PSTAY = 0.05

based protocol increases total communication cost by about 15% over that of the home-server protocol. When the domain number is larger than 251, the difference between the two protocols becomes smaller.

Figure 5.4 shows the results of the simulation with PSTAY= 0.95 which means the average consecutive moves of an agent with one domain is 20. In this situation, all the cases show that performance of the proxy-based protocol is better than the home-server protocol although the difference of the communication cost becomes smaller as the number of domains increases. The result also matches the sensitivity analysis; the migration patterns of the agent system have more important impact than the network scale. The proxy-based protocol can perform well in large-scale networks when the mobile agents satisfy certain migration patterns. On the other hand, the home-server protocol is preferable even in small-scale networks where

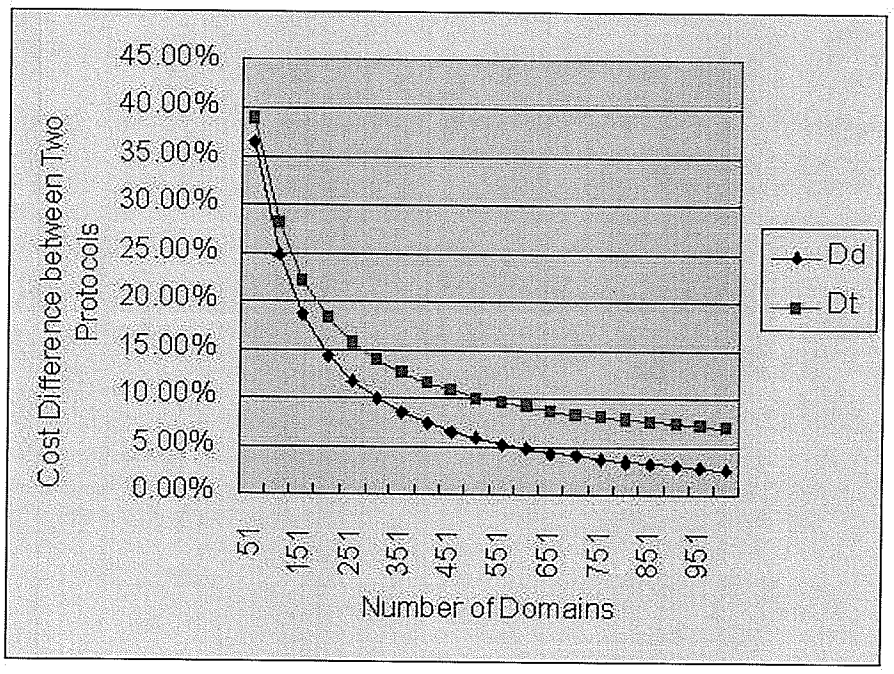


Figure 5.4: Cost Difference between Two Protocols versus Number of Domains. PSTAY = 0.95

the migration patterns are totally random.

5.2 The Limited-Forwarding Algorithm

5.2.1 Experimental Design

In this section, simulations are conducted to investigate the impact of the limited forwarding algorithm on the proxy-based protocol. Here, unlike in the previous section, the focus of the simulation is on the agent system with frequent message changes which means the value of the time interval between message exchange is set to low. The values of unimportant factors, such as the number of computers, master agents and child agents, are set to 50 percentile of their ranges. The other three important factors, NUMDMN, INTMIG and PSTAY, have two alternative

values: the 5 percentile and 95 percentile of their ranges. All combinations of these three factors are simulated with different factors. Therefore, there is a total of 8 sets of simulations, each set has 20 runs where each run simulates a different forwarding factor. When the value of the forwarding factor is 0, the protocol is equal to the basic proxy-based protocol. From the results of simulation, we can see the impact of the limited forwarding algorithm in different situations.

5.2.2 Simulation Results

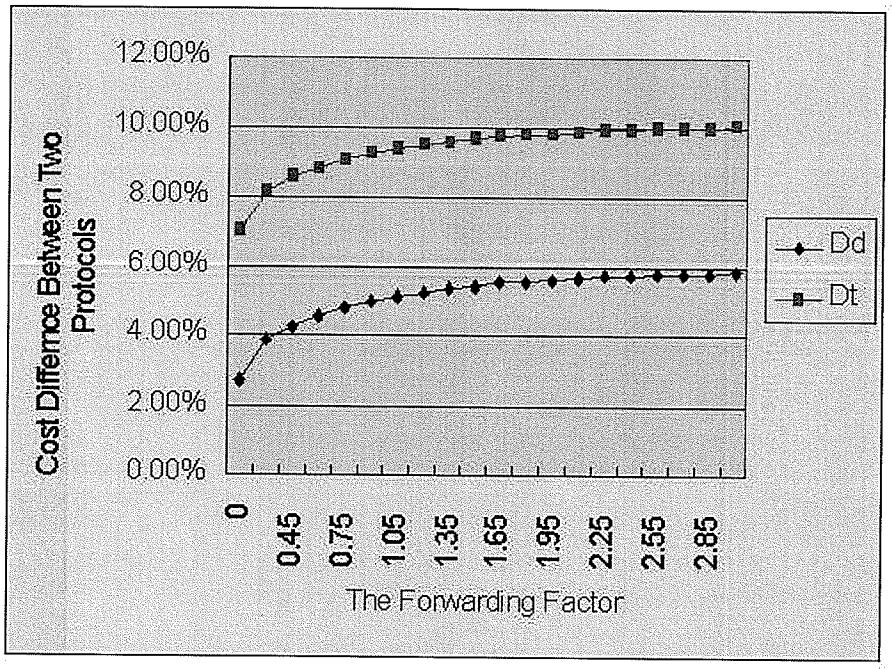


Figure 5.5: Cost Difference between Two Protocols versus Forwarding Factor. NUMDMN = High, INTMIG = Low, PSTAY = High

Figure 5.5 shows the D_d and D_t of a highly-dynamic agent system with high probability of an agent move within the same domain in a large-scale network. We can see the proxy-based protocol becomes more efficient as the forwarding factor increases. Both D_d and D_t improve about 3% when the forwarding factor increases

from 0 to 3.

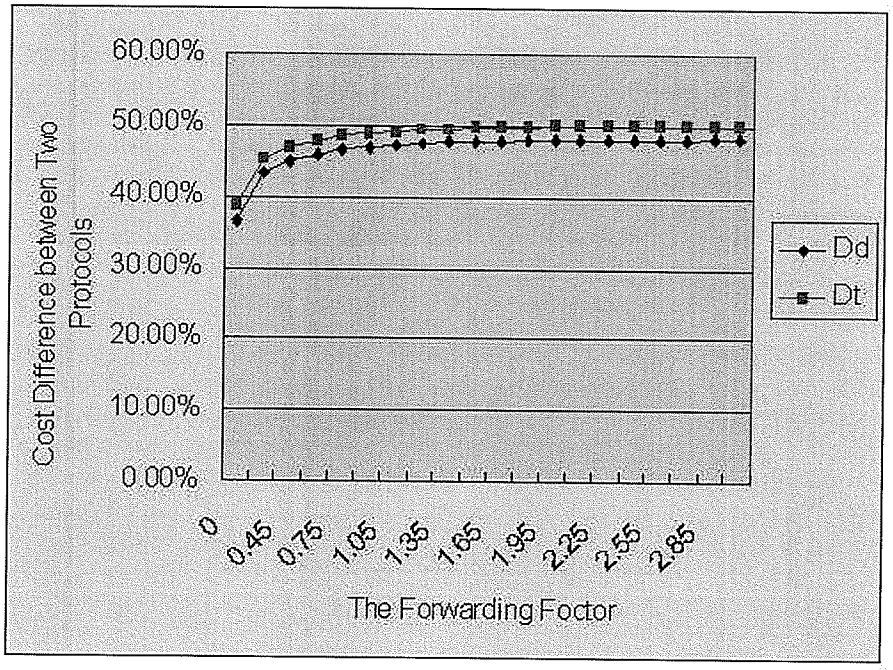


Figure 5.6: Cost Difference between Two Protocols versus Forwarding Factor. NUMDMN = Low, INTMIG = Low, PSTAY = HIGH

Figure 5.6 shows the D_d and D_t of a highly-dynamic agent system with high probability of an agent move within the same domain in a small-scale network. We can see the impact of the limited forwarding algorithm is more obvious in this scenario than the previous scenario. Both D_d and D_t improve about 12% when the forwarding factor increases from 0 to 3 with most of the improvement occurring by FF=1.5.

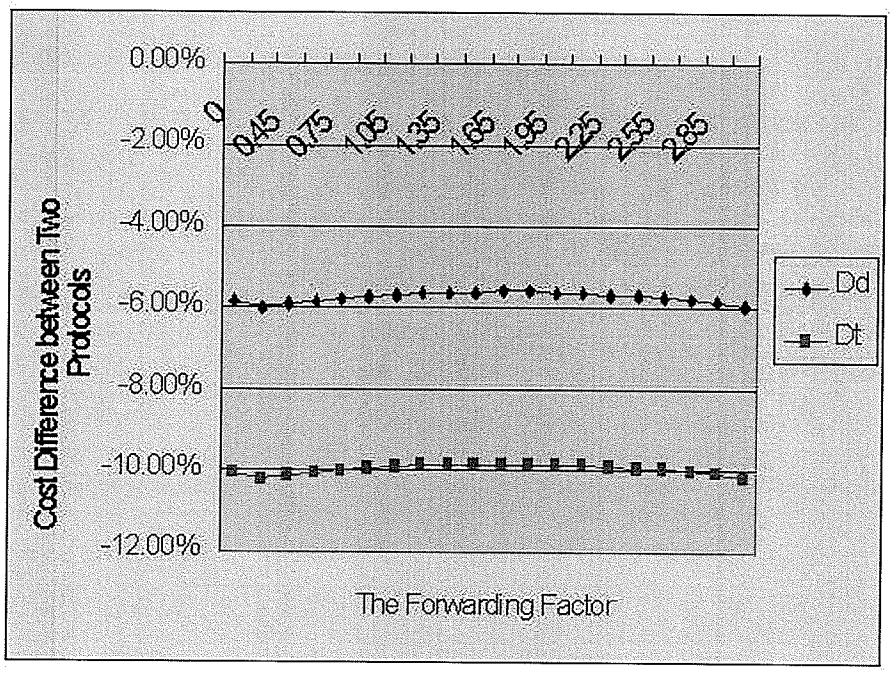


Figure 5.7: Cost Difference between Two Protocols. NUMDMN = High, INTMIG = Low, PSTAY = Low

Figure 5.7 shows the D_d and D_t of a highly-dynamic agent system with low probability of an agent move within the same domain in a large-scale network. We can see the limited forwarding algorithm does not have significant impact on the proxy-based protocol. The D_d and D_t values are almost the same with different forwarding factor.

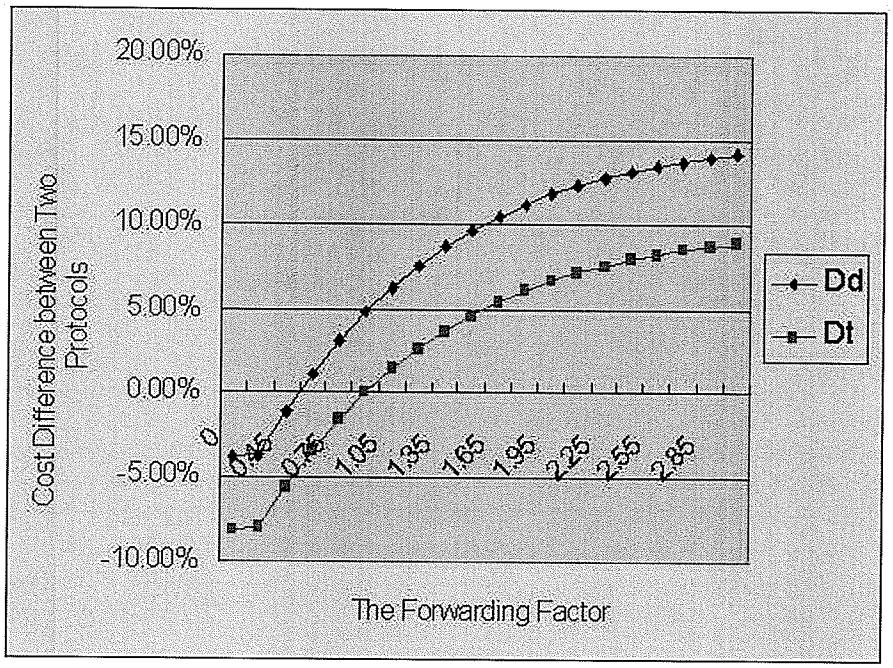


Figure 5.8: Cost Difference between Two Protocols. NUMDMN = Low, INTMIG = Low, PSTAY = Low

Figure 5.8 shows the D_d and D_t of a highly-dynamic agent system with low probability of an agent move within the same domain in a small-scale network. We can see the limited forwarding algorithm works very well in this scenario. Both D_d and D_t improve about 18% when the forwarding factor increases from 0 to 3. The proxy-based protocol performs worse than the home-server protocol when the forwarding factor is less than 1, but when the forwarding factor is larger than 1, the performance of the proxy-based protocol becomes better than the home-server protocol.

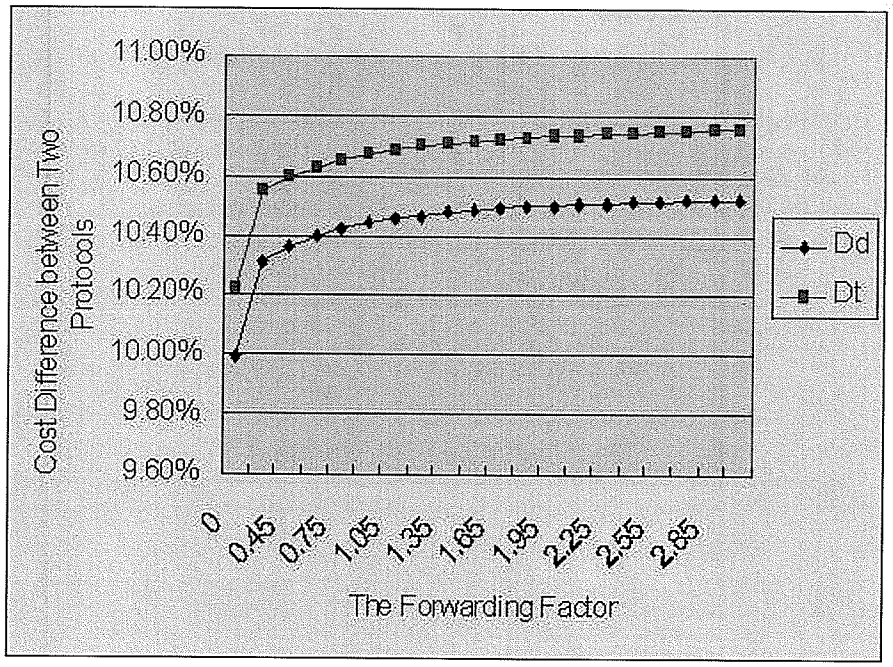


Figure 5.9: Cost Difference between Two Protocols. NUMDMN = High, INTMIG = High, PSTAY = High

Figure 5.9 shows the D_d and D_t of a relatively stable agent system with high probability of an agent move within the same domain in a large-scale network. We can see the limited forwarding algorithm only slightly improves the performance of the proxy-based protocol. Both D_d and D_t improve about 0.5% when the forwarding factor increases from 0 to 3.

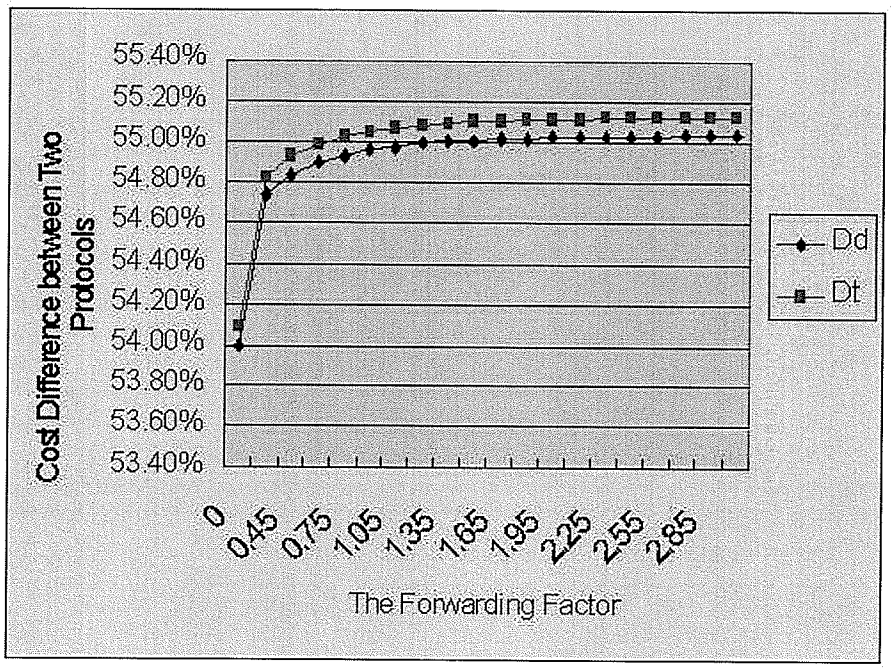


Figure 5.10: Cost Difference between Two Protocols. NUMDMN = Low, INTMIG = High, PSTAY = High

Figure 5.10 shows the D_d and D_t of a relatively stable agent system with high probability of an agent move within the same domain in a small-scale network. Same as the previous scenario, the limited forwarding algorithm only slightly improves the performance of the proxy-based protocol. Both D_d and D_t improve about 1% when the forwarding factor increases from 0 to 3.

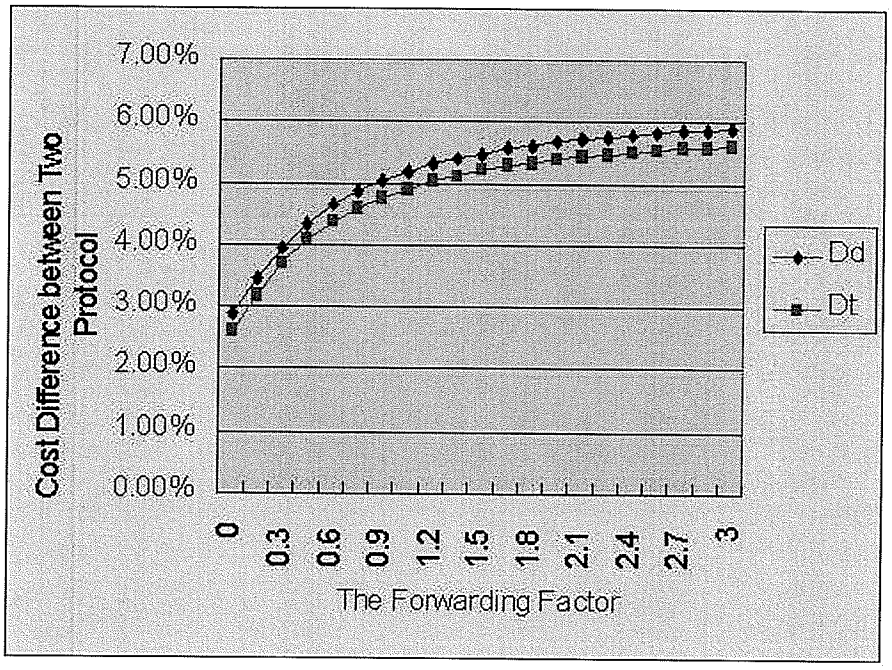


Figure 5.11: Cost Difference between Two Protocols. NUMDMN = High, INTMIG = High, PSTAY = Low

Figure 5.11 shows the D_d and D_t of a relatively stable agent system with low probability of an agent move within the same domain in a large-scale network. The limited forwarding algorithm helps to improve the performance of proxy-based protocol to a certain degree. Both D_d and D_t improve about 3% when the forwarding factor increases from 0 to 3.

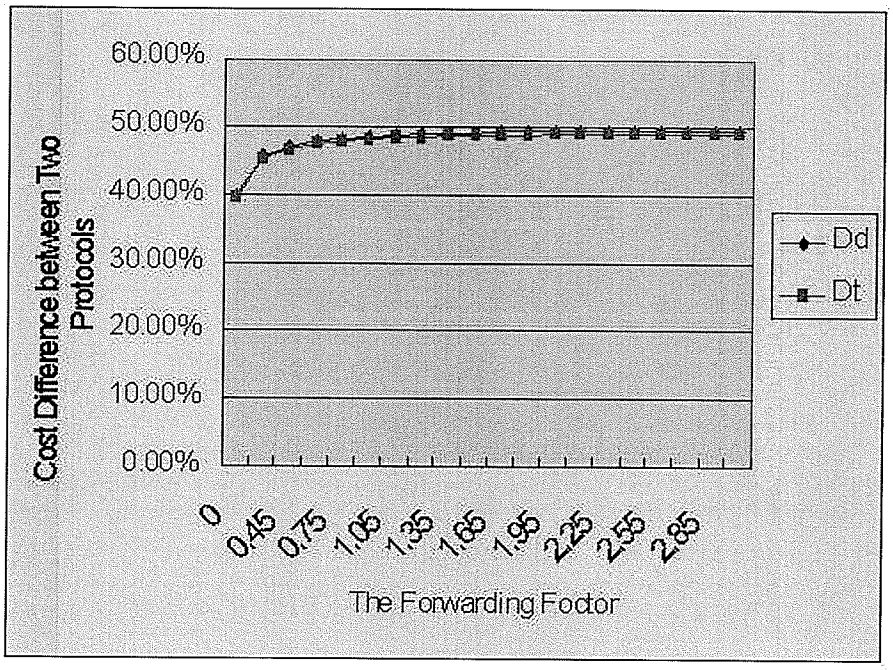


Figure 5.12: Cost Difference between Two Protocols. NUMDMN = Low, INTMIG = High, PSTAY = Low

Figure 5.12 shows the D_d and D_t of a relatively stable agent system with low probability of an agent move within the same domain in a small-scale network. We can see the limited forwarding algorithm works quite well in this scenario. Both D_d and D_t improve about 10% when the forwarding factor increases from 0 to 3.

5.2.3 Summary

Overall, the limited forwarding algorithm helps to improve the performance of the proxy-based protocol. The worst we can get is no improvement at all but the algorithm would not worsen the performance. In some cases, when the basic proxy-based protocol is not suitable, the proxy-based with limited forwarding algorithm performs better than the home-server protocol. In most cases, the performance improves as the forwarding factor increases which means the proxy-based protocol reaches best performance when the forwarding factor is 3. On the other hand, we can see the performance become stable when the forwarding factor reaches about 2. Usually, there is no significant difference between the protocols with forwarding factor 2 or 3. Therefore, the forwarding factor can use any value between 2 and 3.

Chapter 6

Conclusion

6.1 Summary and Contributions

The primary result of this research is a new communication protocol that supports efficient location tracking and inter-agent communication in large-scale multi-agent systems. Mobile agent systems present several challenges to distributed application due to code mobility. Previous research have solved the location transparency problem, but some problems, such as efficiency, reliability and scalability, etc., remained unsolved. The new proxy-based communication protocol described in this thesis uses the concept of proxy agents to reduce communication cost and guarantee reliable message delivery. The significant contributions of this thesis are summarized below.

First, this thesis introduces a proxy-based communication protocol that is capable of supporting efficient and reliable message delivery in mobile agent systems. The system architecture and functionality of its components are presented and algorithms for system operations are discussed in detail.

Secondly, on top of the basic proxy-based protocol, a limited-forwarding algorithm is developed to further improve the performance of the protocol. Using this

algorithm, the proxy agent can determine whether to forward the message to the cached location or to the home server based on the receiver's migration frequency. And by limiting the forwarding steps, the algorithm prevents message chasing and long chain problems that arise in previous forwarding-pointer schemes.

Thirdly, a comprehensive simulation model is developed to simulated both home-server and proxy-based protocol. The model can simulate these two protocols under a wide variety of workloads, network diameters, and behavior patterns of mobile agents; therefore, it provides a good insight into the performance of the protocols in different environments. The simulation results can be sent not only to a graphical user interface but also to an output file which provides a foundation for further data analysis.

Finally, this research provides an in-depth data analysis for the simulation results. For the basic proxy-based protocol, a statistical analysis is used to presents the overall effectiveness of the the proxy-based protocol in terms of performance. In addition, a sensitivity analysis is used to provide further insights for the proxy performance. Important factors are identified and the interpretation of the impact of these factor are given. For the proxy-based protocol with limited-forwarding algorithm, different parameter sets are used to evaluate the performance of the algorithm. The results prove that the limited-forwarding algorithm is able to improve the performance of the proxy-based protocol in most cases.

6.2 Future Work

This section outlines the future research work on the communication protocol for multi-agent systems as an extension of the research in this thesis. The following content will address briefly some of the research problems that remain unsolved in this research.

- Mobile agent systems raise several security issues, such as the authentication of the user, restriction of the user's access and virus detection [16]. Security is a significant concern in electronic commerce in which two parties can conclude a trading contract without prior acknowledgement. The protocol presented in this research uses proxy agents as agent registration and message exchange centers, which provides certain degrees of security control. Based on the proxy-based protocol, further work can be done to resolve the security issues raised by mobile agents.

- In this research, the proxy-based protocol is assumed to be used in absolute reliable networks and multi-agent systems. Since such assumptions are not valid in the real world, future research should focus on improving the robustness of the proxy-based protocol. Different algorithms should be developed to deal with problems such as network failure, agent failure and message loss.

- The analysis results in Chapter 5 indicates that the update cost of the protocol changes more dramatically than the message delivery cost in different environments, which in turn causes significant communication overhead in some cases. The limited-forwarding algorithm only decreases the message delivery cost and has no effect on the update cost. Therefore, more research needs to be done to reduce the communication overhead caused by mobility management. The $dU - SSM$ scheme, an optimal location update and searching algorithm for tracking mobile agents [34], can be considered in the proxy-based protocol to minimize the location update cost. However, to use the algorithm, the architecture of the proposed protocol must be altered and further simulation must be performed to prove the efficiency of the algorithm.

- The sensitivity analysis shows that the migration pattern of the mobile agents is the most important factor for the effectiveness of the protocol. However,

to simplify the simulation model, only one parameter, the probability of an agent move within the same domain, is used to describe the migration pattern. Therefore, the simulation model does not capture all the characteristics of the migration patterns. To get more accurate evaluation, further research can be done to perfect the simulation model by adding more migration parameters.

- In this research, the simulation model is temporal and spatial homogeneity (i.e. all agents have the same delivery and migration patterns). To get more accurate results, agents with various delivery and migration patterns can be used in future simulation model to simulate a more realistic multi-agent system.

References

[1] IBM Official Aglets Homepage. http://www.trl.ibm.com.jp/aglets.

[2] "IP Mobility Support". *Technical Report RFC2002*, Network Working Group, 1996.

[3] ITU G.114: One-way Transmission Time. *International Telecommunication Union*, February 1996. Geneva, Switzerland.

[4] Distributed Computing Overview. *Technical Report*, QUOIN Inc., 1208 Massachusetts Ave., Suit 3, Cambridge, Massachusetts, 1998.

[5] The ObjectSpace Voyager Universal ORB, 1999.

[6] E. Ariwa, M. Senousy, and M. M. Gaber. "Facilities Management and E-business Model Application for Distributed Data Mining using Mobile Agents". *The International Journal of Applied Marketing*, 2(1), 2003.

[7] Baruch Awerbuch and David Peleg. "Online Tracking of Mobile Users". *Journal of the ACM*, 42(5):1021–1058, September 1995.

[8] Jerry Banks, Barry Nelson, and John Carson. *Discrete-Event System Simulation.* Prentice-Hall, Upper Saddle River, N.J.,, 2000. Third ed.

[9] F. Bellifemin, G. Caire, A. Poggi, and G. Rimassa. "JADE - A White Paper". *EXP - in search of innovation*, 3(3):6–19, September 2003.

[10] Andrzej Bieszczad, Bernard Pagurek, and Tony White. "Mobile Agents for Network Management". *IEEE Communications Surveys*, 1(1), 1998.

[11] Grady Booch. *Object-Oriented Design with Applications.* The Benjamin/Cummings Publishing Company, RedwoodCity, Ca., 2 edition, 1994.

[12] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley, 1999.

[13] Brian Brewington, Robert Gray, Katsuhiro Moizumi, David Kotz, George Cybenko, and Daniela Rus. "Mobile Agents in Distributed Information Retrieval". In *Intelligent Information Agents.* Springer-Verlag: Heidelberg, Germany, 1999.

[14] Jiannong Cao, Xinyu Feng, Jian Lu, and Sajal K. Das. "Mailbox-Based Scheme for Designing Mobile Agent Communication Protocols ". *IEEE Computer*, 35(9):54–60, 2002.

[15] Luca Cardelli and Andrew D. Gordon. "Mobile Ambients". In *Proceedings of Foundations of Software Science and Computation Structures: First International Conference, FOSSACS '98.* Springer-Verlag, Berlin Germany, 1998.

[16] David Chess, Colin Harrison, and Aaron Kershenbaum. "Mobile Agents: Are They a Good Idea?". *Technical Report RC 19887*, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, New York, March 1995.

[17] Peter Coad and Edward Yourdon. *Object-oriented Design.* Prentice Hall, 1991.

[18] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes. *Object-oriented Development: The Fusion Method.* Prentice-Hall, 1994.

[19] R.J. Cypser. *Communication for Cooperating Systems.* Addison Wesley, 1991.

[20] Prithviraj Dasgupta, Nitya Narasimhan, Louise E. Moser, and P.m. Melliar-Smith. "MAgNET: Mobile Agents for Networked Electronic Trading". *IEEE Transactions on Knowledge and Data Engineering*, 11(4):509–525, 1999.

[21] Jocelyn Desbiens, Martin Lavoie, and Francis Renaud. "Communication and Tracking Infrastructure of a Mobile Agent System". In *Proceedings of Thirty-First Annual Hawaii International Conference on System Sciences*, page 54, Kohala Coast, Hawaii, USA, January 1998. IEEE Computer Society.

[22] Fred Douglis and John K. Ousterhout. "Transparent Process Migration: Design Alternatives and the Sprite Implementation". *Software - Practice and Experience*, 21(8):757–785, 1991.

[23] Jon Postel (ed.). "User Datagram Protocol". *Technical Report RFC768*, Network Information Center, 1980.

[24] Jon Postel (ed.). "Transmission Control Protocol - DARPA Internet Program Protocol Specification". *Technical Report RFC793*, Information Sciences Institute University of Southern California, 1981.

[25] Sylvanus A. Ehikioya and Quang Trinh. "Distributed Query Processing via Mobile Agents". In *2nd IEEE Electro/Information Technology Conference (EIT-2001)*, Rochester, Michigan, 7-9 June 2001.

[26] Allen Holub. *Taming Java Threads.* Apress, 2000.

[27] Norman Hutchinson. *Emerald: An Object-Based Language for Distributed Programming.* PhD Thesis, University of Washington, January 1987.

[28] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Overgaard. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Reading/Mass. Addison-Wesley, 1992.

[29] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley-Interscience, New York, NY, April 1991.

[30] Danny Lange and Mitsuru Oshima. *Programming and Deploying Java^{TM}Mobile Agents with Aglets^{TM}*. Addison Wesley Professional, 1998.

[31] Danny B. Lange and Mitsuru Oshima. "Seven Good Reasons for Mobile Agents". *Communications of the ACM*, 42(3):88–89, March 1999.

[32] Averill M. Law and David W. Kelton. *Simulation Modeling and Analysis.* McGraw-Hill, New York, 2000. Third ed.

[33] Pierre L'Ecuyer, Lakhdar Meliani, and Jean Vaucher. "SSJ: A Framework For Stochastic Simulation in JAVA". In *2002 Winter Simulation Conference*, pages 234–242, San Diego, California, December 2002.

[34] Tie-Yan Li and Kwok-Yan Lam. "An Optimal Location Update and Searching Algorithm for Tracking Mobile Agent". In *The first International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 639–646, Bologna, Italy, 2002. ACM Press.

[35] Hyojun Lim and Chongkwon Kim. "Multicast Tree Construction and Flooding in Wireless Ad Hoc Networks". In *Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 61–68, Boston, Massachusetts, United States, August 2000.

[36] Yi-Bing Lin. "Determining the User Locations for Personal Communications Services Networks". *IEEE Transactions on Vehicular Technology*, 43(3):466–473, 8 1994.

[37] Anselm Lingnau and Oswald Drobnik. "Agent-user Communications: Requests, Results, Interaction". In *2th International Workshop on Mobile Agents (MA98)*, pages 209–221, Stuttgart, Germany, September 1998. Springer Verlag.

[38] Dejan Milojicic, Markus Breugst, Ingo Busse, John Campbell, Stefan Covaci, Barry Friedman, Kazuya Kosaka, Danny Lange, Kouichi Ono, Mitsuru Oshima, Cynthia Tham, Sankar Virdhagriswaran, and Jim White. " MASIF, the OMG Mobile Agent System Interoperability Facility ". In *Proceedings of 2nd International Workshop Mobile Agents (MA98)*, pages 50–67, Stuttgart, Germany, September 1998. Springer-Verlag.

[39] Luc Moreau. "Distributed Directory Service and Message Routing for Mobile Agents". *Science of Computer Programming*, 39(2–3):249–272, 2001.

[40] Luc Moreau. "A Fault-Tolerant Directory Service for Mobile Agents based on Forwarding Pointers". In *Proceedings of the 17th symposium on Proceedings of the 2002 ACM symposium on applied computing*, Madrid, Spain, March 2002.

[41] Luc Moreau and Daniel Ribbens. "Mobile Objects in Java". *Scientific Programming*, 10(1):91–100, November 2002. *Special Issue of the International Workshop on Performance-oriented Application Development for Distributed Architectures (PADDA'2001).*

[42] Amy L. Murphy and Gian Pietro Picco. "Reliable Communication for Highly Mobile Agents". In *Proceedings of First International Symposium on Agent*

Systems and Applications (ASA'99)/Third International Symposium on Mobile Agents (MA'99), pages 141–150, Palm Springs, CA, USA, October 1999.

[43] Saurab Nog, Sumit Chawla, and David Kotz. "An RPC Mechanism for Transportable Agents". *Technical Report PRC-TR96-280*, Department of Computer Science, Dartmouth College, 1996.

[44] Hyacinth S. Nwana. "Software Agents: An Overview". *Knowledge Engineering Review*, 11(2):205–244, 1995.

[45] Tim Oates, M. V. Nagendra Prasad, and Victor R. Lesser. Cooperative Information Gathering: A Distributed Problem Solving Approach. *Technical Report 94-66*, Department of Computer Science,University of Massachusetts, Amherst, 1994.

[46] Katia Obraczka, Kumar Viswanath, and Gene Tsudik. "Flooding for Reliable Multicast in Multi-hop Ad Hoc Networks". *Wireless Networks*, 7(6):627–634, 2001.

[47] Charles Perkins, Andrew Myles, and David B. Johnson. "IMHP: A Mobile Host Protocol for the Internet". *Computer Networks and ISDN Systems*, 27(3):479–491, 1994.

[48] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented Modelling and Design.* Prentice-Hall, 1991.

[49] Andrea Saltelli, K. Chan, and E. M. Scott. *Sensitivity Analysis.* Wiley Series in Probability and Statistics. John Wiley & Sons, October 2000.

[50] H. Sanneck, M. Berger, and B. Bauer. "Application of Agent Technology to Next Generation Wireless / Mobile Networks". In *Second World Wireless Research Forum*, Helsinki, Finland, May 2001.

[51] Paul M. B. Vitányi Sape J. Mullender. Distributed matchmaking. *Algorithmica*, 3:367–391, 1988.

[52] Marc Shapiro, Peter Dickman, and David Plainfossé. "SSP Chains: Robust, Distributed References Supporting Acyclic Garbage Collection". *Technical Report 1799*, Rocquencourt, France, November 1992.

[53] Sally Shlaer and Stephen J. Mellor. *Object Lifecycles: Modeling the World in States.* Prentice-Hall, 1992.

[54] Raj Srinivasan. "RPC: Remote Procedure Call Protocol Specification Version 2". *Technical Report RFC1831*, Network Working Group, Sun Microsystems Corporation, 1995.

[55] Steve Stone, Mike Zyda, Don Brutzman, and John Falby. " Mobile Agents and Smart Networks for Distributed Simulation". In *14th Distributed Simulations Conference*, Orlando, Fl, USA, March 1996.

[56] Sun Microsystems Corporation. *"Java Remote Method Invocation Specification"*, 1.7 edition, 1999.

[57] Pawel T. Wojciechowski. Algorithms for Location-Independent Communication between Mobile Agents. *Technical Report*, School of Computer and Communication Sciences, Swiss Federal Institute of Technology.

[58] David Wong, Noemi Paciorek, Tom Walsh, Joe DiCelie, Mike Young, and Bill Peet. "Concordia: An Infrastructure for Collaborating Mobile Agents". In *First International Workshop on Mobile Agents 97 (MA '97)*, Berlin, Germany, 1997. Springer-Verlag.