THE UNIVERSITY OF CALGARY

The Applicability of Mobile Agents: A Comparative Study

by

YunBo Wang

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

JUNE, 2000

0-612-55253-5

Abstract

Mobile agents offer different opportunities from the client/sever mechanism for structuring and implementing distributed systems. As an emerging technology, it is widely applied in various applications, including electronic commerce and network management. The benefits of mobile agents have been discussed a lot, but very little effort has been made to verify their claimed advantages over the traditional client/server paradigm.

This research aims to develop an in-depth understanding of the mobile agent mechanism by designing and implementing mobile agent based applications, and to evaluate the applicability of mobile agent applications based on comparisons between the client/server mechanism and the mobile agent mechanism. Real and concrete example applications are developed to provide the basis of verification. The final discussion after implementation concludes that mobile agents are appealing as a design paradigm as a whole, even though the author cannot find overwhelming reasons to apply mobile agents in the individual verification examples.

Acknowledgements

The thesis is to the memory of my mother, QinDi, Yang, whose love was never sufficiently acknowledged, and to my father ShunSheng Wang who inspires me to be my best in all I can do. Without them, I could never make it possible.

My gratitude is extended to everyone who contributed to making this thesis successful: to Dr. John Kendall, who is my supervisor, for his timely advice, consultations and encouragement; to Dr. Kremer who gave me valuable advice; to Dr. Shaw and Dr. Gaines who made criticism and encouragement at the beginning of the research work; and to Zhong Zhang for the thoughtful discussions we had during the work. They together have been great sources of support throughout my graduate program.

I would also like to thank people at the Software Engineering Group, Computer Science Department at the University of Calgary. Special thanks go to Roberto Flores-Mendez, who spent tremendous efforts in reading the thesis, and Andy Kremer, who always helps me figure out system problems as I pursued my project.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Aim

The aim of this research is to develop an in-depth understanding of the mobile agent mechanism by designing and implementing mobile agent based applications, and to evaluate the applicability of mobile agent applications based on comparisons between the client/server mechanism and the mobile agent mechanism.

1.2 Motivations

In the past few years, mobile agent based technology has become more and more popular for filtering information, automating the computing process, and even for buying and selling on the internet using e-commerce. The mobile agent is an emerging technology that claims to facilitate the design, development and maintenance of distributed systems. It has been claimed that the mobile agent is a shift in the distributed computing paradigm with appealing advantages: reduced bandwidth usage, flexible and extensible server API, and improved fault tolerance. Despite the promising features agent technology can deliver, some concerns have been raised such as network security and reliability in agent applications. Today, people are still arguing about the possibility of wide deployment of mobile agents on the network. Will mobile agents really replace the traditional client/server technology? Or are they just optimizations of the client/server architecture? Exactly in what fields does the mobile agent framework outperform the client/server approach? Even the author herself is a little skeptical about the feasibility of mobile agent systems because there are very few publications which actually systematically discuss the applicability problem and validate the claimed benefits from mobile agents advocates. Being that the problem is still open and opinions are divergent, it is definitely necessary to investigate further the applicability of mobile agents for typical mobile agent applications.

The basic idea of this thesis is to demonstrate the mobile agent applicability potential through comparisons between the client/server design paradigm and the mobile agent paradigm. Several real and concrete implementations of mobile agent applications will be provided to support the analysis of their applicability.

1.3 Mobile Agents

There are many definitions of agents. Different definitions reflect different viewpoints of agents. A generic operational definition might be as follows (Knapik, M. and Johnson, J., 1998):

- Autonomy: Agents operate without direct intervention from humans.
- Social ability: Agents interact with other agents.
- Reactivity: Agents perceive and respond to their environment.
- Proactivity: Agents exhibit goal-directed behavior.

In short, agents can be thought of as "active objects with some special properties tailored to open environment" (Jain, A.K., Aparico M., and Singh M. P., 1999). An agent is mobile if it can move. A mobile agent can move to execute on more than one processor, or move in a network in order to use other resources in the most advantageous way. For example, a server can send a mobile agent to a remote host to perform some data processing on the remote host.

There are many examples given as the advantages of mobile agents:

- Elimination of continuous network usage since the mobile agent only occupies the network during the migration phase.
- Real-time updating of services on remote devices.
- Improvement of performance in the network communication mechanism.

1.4 Objectives

The primary objective of writing this thesis is to evaluate the applicability of mobile agents. The actual approaches to reach this objective use the design and implementation of typical mobile agent based applications, and investigation of the advantages and disadvantages of the mobile agent paradigm.

Because the mobile agent mechanism can be applied to many application domains, which can be very broad and complex, the research work will not focus on developing a fully functional multi-agent system from scratch. The IBM Aglets was chosen to be the mobile agent development environment and several application scenarios were designed with simplicity in mind. Though the applications are intended to be test cases for evaluating purposes, it is also possible to integrate these applications into an interactive multi-agent system in the future.

The following objectives are going to be discussed again in chapter 3 when discussing the requirements of mobile agent applications.

Objectives:

- To broadly review literature on mobile agent applications.
- To select typical application scenarios which demonstrate advantages of the mobile agents over the client/server paradigm.
- To develop a mobile agent based application as well as client /server based applications for the purpose of performance comparisons.
- To carry on an evaluation of the applicability of mobile agents based on the application design, implementation result analysis and performance comparisons.

1. 5 Implementation of the Research

Since the aim of this research is to evaluate the applicability of the mobile agent paradigm in distributed computing, the research will use the following approaches:

- Select typical mobile agent application scenarios to reflect the properties of mobile agents. Mobile agent applications, though implemented differently, share common features. Mobile agent applications seek to utilize the distributed network resource in order to accomplish certain tasks. They migrate to a remote host, send information, retrieve information or exchange information, then reside on the remote host for a fixed term or go back to the original host.

- Design and implement the mobile agent applications, as well as an equivalent client/server solution. The term "equivalent" here means the two implementations will result in the same functionality for the communication mechanism comparisons. Concerning the "extended server" functionality and "disconnected network" functionality, the implementation will only provide for mobile agent application.

- Collect relevant data to support the detailed analysis of the pros and the cons of mobile agent communication mechanism. The intended performance data include bandwidth usage.

- Conduct analysis of the applicability of the mobile agent paradigm based on the experiments of mobile agent applications and Client /Server solutions.

1.6 Research Evaluation

The thesis design illustrates the outline of the project. The evaluation of the thesis work is going to follow the criteria below:

- Whether the aim and goals of the thesis are appropriately addressed. Actually, the most important rule for carrying on the research is to follow the aim and objectives of the thesis. So it is necessary to check out whether the outcome and the procedure of the research is actually focusing on accomplishing the aim and objectives.

- Whether the application scenarios are a typical representation of the properties of mobile agents. The application scenarios are the basis of the analysis, so it is expected that those scenarios are representative, embodying the common features of mobile agents, for example, mobility and autonomy.

1.7 Thesis Structure

Chapter 2 covers major background knowledge about mobile agents and paves the way to further development in this thesis. It covers the definition of "agent" as well as "mobile agent" and digs into various research aspects about mobile agents. General knowledge about the client/server paradigm and Java RMI is also presented.

Chapter 3 analyzes three problem situations and summarizes the functional requirements for each individual application. The requirements listed above are based on the overall objective of the project, which is to validate the applicability of the mobile agents.

Chapter 4 validates the mobile agent applicability by three different applications: traveler, watcher, and file searcher. These three prototypes and their designs are used to better understand mobile agent applicability aspects. Some conclusions are drawn from the experiments.

Chapter 5 describes the overall research work that evaluates the applicability of mobile agents, based on the application scenarios and performance comparisons. This chapter concludes the thesis work, summarizes the contributions and points out the future directions for this research.

1.8 Summary

This chapter outlines the main theme of the thesis. The aims and objectives of the thesis are introduced and the research approaches are specified. Mobile agents, as an emerging technology, is appealing but controversial. The thesis will identify some strengths and weaknesses claimed by the mobile agent technology based on a comparison with traditional client /server technology.

Chapter 2. Mobile Agents and the Client/Server Paradigm

2.1 Introduction

What is an agent? There are so many different opinions and debates on this question. "Agents are active, persistent (software) components that perceive, reason, act, and communicate." (Huhns, Singh, 1997).

For the "mobile agent" concept, there is also terminological confusion. This term is used somewhat differently with overlapping semantics in both the distributed system and artificial intelligence research communities (Fuggetta, A., Picco, G.P., and Vigna, G., 1998). A software "mobile agent" is the opposite term of a "stationary agent", which executes only on the system where it begins execution (Lange, D.B. and Oshima, M., 1999). A mobile agent is not bound to the system on which it begins execution. "Mobility" can be explained as "the degree to which agents themselves travel through the network." (Bradshaw, 1997) The mobile agent paradigm is based on the idea that the whole public network is a platform on which developers can build their applications. Mobile Agents are a competitive concept of client/server computing and are especially suitable in constantly changing mobile environments characterized by low bandwidth communication facilities and *ad hoc* connections/disconnections to stationary systems. They are proposed as a suitable paradigm for performing operations in a distributed information space.

As already mentioned above, this thesis is motivated by the need to evaluate the applicability of mobile agents as an emerging distributed design paradigm. It has been claimed that mobile agents are able to personalize the services, reduce network traffic, overcome network latency, and most importantly, construct robust and fault tolerant systems by utilizing the asychronization and autonomy nature of mobile agents. The idea of the thesis is to evaluate the claimed potential applicability of mobile agents by comparing them to client/server design, the most popular distributed design mechanism today.

2.2 The Concepts of Agents and Mobile Agents

2.2.1 Agents

"Agent" is a term which we are very familiar with such as "real estate agent". For people in computer science, it represents a new paradigm due to the proliferation of computing and networking.

Recently, various kinds of software programs have been called agents. There are a number of definitions of what a software agent is and there is no consensus about it. For some, the term "agent" means only "autonomous, intelligent" agent. Franklin and Graesser (Franklin, S. and Greesser, A., 1996) illustrate this type of definition in their paper "Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents". Their work is convincing because they surveyed different views of "agent definitions" and classified the agents' property as shown in Table 2.1:

Property	Other Names	Meaning
reactive	sensing and acting	responds in a timely fashion to changes in the environment
autonomous		exercises control over its own actions
goal-oriented	pro-active purposeful	does not simply act in response to the environment
temporally continuous		is a continuously running process or not
communicative	socially able	communicates with other agents, perhaps including people
learning	adaptive	changes its behavior based on its previous experience
mobile		able to transport itself from one machine to another
flexible		actions are not scripted
character		believable "personality" and emotional state.

Table 2.1 Agent Properties

They made an attempt to capture the essence of agency in a "formal definition" which allows a clear distinction between a software agent and an arbitrary program. Their definition is (Franklin, S. and Greesser, A., 1996): "*An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.*" The Franklin and Graesser's definition equates being an agent with the quality of "autonomous". In the paper "Agent-Based Engineering, the Web, and Intelligence"

(Petrie, C.J., 1996), Petrie enhanced the view of Franklin and Grasser, arguing that "intelligence" is not a necessary property of useful agents and is not helpful in distinguishing agents from other kinds of software.

In this thesis, the author is not trying to identify the various types of definitions of the term "agent". No matter how controversial these definitions are, there are common attributes of programs called "agents" such as reactivity, autonomy, collaborative behavior, and communication ability.

2.2.2 Mobile Agents

As the author have stated, the terminological confusion about "mobile agent" comes from two different research communities (Fuggetta, A., Picco, G.P., and Vigna, G., 1998). In the distributed community, the term "mobile agent" is to denote a software program that is able to move between different execution environments. While in the artificial intelligence community, there is a tendency to blend the notion of "agent", which is illustrated in the above section, with the definition from the distributed community. This mix assumes implicitly that a mobile agent is also intelligent.

In this thesis, the concept "mobile agent" follows the definition of the distributed system community. *Code mobility* can be defined informally as the capability to dynamically change the bindings between code fragments and the location where they execute (Carzaniga, A., Picco, G.P., and Vigna, G., 1997). In the distributed system community, the main problem for code mobility is to support the migration of active processes and objects. The process migration allows the transfer of an operating system process from the machine where it is running to a different one. Object migration makes it possible to migrate objects among address spaces (Fuggetta, A., Picco, G.P., and Vigna, G., 1998). Those migration techniques provide a starting point for mobile agent systems.

Three mobile computation design paradigms have been identified: Remote Evaluation (REV), Code on Demand (COD) and Mobile Agent (MA) (Ghezzi, C. and Vigna, G., 1997). These design paradigms differ in how the code, which is necessary to accomplish the computation and is called "*know-how*", the *resources*, which are the inputs/outputs of the computation, and the *processor*, which is the abstract machine that executes the code and holds the state of computation, are distributed in the two interacting components of distributed architecture. The following table shows the locations of "*know-how*", "*processor*" and "*resources*" before and after the interaction happens between two sites.

Paradigm	**Before**		**After**	
	A Site	**B Site**	**A Site**	**B Site**
Remote Evaluation	know-how	resources processor		know-how resources processor
Code on Demand	resources processor	know-how	know-how resources processor	
Mobile Agent	know-how processor	resources processor	processor	know-how resources processor

Table2.2 Mobile Code Paradigms

A concept of *Remote Evaluation* introduced by Stamos and Gifford (Stamos, J. and Gifford, D., 1990) views the remote server as a programmable interface. In the REV, *resources* and the *processor* are offered by B; it is A that sends the *know-how* to B. The *know-how* will be executed on B (Ghezzi, C. and Vigna, G., 1997). The A host sends the code describing the service to be executed to B. Upon execution on the B host, the code will be allowed to access *resources* and use *processor*. Hence, the A host owns the code, while the B host owns the *resources* and *processor*.

In the COD paradigm, A is unable to execute the task until B provides the code, the *know-how*. Once A receives the code, the computation is carried out on A (Ghezzi, C. and Vigna, G., 1997). The A host can download the code from the B host to perform a given task. In a COD paradigm, the A host owns the *resources* needed to perform a service and *processor*, but is short of code to implement the service.

In the mobile agent paradigm, A has the *know-how* and *processor* capabilities. The computation takes place on B where the resources and another *processor* involved (Ghezzi, C. and Vigna, G., 1997). Hence, the agents in a mobile agent paradigm own the code and *processor* to perform a service, but do not own *resources* to accomplish it.

Mobile agents are software programs, that may be dispatched from a computer and transported to a remote computer for execution. (Harrison, C.G., Chess, D.M., Kershenbaum, A.K., 1995). In other words, a mobile agent is a program that can suspend itself during execution. For the purpose of brevity, the benefits of mobile agents can be listed as follows(Danny, L., 1998):

- Mobile agent technology reduces network traffic because it is relatively convenient to send an agent to a data resource than to send all intermediate data to the remote host. In other words, the mobile agent does not require a permanent connection from the starting server to the destination server.

- Mobile agents can overcome network latency since they can be dispatched to act locally and directly execute the controller's directions.

- Mobile Agents encapsulate protocols. We all know that it is pretty hard to upgrade a protocol if there are security and efficiency requirements on the network, but mobile agents are able to roam on the host server to establish "channels" based on proprietary protocols.

- Mobile agents have the ability to sense their execution environment and react autonomously to changes.

- Mobile agents are generally independent of the computer and the transport layer, and dependent only on their execution environment. So, in essence, mobile agents are compatible with the heterogeneous network computing environment.

- Mobile agents have the ability to react dynamically to unfavorable situations making it easier to build robust and fault tolerant distributed systems.

2.2.3 The Controversy about Mobile Agents

Although the mobile agent community has experienced a strong increase in research activity during the past several years, the use of mobile agents also raises some difficult issues. The research report "Mobile Agents: Are they a good idea?" (Harrison, C.G., Chess, D.M., Kershenbaum, A.,1995) from IBM's T.J. Waston Research Center evaluates mobile agents in a comparing way. An alternative choice is RPC (Remote Procedure Call), which is compared to the mobile agent technique with respect to the protocol they utilize.

- RPC is synchronous; the client process suspends itself, maintaining the entire process state until the server returns the call.

- Mobile agents employ a messaging framework for transport, and hence, they have the ability to be asynchronous. Once the client hands off a message, it can continue its execution.

- The benefit of RPC is that it has high efficiency and low network latency.

- The benefit of messaging is its robustness.

The paper also suggested several problems of mobile agents; namely (Harrison, C.G., Chess, D.M., Kershenbaum, A.,1995) :

- Efficiency -- The agent execution environment can require significant incremental computational resources.

- Flexibility –whether the mobile agent provides a more flexible and robust method of communication than RPC (Remote Procedure Call)or REV (Remote

Evaluation); Is it possible that the agent execution environment could be rapidly deployed on network servers?

- Security -- If the host server can not prevent a malicious agent or an agent can not identify a malicious server. In other words, if the security mechanism is not considered, mobile agent technology could never be successful.

The advantages offered by mobile agents are assessed against alternate methods of achieving the same function. The individual advantages appear to be (Harrison, C.G., Chess, D.M., Kershenbaum, A.,1995) :

- High bandwidth remote interactions.
- Support for disconnected operations.
- Support for mobile clients.
- Lower overhead for secure transactions.
- Robust remote interactions.

2.3 Agent Infrastructures and Agent Architectures

2.3.1 Infrastructures

The term "Infrastructure" layer is often used to refer to a layer in the middle of the operating system, networking software or application specific software. This middleware services have standard programming interfaces and protocols (Knapik, M. and Johnson J., 1998) Examples of infrastructure software includes:

- Inter application communications or interoperability mechanism and frame work, such as RPC (Remote Procedure Call) and compliant products.

- Standard programming language implementation or standard class libraries.

The definition of an infrastructure is "The software that, while not having functions produced directly for the purpose of supporting agent development, can nevertheless be directly useful to the implementation of certain agent-specific mechanisms. " (Knapik, M. and Johnson J., 1998):

For agent infrastructure, the focus is on the following categories (Huhns, M.N., and Singh, M.P., 1997):

- Ontologies: Agents need a common representation of knowledge, which referred as common ontologies, to mediate among the semantic representations of different agents.

- Communication Protocols: The interoperation of multi-agent systems covers the basic communications and cooperation needs for agent systems. The communication and cooperation of agents is approached through the use of an Agent Communication Language (ACL). The Knowledge Query and Manipulation Language (KQML) (Finn, T., University of Maryland, Technical Report, 1994, see http://www.cs.umbc.edu/agents/) describes the design of and experimentation with KQML, a language and protocol for exchanging information and knowledge. KQML is part of a larger effort, which was originally aimed at developing techniques and methodologies for building large-scale knowledge bases which are sharable and reusable. KQML is both a message format and a message handling protocol to support run-time knowledge sharing among agents. Together with other particular conversation policies or patterns, KQML forms the basis of agent cooperation and coordination. KQML can be used as a language for a specific application program to interact with an intelligent system or for two or more intelligent systems to share knowledge in support of cooperative problem solving. KQML is the most prominent emerging agent standard.

- Interaction protocols. Sometimes agent interaction can result in conflicts because agents are usually designed with self-interest in mind. Interaction protocols are to maintain globally coherent performance without violating autonomy. The important aspects include the determination of shared goals and solution for unnecessary conflicts.

The infrastructure that supports the agent application usually offers the following typical services (Knapik, M. and Johnson J., 1998):

- Computation: Sorting, math services
- Information management: Directory services, log manager, file manager, record manager

- Communication: Peer to peer messaging, RPC, message queuing, Electrical Data Interchange support.
- Control: Thread manager, transaction manager, resource broker.

The services that the agent infrastructure offers is very similar to those needed for a complex application. For agent infrastructure, the key services are messaging, remote operation invocation, tasking, scheduling, persistence, resources allocation and deallocation. In addition, mobile agents need specific services like mobility, security and authorization.

2.3.2 Agent Architecture

An agent architecture is the way that agents are put together, that is, the form or structure of their relationship and interactions. (Knapik, M. and Johnson J., 1998) The relationship between the agent, agent architecture and other system components is below (Figure 2.1):

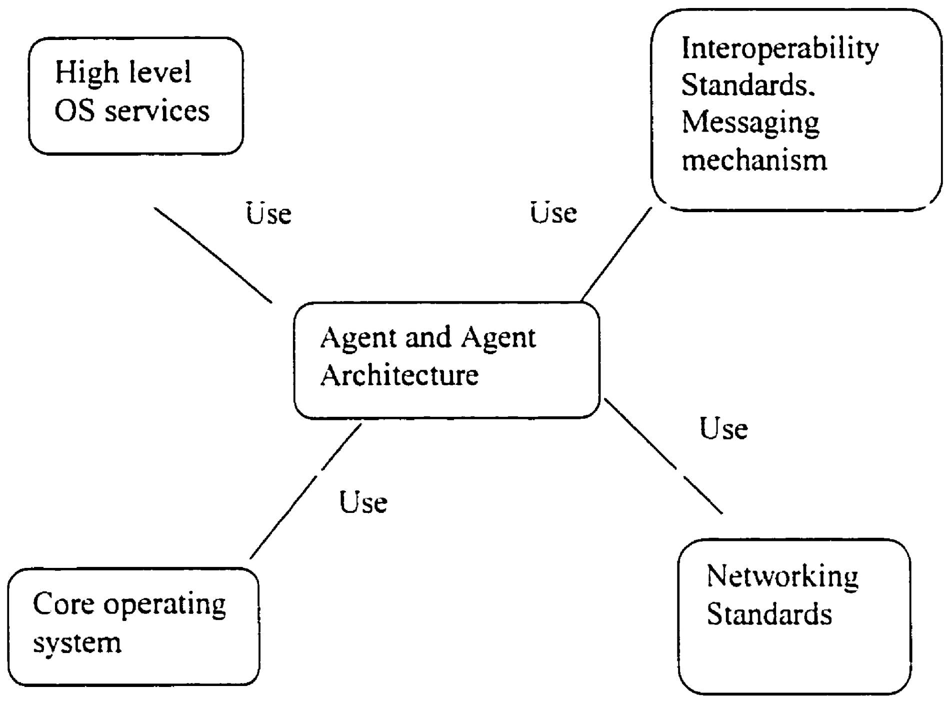
High level OS services
Use
Use
Interoperability Standards, Messaging mechanism
Agent and Agent Architecture
Use
Use
Core operating system
Networking Standards

Figure 2.1 Relationship among Agent System (Knapik, M. and Johnson J., 1998)

Figure 2.1 illustrates that the fine line between infrastructure and architecture. The agent and agent architecture can be seen as a component in the overall system that utilizes infrastructures and other high level components. Agent architecture depends heavily on the available computing and network resources assumed to be part of the infrastructure. On top of these infrastructures and system components lays an agent architecture that directly supports agents.

MASIF

MASIF represents "Mobile Agent System Interoperability Facility", which is a a joint submission of GMD FOKUS, IBM, Crystaliz, General Magic, and the Open Group.

It is an effort to standardize various mobile agent systems and facilitate the interoperability and proliferation of the agent technology. It is a collection of definitions and interfaces which enable the interoperability of multi agent systems. (Milojicic, D., Breugst, M., Busse, I., Campbell, J., Covaci, S., Friedman, S., Kosaka, K., Lange, D., Ono, K., Oshima, M., Tham, C., Virdhadriswaran, S., and White, J., 1998, see http://www.fokus.gmd.de/research/cc/ima/masif/entry-old.html). MASIF specifies two interfaces: MAFAgent System and MAFFinder. MASIF tries to standarize the following:

- Agent Management
- Agent Transfer
- Agent and Agent System Names
- Agent System Type and Location Syntax

The MASIF has not been successfully implemented yet.

2.4 Mobile Agent Applications

2.4.1 Mobile Agent Application Models

In this section several typical mobile agent application models are illustrated (Chess D., Grosof, B., Harrison, C., Levine, D., Parris, C., and Tsudik, G., 1997); namely:

- An information retrieval model -- This model represents an "ask/receive" paradigm between an itinerant agent and a static agent, for instance, when a user sends his agent to various host servers to retrieve some information;

- A collaborative model --This model represents a more complicated interaction in the sense that agents are required to not only to ask for and receive information, but to evaluate and compromise based on a range of preferences. In this case, the itinerant agents convey not only the specified task, but relevant knowledge from the rule bases of the requesters.

- A procedural model -- This model represents a complex interaction controlled by certain protocols in which the agent's goal and resources are hidden from other agents. The example of this model can be an open-bidding auction in which the itinerant agents attempt to bid for goods or services offered by an auctioneer, which is a static agent.

2.4.2 Mobile Agent Design Patterns

Because "agent" technology is a relatively new and emerging field, agent based application design is still a pioneering discipline. From the development of Aglet, an agent development environment, a number of recurrent patterns were recognized (Aridor, Y. and Lange, D.B., 1998).

The patterns they suggested can be divided into three categories: *traveling, task, and interaction*. The division is primarily from the domain and application perspectives in each pattern. Traveling is the essence of mobile agents. The traveling category controls the movement of mobile agents and it mainly consists of three patterns:

- Itinerary -- Objectifies agents' itineraries and routing among destinations. An itinerary maintains a list of destinations and defines a routing scheme.

- Forwarding -- Provides a way for a destination host to forward an arrived agent to another destinations.

- Ticket -- Objectifies a destination address and encapsulates the quality of service and permissions needed to dispatch an agent. A ticket is an enriched version of URLs that embody requirements concerning quality of services, permissions and other data. For example, it may include time out information so that a dispatched agent is able to make reasonable decisions in certain circumstances.

The task category is concerned with the breakdown of tasks and the arrangement of those tasks. The following patterns are fundamental patterns in this category:

- Master-Slave -- It allows a master agent to delegate the tasks to a slave agent. The slave agent will move to a destination to perform certain tasks before it returns or disposes of itself.

- Plan -- The plan pattern is more complex and it adopts a workflow concept to arrange multiple tasks to perform in sequence or in parallel.

The interaction patterns deal with the cooperation of multiple agents. The following are examples:

- Meeting -- This provides a way for two or more agents to initiate a local interaction at a given host. Agents can be dispatched to a specific *meeting place* where they are able to engage in local interaction.

- Locker -- Agents can exploit a *locker* pattern to temporarily store data in private. It defines a private storage space for data left by an agent before it is temporarily dispatched to another host.

- Messenger -- Agents establish communication by employing messenger patterns, which objectify messages in the form of agents that carry and deliver messages between agents.

- Facilitator -- This is a naming and locating service for agents. It is generally convenient to assign a name to an agent in order to locate it.

- Organized Group -- This pattern is used to compose several agents into a group in which all members of the group travel together. This pattern is seen as a fundamental element of collaboration among multiple agents.

2.5 Mobile Agent Development Environments

The idea for mobile agent based computing has been popularized most notably by James White (White, J.E. 1994, see http://www.genmagic.com/technologiy) at General Magic, as well as by the research done by Dartmouth Transportable Agents Laboratory, which is partly supported by the Navy and Air Force. Now Java based mobile agents are becoming more and more popular.

2.5.1 Telescript Technology

The Telescript technology is considered to be the first commercial implementation of mobile agents. In telescript technology, the following principle concepts are employed (White, J.E. 1994):

- Places-- A place offers a service to the mobile agents that enter it
- Agent-- Each agent occupies a particular place.
- Travel --Travel lets agent get service from a remote server and return to its starting place.
- Meetings-- A meeting lets two agents in the same place meet.
- Connections-- A connection lets two agents on different computers communicate.
- Authorities-- Telescript lets one place or agent discern the authority of another.

- Permits--Telescript lets the authorities limit what agents and places can do by assigning permission to them.

Telescript posseses three major components: the language, the interpreter or engine, and the communication protocol. It has been primarily used in relation to network management, active mail, electronic commerce and business process management. But the details of how exactly the Telescript agents travel through mobile hosts is not in the public domain and is not easily understood.

2.5.2 Agent TcL (Tool Communication Language)

Agent TcL (Rus, D., Gray, R., and Kotz, D, 1997) is a research project being pursued at the Dartmouth Transportable Agents Laboratory in Dartmouth. This project has the following attributes:

- It finishes migration in a single instruction.
- It provides simple communication among agents.
- It supports multiple languages and transport mechanism.
- It runs on different platforms.
- It provides effective security, fault tolerance and performance.

Agent TcL is currently implemented in the Tool Communication Language (TcL), a high level scripting language. Thus, a mobile agent is simply a TcL script that runs on the TcL interpreter. To move to another computer, the agent issues a command, suspending the current execution, capturing and saving the internal state of the script, and sending the state image to the destination server , where the agent (TcL Script) continues its execution.

Agents in this system will sense three major external states: hardware, software and other agents in order to remain efficient. Agents can detect whether a network site is reachable. These agents also use information retrieval techniques to detect software changes.

Agents in Agent TcL use an implicit scheme that provides a system of *virtual yellow pages* to help agents find their navigation route. These yellow pages contain lists of service and resources. By consulting the yellow page, an agent can select services relevant to its tasks and get an applicable navigation plan. The agents are able to formulate and reformulate their routing plan by consulting their external state sensors and adapt on-line to changes in network configuration and software content. For instance, if an agent's original plan is to visit three remote hosts, A, B and C, and if the host A is down by accident, the agent is able to adjust its route to B and C.

2.5.3 Java-Based Agent Development Environment

The mobile agent development environment used to be considered for implementation in a script language; but now, the Java platform is becoming more and more popular. The reasons leading to this change can be summarized as follows:

- Java is designed to be a portable, clean and easy- to- learn objected oriented language, which has been re-targeted by the growth of internet. The key success factor is its integration with World Wide Web technology.

- Java is a platform independent language. It is designed to operate in heterogeneous networks. As the mobile agent based application is basically network-centric, so Java which offers platform-independent feature is suitable for this heterogeneous environment.

The java platform has some disadvantages, though, some are listed below:

- Java can not limit the occupation of processor and memory resources by a given object or thread.

- It is currently impossible in Java to retrieve the full execution state of an agent. The dynamic class loading mechanism only supports to retrieve code fragments, not entire execution state such as program counters and call stacks.

Currently, there are several Java based mobile agent environments (Lange, D.B., 1998):

- *Aglets:* Aglets are Java-based autonomous agents developed by the IBM Tokyo Research lab. Aglets were originally designed to bring the flavor of mobility to the applet. So the term "aglet" actually is the combination of "agent" and "applet". Aglets provide the basic requirements for mobility and each aglet has a globally unique name. In order for an aglet to run on a specific host, the host should install an aglet server.

- *Odyssey:* Odyssey is a set of Java class libraries from General Magic. When incorporated in a network application, the libraries enable one to erect agent-accessible places in your network, to station agents in those places, and to let agents travel between places whenever they need to. Whenever an agent travels from one place to another, Odyssey calls upon a transport API designed by General Magic for that purpose.

- Voyager: Voyager is an agent-enhanced Object Request Broker from ObjectSpace, Inc. An ORB provides the capability to create objects on a remote system and to invoke operations on those objects. Voyager agents have mobility and autonomy.

The reason to choose the aglet implementation environment as the tool to build the system presented in this thesis is that:

- Aglets is a pure agent oriented environment, which is good to the thesis work because the author wants to focus on mobile agent related issues. Voyager is a good alternative but it uses concept "object" as building block, which creates a lot of object oriented issues.

- Odyssey is less popular than Aglets and Voyager, and there seems not enough documentation on the system from General Magic.

2.5.3.1 Aglet Development Environment

The Aglets Software Development Kit is an environment for programming mobile Internet agents in Java and was developed by IBM research Toyko Lab. As mentioned, an Aglet actually is a combination of "agent" and "applet". Aglets are Java objects that can move from one host on the Internet to another. That is, an aglet that executes on one host can suddenly halt execution, dispatch itself to a remote host, and resume execution there. When the aglet moves, it takes along its program code as well as its data.

Aglet Architecture

The Aglet architecture consists of two layers and two APIs that define the interfaces necessary for accessing their functions. The Aglets runtime layer is the implementation of the Aglet API, and defines the fundamental classes of the API components, such as AgletProxy and AgletContext. It provides the basic functions required for aglets to be created, managed, and dispatched to remote hosts. The communication layer is for transferring a serialized agent as an object to a destination and receiving it. The relationship between the run time layer and the communication layer is described in the following Figure 2.2 (IBM Tokyo Lab, 1999,see http://www.trl.ibm.co.jp/aglets/spec 11.html):

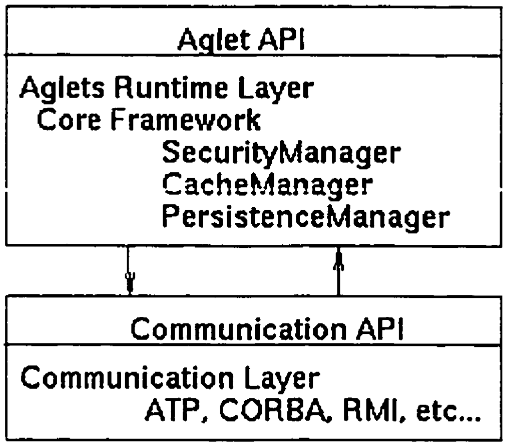

Figure 2.2 Aglet Architecture (IBM Tokyo Lab, 1999)

Aglet Object Model

Aglets Software Development Kit defines the fundamental functionality of mobile agents. Figure2.3 is a simple illustration of aglet common interfaces and APIs (IBM Tokyo Lab, 1999, see http://www.trl.ibm.co.jp/aglets/spec11.html):

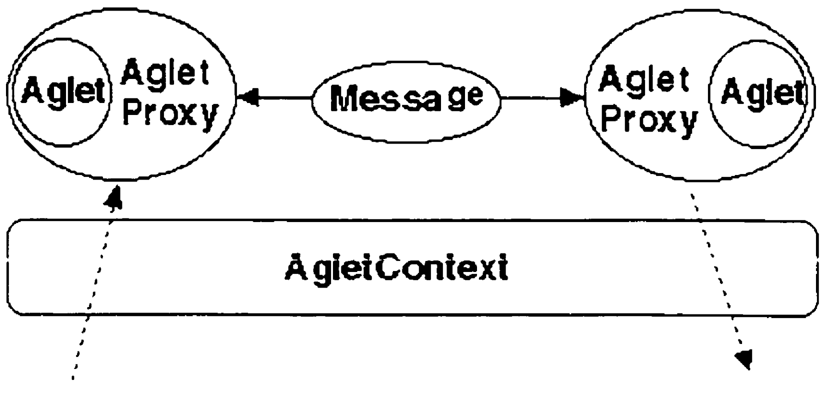

Figure 2.3 Aglet Object Model (IBM Tokyo Lab, 1999)

- The *Aglet* abstract class contains the basic methods used to control the mobility and life cycles of mobile agents. All mobile agents defined in the Aglet development environment have to extend this abstract class.

- The *AgletProxy* interface object acts as a mask for an aglet and provides a common way of accessing the aglet behind it. For security reasons, an aglet class has several public methods that should not be accessed directly from other aglets, any aglet that wants to communicate with other aglets has to first obtain the proxy object, and then interact through this interface. In this way, the aglet proxy object protects an agent from being destroyed.

- The *AgletContext* class provides an interface to the runtime environment that the aglet occupies. The aglet context concept actually is an abstraction of the aglet computation environment.
- Objects of the *Message* class are used for communication between aglets. A message object has a String object to specify the kind of the message and arbitrary objects as arguments.

Mobility

In the Aglets execution environment, mobility is realized by the dynamic class loading and transferring. The class loader is invoked by the java virtual machine (JVM) when the code currently in execution contains an unresolved class name (IBM Tokyo Lab, 1999, see http://www.trl.ibm.co.jp/aglets/spec11.html).

Communication

The Aglets runtime uses the communication API. In this API, there are methods defined for transferring agents, tracking agents, and managing agents. The aglet system presented in this thesis uses the Agent Transfer Protocol (ATP) as the default implementation of the communication layer. The ATP is an application layer protocol for transmission of mobile agents and it is based on the HTTP protocol.

Security

Security is the most important concern for mobile agent systems. The following security features are supported in the latest Aglets version of runtime authentication of users and domains (IBM Tokyo Lab, 1999, see http://www.trl.ibm.co.jp/aglets/spec11.html).

- Integrity checked communication between servers within a domain.
- Fine-grained authorization similar to the JDK1.2 security model.

For security reasons, agents must provide proper user identities so that agent systems can control them according to the access rights of the users and the agent's builders. The domain authorization is also important since aglets from certain organizations can be trusted. Here "domain" means the builder or manufactures of the aglet. Aglet servers are able to determine and authenticate whether the contacting server belongs to a given domain.

Advantages of Aglets

The following is summarized advantages for Aglets (Kiniry J. and Zimmerman, 1997):

- Aglets is very easy to learn. It is well documented and every component of the system comes with GUI.
- Aglets has a clean design. IBM makes the system mirror several models in Java: applets, AWT callbacks, and Java Beans.

2.5.3.2 Voyager

Voyager is a mobile agent development platform developed by ObjectSpace. It provides the flexibility to use both traditional and agent-enhanced distributed programming techniques to create sophisticated agent applications. Voyager philosophy is that an agent is simply a special kind of object that has the ability to move itself and continue to execute as it moves.

It has following properties (ObjectSpace, 1997):

- Voyager uses Java 1.1 and it uses serialization extensively. Voyage was designed to use regular Java message syntax to construct remote objects, send them messages and move them between applications.

- Voyager is a single, unified platform providing mobility and autonomy. In Voyager, an agent is a special kind of object. Voyager supports object mobility using object serialization. Objects can be moved to a new program even when they are receiving messages. Voyager takes care of all the synchronization and message –forwarding issues.

- Voyager supports different communication facilities: one way, future and synchronous message using TCP communications. Voyager allows you to send a regular Java message to a stationary or moving agent

- Voyager comes complete with a VoyagerSecurityManager that can be optionally installed to restrict the operations mobile agents can perform.

2.6 The Client/Server Paradigm

2.6.1 Concept

There seems to be a general agreement that Client/server computing is the logical extension of modular programming. Modular programming has as its fundamental assumption the division of large pieces of software into constituent parts of "modules" and creates the possibility for easier development and better maintainability. As mentioned by Larry T. Vaughn, "the client/server architecture is an application design approach that results in the decomposition of an information system into a small number of server functions, executing on one or more hardware platforms, that provide commonly used services to a larger number of client functions, executing on one or more different but interconnected hardware platforms, that perform more narrowly defined work in reliance on the common services provided by the server functions. " (Vaughn, L., 1994)

Therefore, client/server can be seen as a design approach that distributes the functional processing of an application across two or more different processing platforms. In short, the client is a process that requests services from a server process. Client processes usually manage the user-interface portion of the application, validate data entered by the user, dispatch requests to server programs, and sometimes execute business logic. Server programs generally receive requests from client programs, execute database retrieval and updates, and manage data integrity and dispatch responses to client requests.

2.6.2 The Client/Server Architecture

The client/server architecture may be summarized as follows:

- Two-tier architecture: where a client talks directly to a server, with no intervening server. This is typically used in small environments.

- Three-tier architecture: This introduces another server between the client and the server. The role of the middle server is manifold. It can provide translation services (as in adapting a legacy application on a mainframe to a client/server environment) or metering services (as in acting as a transaction monitor to limit the number of simultaneous requests to a given server), and so on.

The basic characteristics of client/server architectures are:

- The client and server processes have different requirements for computing resources such as processor speeds, memory, disk speeds and capacities, and input/output devices. The server processes require more system resources.

- Client/server applications can be scaled in different ways. Scalability is a characteristic which describes how easily a given system solution can be scaled upward to maintain an acceptable response. It is easy to add or remove client workstations with only a slight performance impact or to migrate to a larger and faster server machine or multi servers.

2.7 Java RMI

Remote Method Invocation (RMI) is a technology intended to create distributed Java applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, even on different hosts. A Java program can make a call on a remote object once it obtains a reference to that remote object. RMI is a remote procedure call for objects. RPC (Remote Procedure Call) means calling a function in a library. RMI is the natural progression when moving the same notion into the OO world.

From the architectural view, a remote method invocation from a client to a remote server object travels down through the layers of the RMI system to the client-side transport, then up through the server-side transport to the server. The RMI consists of three layers (Sun Microsystems, Inc. see http://java.sun.com/products/jdk/1.1/docs/guide/rmi/) as the following Figure2.4 shows:

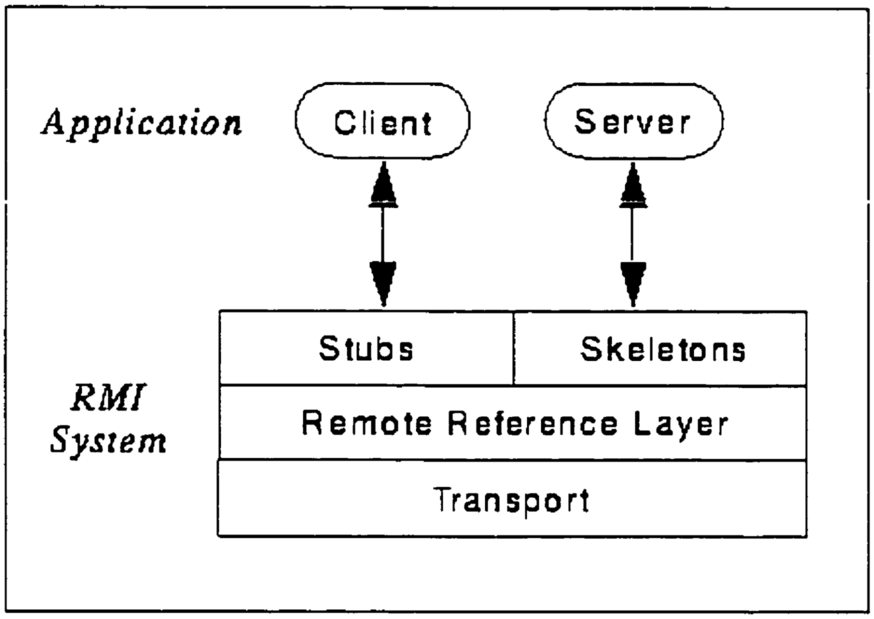

Figure 2. 4 RMI Architecture

- The stub/skeleton layer is actually a proxy layer for the client and the server. When a client makes a remote call to a server object, the stub or proxy of the remote object on the local client acts as a channel to the remote object. A stub can transfer a request to the reference layer using object serialization. A skeleton for a remote object is a server-side proxy that contains a method which dispatches calls to the actual remote object. The stub and skeleton classes are determined and dynamically loaded at run time.

- The remote reference layer handles the transport interfaces and also transfers data to the transport layer.

- The transport layer takes care of the connection details. It is responsible for setting up the connection and managing the connection.

2.8 Summary

This chapter covers major background knowledge about mobile agents and paves the way to further development in this thesis. It covers the definition of "agent" as well as "mobile agent" and digs into various research aspects about mobile agents. The general knowledge about client server paradigm and Java RMI is also introduced.

Chapter 3 Requirements Analysis

This chapter analyzes three problem situations which show the advantages of the mobile agent paradigm over the client/server paradigm according to objectives of the research, and summarizes the requirements of the typical mobile agent applications and client/server applications.

3.1 Objectives of Research

Mobile agents are intelligent programs that can migrate through computer networks. The concept of having mobile agents carrying out tasks and traveling through a network is creating a new paradigm for network-enabled distributed computing. Now an understanding is just emerging of the potential brought by this technology and of various arguments surrounding it. In contrast, the client/server approach is a mature distributed computing paradigm which is broadly applied in today's network computing.

For example, in Figure 3.1(Lange, D.B.,1998), the comparison of client/server paradigm and mobile agent paradigm from the perspective of system developer is shown. In the client server paradigm, the server possesses a set of services that "know how" to access the resources and the services code is hosted on the server itself. So far, most distributed systems have been based on this paradigm. In the mobile agent paradigm, a mobile agent holds the "know how" , has the ability to move from one host to another, and access their resources. So the client and server have merged and become a host and "Know

How" is not tied to a single host but available throughout the network. This is a high level advantage.

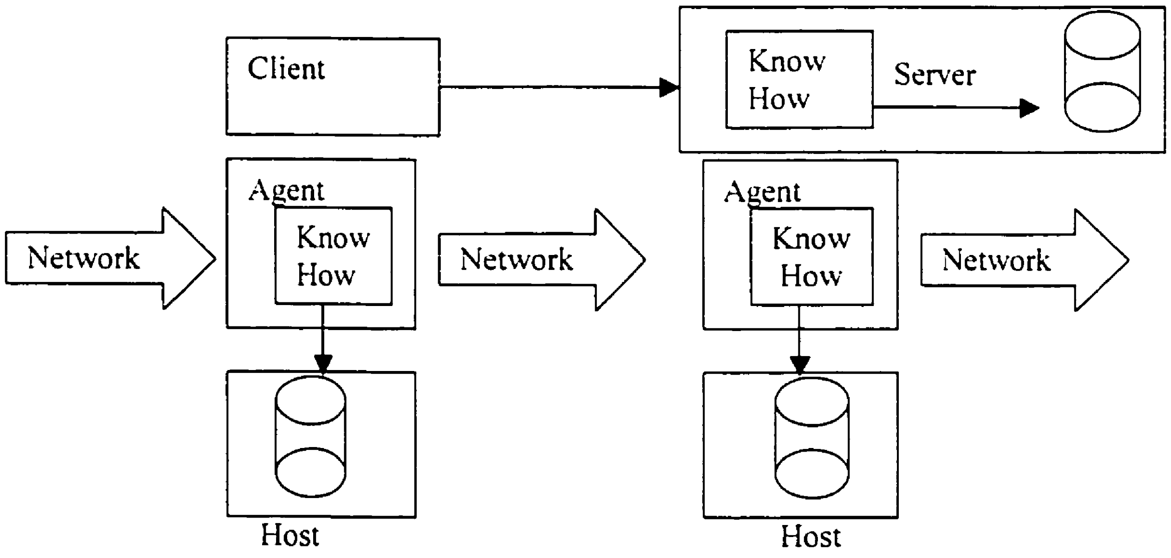

Figure 3.1 Client/Server Paradigm vs. Mobile Agent Paradigm

As mentioned in Chapter 1, there are disputes about what kind of benefits mobile agents can generate and little systematic work has been done around this problem. So the comparison of these two paradigms should bring out an in-depth understanding of the mobile agent technology. The thesis work will focus on evaluating the applicability potential of the mobile agent system through real and concrete examples.

Now at this stage, in order to discuss the basic requirements of the thesis work, I will review the objectives of the thesis:

- To broadly review literature on mobile agent applications.
- To select typical application scenarios which demonstrate the advantages of mobile agents over the client/server paradigm.
- To develop a mobile agent based application as well as a Client /Server based application for the purpose of performance comparisons.
- To carry out an evaluation of the applicability of mobile agents based on the application design, implementation result analysis and performance comparisons.

3.2 Problem Description

3.2.1 Support of disconnected network

Motivation

In a real distributed environment, mobile devices like laptops have become more and more popular. The mobile devices are usually light equipped with resources, which makes it is impossible to run some complex computations locally. Those computations can run at a remote host with more specialized resources. The traditional client/server relies on the continuous network connections, which is fragile and inconvenient for people holding the mobile devices.

Mobile Agent Solution

The autonomy and asynchronization nature of mobile agents make it possible to resolve this problem. The merits of mobile agent are described in figure 3.2 (Lange, D.B.,1998). The mobile agents can be dispatched into the network, thus the network can be disconnected from client in order to reduce the dependency on fragile network connections. After being dispatched, the mobile agents become independent of the creating process and operate asynchronously and autonomously. The client can reconnect later when the agent finishes the task. During the execution of the task, if the remote server is not responding due to some unpredictable reasons, the task agent can dynamically adjust its visiting plan, either changing its route or returning to its origin.

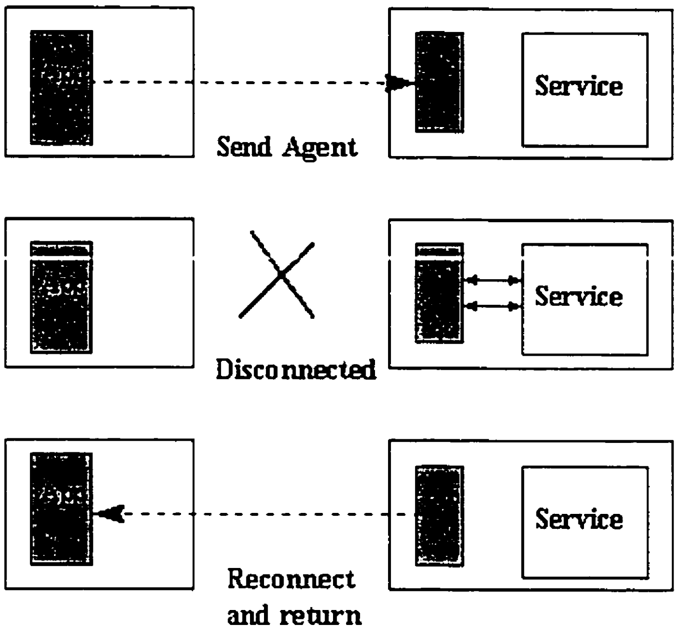

Figure 3.2 Disconnected Operation in a Mobile Agent System

Problem 1: Traveler Scenario

This scenario is an example of a disconnected network situation. The application is to verify whether the mobile agent can be a solution in supporting fragile network connections and mobile devices.

In the traveler application, an agent sends a travel agent to a remote site. The travel agent is required to fetch certain documents back. Once the travel agent finds the documents and prepares to come back, it may not able to find his original host because it is off from his work or the host is disconnected from the network. Then the travel agent has to wait on the remote site, check whether the original host is back frequently, and return to the original host with the document when the original host is back.

3.2.2 Flexibility and Simplicity of Server

Motivation

In a distributed application based on client/server structuring techniques, clients depend on the static application API (Application Program Interface) offered by a server. If the API is not rich enough, one problem exists. The client may need several interactions with the server to get a final result, which means more utilization time for network connections and the reliability of the network needs to be guaranteed during the transactions. If we place a rich API on the server which provides more services, the maintenance and any upgrading of the server API are going to be more difficult than the non-rich API server for programmers, especially when user requirements for server API services change over time.

Mobile Agent Solution

Mobile agent technology is claimed to solve the above basic API problems. The point here is that the server offers some basic APIs and Client expands the APIs by binding the mobile agent to server. That means the server can receive, install and execute the client-side code. Therefore, users will have the freedom to personalize server behavior and on the server side, the goals of flexibility and simplicity can be reached. The trade off of this approach is that a security model must be established to address behavioral concerns. For instance, the application of authentication to establish trust.

Although the idea of extended servers is still very controversial today, it is worth consideration because the flexibility it emphasizes is a very valuable new capability.

Problem 2: Watcher Scenario

This scenario is to install a server API using a mobile agent. The purpose of this application is to see whether mobile agents are a good way for enriching the services on the network.

There are times when changes in certain documents at a remote site need to be tracked. But on the remote site there is not this kind of service: for example, the ability to inform users about when changes happen to the documents. The question is, can mobile agent help with this?

A watcher agent sends a task agent to this remote site, and this task agent resides at this remote host, watching for any changes . If changes happen, the task agent will send a message back , notifying the watcher of these change.

3.2.3 Improved performance in Communication

Motivation

Currently, the network communication is dominated by the principle of the *remote procedure call,* which transfers the communication between two computers to procedure calls from one computer to another computer. This paradigm requires that both computers should understand the content of each message and the types of arguments transported by message, which means a protocol is needed for the communication. Another significant feature of this paradigm is that once the interaction between two computers begins, the two computers never stop communication until all interactions finish. Figure 3.3 illustrates the Remote Procedure Call mechanism (White, J.E., 1994). The client /server paradigm has a unique alternative to achieve these objectives, which is to raise the granularity level of the service offered by the server. That means, a single interaction between client and server is sufficient to specify a high number of low level operations, which are performed locally on the server without involving communication over the physical link. Nevertheless, this unique solution brings out other problems such as increased server complexity and size, as well as reduced flexibility of the server. More importantly, the upgrade and maintenance of the server become more difficult.

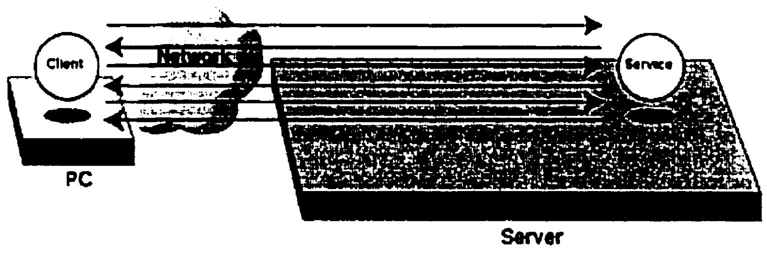

Figure 3.3 Remote Procedure Call

Mobile Agent Solution

Once again, mobile agents come to rescue. The approach allows one running procedure to continue its process after being sent to another computer. In another words, one running procedure can save its state and finish its tasks at a destination computer. And both computers understand instructions and data types of this procedure. as described in Figure 3.4 (White, J.E.,1994). Among the most compelling advantages that mobile agents can provide over traditional client/server computing are:

1. overall network traffic reduction
2. limited network latency.

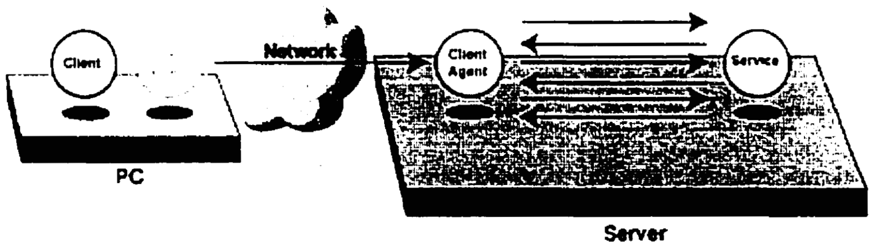

Figure 3.4 Mobile Agent in Distributed Computing

Problem 3: File Search Scenario

Reduced network traffic is often referred as a benefit potential for the mobile agent paradigm. The author is to verify this by using comparisons of three models: a mobile agent model, a traditional client/server model, and a RMI model. The file search application is designed to test this performance issue. In this comparison, three applications will complete exactly the same functionality: search a file on the remote host according to key words. The execution time to finish the search is the performance data collected to compare bandwidth usage.

3.3 Required Aglet System Properties

An agent system is a platform that can create, transfer and terminate agents. Agent applications are built on different agent system platforms. For the specific purposes of the thesis, the author has chosen IBM aglet as the agent system platform for implementing agent applications. From the functional view, this agent application should utilize the aglet system properties in three functional blocks: mobility, communication and security.

3.3.1 Mobility

A mobile agent is not bound to the system where it begins its execution, since it has the ability to move from one host in a network to another host.

- The system should support the creation of mobile agents. The creation of an agent means the instantiation of an agent class within a default place or a place specified by a user. After the creation takes place, the agent is assigned an unique identifier and must be initialized.

- The system should support the disposal of an agent. The disposal of an agent means that instance of an agent class ceases to exist and the agent will halt its current execution and be removed from its current place.

- The system will dispatch an agent from one place to another destination where it restarts its execution. The agent's state and code should be transported when agents travel. This functional requirement turns out to make the class transfer between hosts in a network. The source (sender) agent system sends all java classes necessary to execute the agent to another host.

- Each agent should posses its own thread of execution. As agents are often executed as parallel processes, threads are employed to keep the agents under control.

3.3.2 Communication

For the purpose of cooperation, mobile agents must establish communication relationships from time to time. The agents system prototype should offer a communication mechanism for both local and remote communication. There are various options for mobile agent communication, the messaging mechanism is widely adopted due to its efficiency and simplicity. IBM Aglets also uses messaging as its communication mechanism.

- The system should allow sending, receiving and handling message operations. These operations are basic messaging functions.

- The communication system should be able to identify a communication partner. For instance, a message sender should be able to identify the message recipients.

- The communication process should support both synchronous and asynchronous message passing. The synchronous passing of a message means that the message will block all current execution until the receiver has completed the handling of the incoming message. Asynchronous message passing means that the message will not block the current execution of the receiver. This requirement is needed for the purposes of disconnected operation.

3.3.3 Security

Despite mobile agent 's many practical benefits, the technology results in significant new security threats from malicious agents and hosts. So security factors will also be considered in this thesis, although they are not as important as the previous system properties.

- The system should be able to give privileges needed for an agent to carry on a task. A simple example is that an agent with read permission can not write to a specific file.

- The agent system should be able to identify the original host of the agent, and only agents from recognized hosts can access another host. This domain authorization is convenient for the implementation of the applications being examined and discussed in this thesis.

3.4 Requirements of Applications

3.4.1 General Requirements

- Requirement 1 The task agent should be able to go to any destination host assigned by the user.

- Requirement 2 The agents should be able to print out the results from the remote host on the screen.

3.4.2 Traveler Application

To achieve the specific functionalities for the traveler application, the mobile agent prototype must meet the following requirements:

- Requirement 3 The agent should be able to fetch specified documents from either the local machine or the remote machine.

- Requirement 4 The task agent which is dispatched to a remote host should be able to check the state of the original host if it wants to get back after it finishes its task. If the original host is still available, it goes back directly; if not, the agent can wait on the remote host until the original host is available.

3.4.3 Watcher Application

The specific requirements for the watcher application are:

- Requirement 1 The agent should be able to reside on the remote host to monitor a document; if the document gets modified, the agent should send back a message, notifying its owner of the change.

- Requirement 2 The implementation should be able to set the time interval for monitoring frequency.

3.4.3 File Search Application

For the file searcher applications, the specific requirements are:

- Requirement 1 The agent should be able to search files using key words at a remote host or a local host in any open directory. Text file retrieval is a major task for the agents in this application.

- Requirement 2 In client server and RMI implementations, the client should be able to send search request, including key word and search directory requests to the server.

- Requirement 3 In the client server model, the client should be able to download data from the server and implement a local search.

- Requirement 4 In the mobile agent implementation and the RMI implementation, the actual search should occur on the server

- Requirement 5 The three implementations should be able to record execution time for the file search operation.

- Requirement 6 The three implementations should use the same file operation class and search methods in order to reduce deviations.

- Requirement 7 The three implementations should also use the same time counting method in order to reduce deviations.

3.5 Summary

This chapter analyzes three problem situations and summarizes the functional requirements for the individual application. The requirements listed above are based on the overall objectives of the project which is to validate the applicability of the mobile agents.

Chapter 4 System Design and Result Analysis

4.1 The Overview of the System

This section describes the system architecture of the mobile agent applications and the client/server applications.

The mobile agent system mainly consists of three applications that were chosen to verify the applicability of mobile agents: Traveler, Watcher and File Reader. The Traveler application is designed to verify the ability of mobile agents that can travel on many different hosts, even when the original host is temporarily unavailable, the aglet can still return after the original host is connected to network again. The watcher is an aglet that can reside on a remote machine and act as an extra server application interface, notifying the owner of the changes occurring to a specific document. The file reader application is for comparing with the equivalent client/server application for performance criteria.

1. The traveler application is the implementation of problem 1 detailed in Chapter 3. The purpose of this application is to verify the ability to transfer information in the case of network trouble, such as a disconnection, and to identify whether the mobile agent is able to improve the fault tolerance and support information transmission in the network emergency.

2. The watcher application is intended to illustrate that mobile agents can function as an additional API on the server and show the extensibility and flexibility brought by mobile agents. This implementation is to implement the problem 2 in Chapter 3.

3. The file search application is designed for performance comparisons in three different models: client/server, mobile agent and Remote Method Invocation (RMI). It is claimed that mobile agents improve the system performance in terms of reducing network traffic. The comparison is not limited between client/server and mobile agent but includes RMI, because the RMI is a combination of traditional client/server paradigm and object oriented technology.

4.2 Traveler Application

4.2.1 Design: Master - Slave Pattern

The implementation work of this thesis utilizes the master-slave pattern. In fact, the master slave pattern is widely used in aglet implementations. The master slave pattern defines a scheme in which a master agent can assign a task to a slave. The reasons why many aglets implementations utilize the master slave pattern are two fold (Aridor, Y. and Lange, D.B., 1998) :

1. A master agent can continue to perform tasks in parallel with the slave agent. This feature offers convenience for agent applications because most agent applications require a communication center to which traveling agents can send back information, retrieve information from or exchange information with. In the master- slave pattern, the communication role can simply be replaced by a master agent.

2. A master -slave pattern can provide a GUI (Graphic User Interface) for inputting data and displaying results and messages from remote agents. If there is only one travel agent in the application, it is not possible to maintain a GUI after the agent leaves the original hosts.

The master slave pattern in aglet packages utilize the abstract classes of *Aglet* and *Slave*, which localize the invariant parts of delegating a task between the master aglet and slave aglet. The invariant parts include dispatching a slave aglet to a remote site and initiating the task's execution. The following figure (Figure 4.1) is a simplified class diagram for this design.

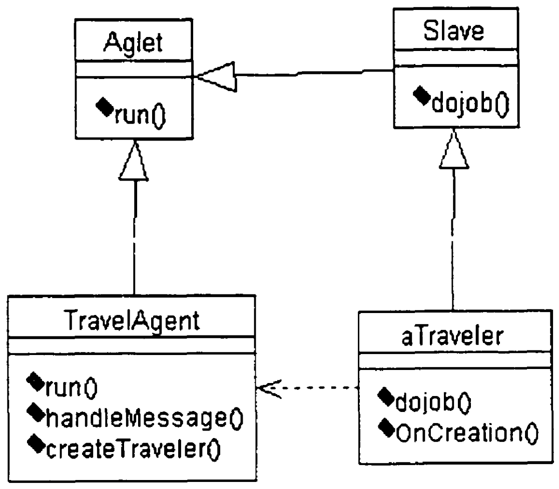

Figure 4.1 Class Diagram for Traveler Application

In this design, the TravelAgent class acts as a master class and aTraveler class is a slave class. The travel agent is inherited from the *Aglet* class and creates the traveler and sends it to various destinations to finish a file reading task. The traveler is a task aglet and is inherited from slave class. The dotted line from aTraveler to TravelAgent indicates the dependency relationship. A traveler has to go to the destination according to the Travel Agent's instructions. It should be noted that, for the sake of simplicity, the methods and classes shown here are not complete.

4.2.3 Implementation Result Analysis

The implementation of this application is done on two Windows NT 4.0 workstations, with 64 MB RAM and Pentium 200 Mhz CPU. The application runs on the Aglet 1.1

beta 1 and Jbuilder 2. One of the intentions of this work is on verifying the support which agents can provide for the mobile client or disconnected operation. A travel agent creates a traveler which needs to perform a file access operation on a remote host. After the traveler is sent out to the remote host, the connection between the sender host and receiver host is cut. When the traveler finishes its task, it tries to return the sender host which is now not available. The traveler then tries to dispatch itself back periodically. Once the original host is back, the dispatch operation is considered successful. The application shows the traveler can successfully return from the remote host to the original host after a disconnect and a re-connect.

The application verifies the claim that agents can provide better support for the mobile clients such as laptop and notebook computers, as well as for thin client devices like personal communicators. There are several convincing technical features demonstrated here, namely:

1. The mobile agent solution tends to be more asynchronous in nature. Once the agent is dispatched, the sending host can continue to work on other tasks until the agent comes back. Unlike traditional client/server, the connection between two hosts can be closed during the process until it is needed again. The asynchronous computing model supported in the agent execution environments enables the agent to be less dependent on the length of the network sessions.

2. The mobile agent can perform both information retrieval and filtering on the network nodes where the needed resources resides, which will relieve the resource usage on the mobile devices. Usually laptops and personal communicators have limited storage and processing capacity.

3. Enhanced fault tolerance. Due to weak network connections, some network transactions could lost ; the mobile agent application proved to be efficient in this case.

The problems appearing during this application are:

1. It is argued that each host of the entire network needs to have an agent server installed to meet the mobile agent application needs. In the application, both the original host and remote host need to install an aglet server. If this is the case, the computational resources required by the execution environment of agent applications is a negative factor for the deployment of the "agent " technology.

2. The fundamental problem is how vital the "agents" are to solve the particular mobile clients problem. It is argued that the problem can be solved by using other mechanisms such as remote shell facilities or even manual operations. The author could not find any strong arguments that are only applicable to agents.

4.3 Watcher Application

4.3.1 Design: Notifier Aglet

The Notifier is the class of aglet pattern package. It is widely used for agent applications and can be seen as special slave aglet implementation. The benefit of the notifier is that it provides an easy way to extend the server application interface, in another word, to install client software on a remote host.

The following is a class diagram (Figure 4.2) of the application:

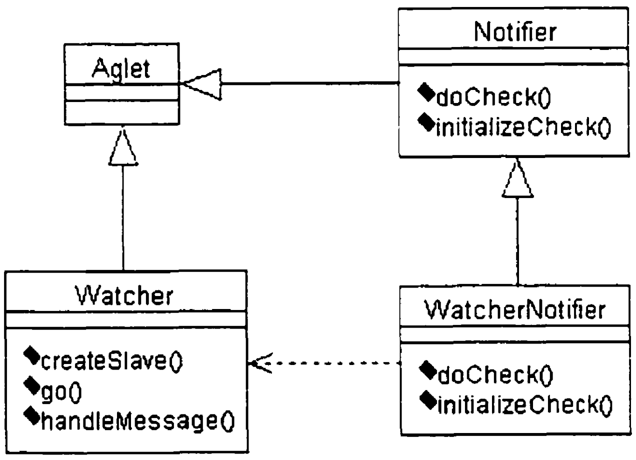

Figure 4.2 Class Diagram for Watcher Application

4.3.2 Implementation Result Analysis

The Watcher application runs on the same workstation as described in the traveler application. Watcher is an aglet-based implementation for monitoring a specific file update status. It consists of two aglets, one stationary watcher aglet and one mobile watcher notifier aglet. The latter is dispatched by the watcher aglet to a remote location where it stays and keeps watching a file being updated. Watcher Notifier usually sleeps and periodically gets up and checks if the file was updated or not. Whenever the file is updated, the Watcher Notifier notifies the Watcher.

The application verifies that the mobile agent can go to the remote host and provide notifying service to its master agent. If the specific document is updated, it will send back a message, indicating the modification time. The implementation illustrates the following points:

1. Agents are a convenient way to install software on remote hosts, in another word, using agents as an extensible server has the potential to be implemented widely. This benefit perhaps is the most appealing feature in some proposed application models (Johansen, D., 1998):

    - Server API patching: the agent is installed as a patch to provide a new API. The original host can access this remote agent through the regular network communication interface. The installation is more or less permanent and agents can act periodically or around the clock.

    - Task Agent: the agent is dispatched to a remote host on behalf of the sending host, and either returns the potential results to the sending host or continues traveling on the network according to a route plan.

Some concerns about the applicability of this remote sever application exist:

1. The first question is whether the agent is the only solution in the extensible server application. The answer is no. Some old techniques also provide the structured facilities such as remote shells and remote evaluations. There are some new ideas such as extensible kernel which utilize process migration at the system kernel level. (Johansen, D., 1998)

2. The second question is the performance issue. Does the agent solution offer improved performance among those alternatives such as remote shells? There is no obvious evidence which can provide convincing results.

3. Security is the key. In most of existing system security implementations, there are several underlying assumptions (Chess, D.M., 1998):
   - Most of the systems assert that the intensions of users are identical to the action of the program. That means when a program attempts some actions, users usually have the rights to decide whether or not the action should be taken by checking the details of the program.

- Some systems specially mark the rights granted to some users for running the program. This assumes only persons who are known to system can execute programs.

- All programs are obtained from easily identifiable and generally trusted sources.

Mobile agent systems violate all assumptions laid out above:

- Since the mobile agent programs may not reflect the intent of the user, we can not determine whether or not the actions should be taken.
- An unknown user may run mobile agent programs if they can obtain the mobile agent services from the network.
- In mobile agent systems, many programs may be from unknown sources.

There are three major security issues in the mobile agent system:

- Protecting the host from malicious agents.
- Protecting an agent from malicious hosts.
- Protecting one agent from another agent during their interactions.

4.4 Performance Comparison: File Searcher Application

4.4.1 Application Design

The file search implementation in the thesis is to compare the performance in three different modes: mobile agent, traditional client/server model, and remote method invocation (RMI). It is claimed that agents potentially improve the performance in terms of bandwidth usage. A simple straightforward file searcher application is ideal for performance comparison.

In a distributed environment, it is common to store data in a server, and client computers pull data for local processing. The client/server implementation is designed in this way. The RMI implementation invokes the search method from the client, conducts searching on the server and return results to the client. The agent application ships an agent over the network to the destination host, completes the task on the host, and return the results to the original host.

The reason for using RMI as another implementation to compare performance is manifold:

- RMI is still considered as a client/server technology.
- RMI is not limited to traditional client/server and it also provides the ability to transfer objects to multiple servers.
- RMI uses techniques such as object serialization and dynamic class loading which are also fundamental technologies used by mobile agents.

For the purpose of consistency, the author employs the same Java classes and methods in three models in order to avoid deviation caused by actual code. The file search functionality is fulfilled by methods in class java.io.RandomAccessFile.

The author implements the timing using System.currentTimeMillis() to measure the response time between two workstations. In the aglet application, the response time is measured according to the following criteria:

1. The measurement begins with dispatching the file search aglet to the network.

2. The response time includes the time of installing and running the aglet on the destination workstation.

3. The measurement also includes the time spent by the aglet on returning to the host workstation and printing out the results on the screen.

For the client/server implementation, the end to end response time is measured in the following way:

1. The measurement starts from client downloading data from the server.

2. The measurement includes the time of processing data on the local client workstation.

3. The time of printing results on the screen is also included.

For the RMI implementation, the response time includes the following time frame:

1. The time counting starts with initiating a search call to a remote object.
2. The time for processing includes: the stub transfers arguments to the reference layer, the reference layer passes on the request to the transport layer, then the skeleton reads the arguments from the reference layer in the server side, and the skeleton makes a call to the actual remote object implementation.
3. The results are returned in the same way but opposite direction, and the program prints out search results on the screen.

4.4.2 Implementation Result Analysis

The following is a table (Table 4.1) of experiment results, indicating the average response time for various data size in the three different applications. The experiments are conducted on two Windows NT 4.0 workstations with 128 M byte RAM and 200 Mhz CPU . The two workstations are connected by a 100 Mbyte Ethernet. This work reports the experiment results from 12 samples. The response time data in each cell is the average value of three experiments conducted with same size data.

Data Size (MB)	C/S Response Time (s)	M/A Response Time (s)	RMI Response Time (s)
0.62	34.54	11.57	9.83
1.19	69.12	21.48	19.52
1.83	105.76	31.80	29.95
2.48	142.97	41.55	40.46
3.12	180.06	51.69	50.85
3.78	217.54	61.82	61.83
4.39	251.18	70.93	71.48
7.57	437.62	122.50	121.87
11.4	657.83	182.18	182.96
15.1	887.94	250.24	248.04
18.9	1102.87	310.51	299.49
20.1	1177.33	325.80	319.03

Table 4.1 Experiment Data in Three Models

From the performance data illustrated in the above table, we can see that the mobile agent model and RMI model outperform client server model a lot and there is not much difference between the mobile agent implementation and the RMI implementation. This is shown clearly in the following figure (Figure 4.3):

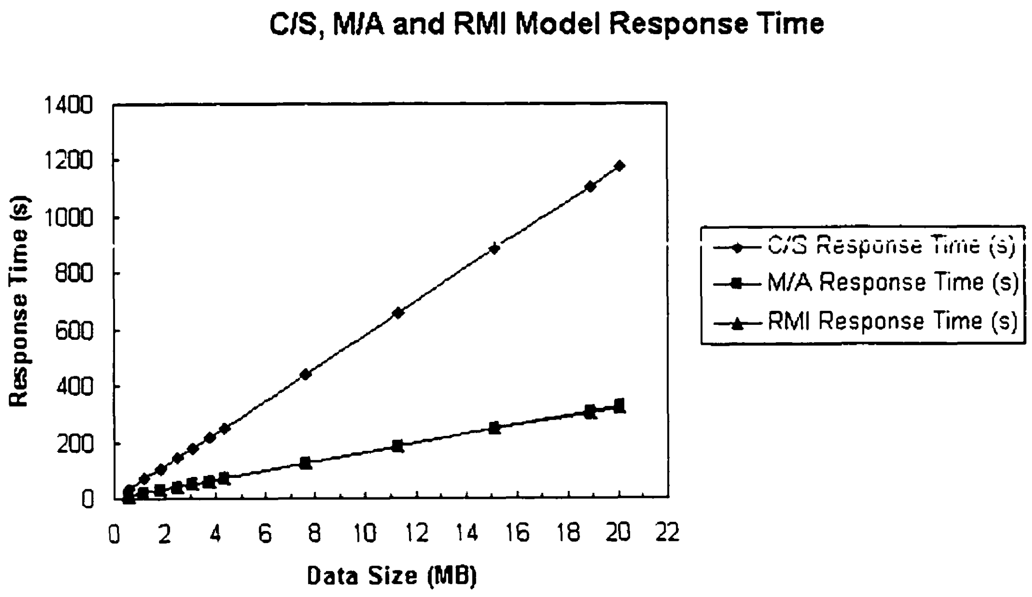

Figure 4.3 Response Time for Three Models

- From the comparison results, the mobile agent approach really seems to reduce the network traffic. This is not surprising, since in the client/server application, it has chosen the worst-case scenario in which all the data must be carried across the network to the client for processing. Now client/server technology is more flexible than the case chosen and a rich server API can help relieve this situation. For instance, in some client server applications, the server handles data processing and returns the results to the client.

- The RMI model also can achieve the same effects as mobile agents. Almost in all cases, the RMI model even outperforms the mobile agent model. How to define the RMI technology is a little bit controversial. RMI is basically Remote

Procedure Call's evolution and in this sense, it is a client server technology; but it also does a couple of additional things:

- o RMI performs network dynamic class loading. It is intended for loading the stub classes for the remote objects. That means, if the client host does not have the code for the remote proxy, RMI can send them to the client.

- o RMI uses standard serialization technology to pass the arguments of the method.

Both of these feature make RMI "kind of" mobile agent technology. Whether it "is" a mobile agent technology brought up the controversy and the argument that is still going on. Depending on how one defines "mobile agent", some people view RMI as a mobile agent technology. In this thesis, the author does not want to continue the argument but just wants to verify that RMI provides the ability to transfer an object to a server and reduce the network traffic. From the experiment results, we know that pure mobile agent technology is not the only way to improve the performance.

- In the three cases, the same information about the remote host are required, such as the location of the files the user wants to search through, or how to connect to a database on the host etc. Mobile agents can not work that out for themselves. From this view, mobile agents do not offer any advantages.

- In the response time measurements of the RMI implementation and the mobile agent implementation, the dominant factor has been the serialization/deserialization delay and not the transmission delay. The Aglet uses standard Java Object Serialization to marshal and unmarshal the state of agents and tries to marshal all the objects that are reachable from the Aglet Object. RMI also employs the standard Java Object Serialization but usually it just serializes/deserializes arguments or returns a value. So this may be considered as the major reason that affects the response time. Of course, there are several other factors that may affect the transmission delay (apart from the transmission rate) such as the transmission protocol. RMI uses the TCP while Aglet uses ATP (Agent Transfer Protocol), which is an application level protocol above TCP/IP.

4.5 Summary

The chapter validates the mobile agent applicability by three different applications: traveler, watcher and file searcher. These three prototypes and their designs are used to better understand mobile agent applicability aspects. From the implementations, some conclusions are drawn:

- The Traveler application shows that the mobile agent can be off loaded from network in a controlled manner, which means mobile agent do improve the fault tolerance and support information transfer in temporally disconnected network.

- The Watcher application indicates that mobile agent is a convenient way to install client specific software even though security is a big concern. The extensible server is a promising feature offered by mobile agents in a sense that mobile agents is an easier and more straightforward way to implement this functionality than other approaches.

- From the implementation results, it is not very convincing that the improved performance in terms of bandwidth usage is a good reason to deploy mobile agent technology.

Chapter 5 Conclusions

This chapter draws conclusions from the research work and evaluates future possible research directions related to the thesis. The evaluation addresses the objectives of the research and explains how these objectives were reached by the thesis. The evaluation also discusses the contributions of this research work compared to the contemporary research work in this field.

5.1 Addressing the Aim and Objectives

The aim and primary objectives of the research work were stated in Chapter 1:

Aim:

The aim was to develop an in-depth understanding of the mobile agent mechanism by designing and implementing mobile agent based applications, and to evaluate the applicability of mobile agent applications based on comparisons between the client/server mechanism and the mobile agent mechanism.

Objectives were:

- To broadly review literature on mobile agent applicability.

- To select typical application scenarios which demonstrate advantages of mobile agents over client/server.

- To develop a mobile agent based application as well as a client/server based application for the purpose of performance comparisons.

- To carry on the evaluation of the applicability of mobile agents based on the application design, implementation result analysis and performance.

Chapter 2 gave a brief introduction about mobile agents and the current state of this research field. The disadvantages and controversy surrounding mobile agents were also discussed. Based on the background introduction, typical application scenarios were selected:

- Traveler application verified the supportive role a mobile agent can play in the emergent event of lost network connection. The improved fault tolerance a mobile agent can offer in the fragile network connection is often referred as a major credit of mobile agent paradigm.

- Watcher scenario was chosen to verify that whether mobile agent technology is a good way to enrich services on the network. A mobile agent is able to function as an additional service provider on the network. This advantage is a main claim from the supporter of mobile agents.

The above two application scenarios are both typical mobile agent applications, demonstrating the major two advantages over client server paradigm. So objective 2 was accomplished.

Objective 3 was reflected in the File Searcher application scenario. Reduced network traffic is often referred to as an advantage brought by mobile agents. The performance comparisons between traditional client/server, JAVA RMI and mobile agents were to verify this claim.

Chapter 3 discussed the requirements for the three application scenarios and Chapter 4 discussed the design and the implementation of the applications. Based on the design and the implementation of those three applications, the evaluation of mobile agent applicability was realized, which satisfied the last objective, and the major conclusions from the experiments are:

1. Small, resource limited computing devices may not be able to remain "connected" for the duration of services. The mobile agent solution tends to be more asynchronous in nature than the traditional client server technology; since mobile agents act asynchronously without requiring a permanent connection.

2. Agents are a robust way to facilitate and automate information distribution; and they are also convenient for installation of services on a remote host, in other words, using agents as extensible servers has the potential to be deployed widely.

3. The support of high bandwidth interaction is the most often cited reason to use mobile agents. From the author's work, it is hard to prove that the reduced network traffic is a major advantage of mobile agents. For example, we can see that Java RMI can actually create very marginally improved results than mobile agents. Reduced network traffic is not a convincing reason for people to choose mobile agents instead of the client server paradigm.

4. Since most of current mobile agents are written in relatively slow interpreted languages like Java for portability reasons on the internet, current mobile agent systems save network bandwidth at the cost of higher loads on the host machines. The mobile agent applications require computational resources such as the agent server, which will increase the difficulty of deploying this technique widely. As mentioned before, in mobile agent applications, the hosts visited by the agents require an agent server, which is an additional resource load to the machine.

5. The most important non-technical problem with the mobile agents is that there is no "killer" application for mobile agents. Though the mobile agent is a new design paradigm in many respects, most applications can be implemented with traditional techniques. The author can not see a convincing reason for applying mobile agents from each of the experiments conducted. For example, a mobile agent is not the only way to support fragile network and to extend server services.

Some old techniques like remote shells are also widely used by people. It is really hard to find an example that there is something that can be done with mobile agents which can not be done by other means.

6. Security problems are a major concern in mobile agent technology. The agent execution environment needs a high level of security management to assure users of the safety of their data and their privacy.

The above description clearly shows that the objectives of the thesis have been accomplished. Based on the process of the whole thesis work, the overall aim is also reached.

5.2 Contributions

Mobile agents have been an intensive focus recently. With the emerging understanding of this technology, people are trying to dig out the potential usage and deploy the new paradigm quickly. Despite the popular support for this new distributed computation paradigm, there are lots of arguments around mobile agents. The controversy focuses on its applicability. People are asking whether it can be a successful replacement for client/ server technology.

Though there are many articles for the mobile agent and agent technology, many of them are trying to build an agent architecture and infrastructure, facilitating the application building, and modeling agency. We still only have mainly qualitative arguments for the benefits of mobility. There are relatively few quantitative results out there. And no real solid argument for why mobility is any more than 'a good thing'. Very few articles actually try to validate the applicability of mobile agents.

Contributions of the thesis work are:

- Built mobile agent based system prototypes to verify the mobile agent applicability potentials.

- Made performance comparisons based on bandwidth usage for traditional client/server paradigm, Java RMI and mobile agent paradigm, and get quantitative results to verify the mobile agent benefit argument.

5.3 Future Directions

5.3.1 System Extensions

For the study, three typical mobile agent applications are chosen and simple system implementations show something that mobile agents are really good at. The experiments based on these applications focus on the objectives of the research. Though the research is complete, there is still much room to improve.

It would be interesting to evaluate a more complete and complex mobile agent system to get a more generalized idea . Even a real life study would be worthwhile. For example, an agent system with a broker or mediator architecture can be a good example, since it represents a widely-used agent system structure or an agent system with more agent interactions and communications.

5.3.2 Performance Comparisons

In the research, the communication effects on the network traffic caused by mobile agents has been verified. The evaluation for bandwidth usage is measured by the response time from sending request to receiving results in three models. There is more to be measured than just bandwidth usage in order to show the complete impact to the system caused by mobile agents. For example, there is an impact that mobile agents and mobile agent servers have on a host while a mobile agent is executing. The impact of a single mobile agent maybe negligible, but once large number of mobile agents are performing transactions, then we need to consider the requirements in terms of CPU, I/O and memory. So a more accurate and detailed study can be conducted.

5.4 Summary

The thesis describes the research work that evaluates the applicability of mobile agents, based on the application scenarios and performance comparisons. This chapter concludes the thesis work, summarizes the contributions and points out the future directions of continuing research work.

References

Aridor, Y. and Lange, D.B., Agent Design Pattern, Proceedings of Autonomous Agents '98, see http://www.genmagic.com/asa/danny/, Minneapolis, Minnesota, USA, ACM Press, May 1998

Bradshaw, J.M., Software Agents, AAAI Press/The MIT Press, Menlo Park, California, 1997

Carzaniga, A., Picco, G.P., and Vigna, G., Designing Distributed Applications with Mobile Code Paradigms, Proceedings of the 19th International Conference on Software Engineering, ACM Press, Minneapolis, Minnesota, 1997

Chess D., Grosof, B., Harrison, C., Levine, D., Parris, C., and Tsudik, G., Itinerant Agents for Mobile Computing, in Readings in Agents, Huhns, M.N., Singh, M.P., (eds.), pp. 267-282,1997

Chess, D.M., 1998, Security Issues in Mobile Code Systems, in Mobile Agents and Security (edited by G. Vigna), pp.1-13, Springer, 1998

Finin, T., Technical Report, see http://www.cs.umbc.edu/agents/ , 1994

Franklin, S., and Graesser,A., Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. See http://www.msci.memphis.edu/~franklin/AgentProg.html. Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, Berlin, 1996.

Fuggetta, A., Picco, G.P., and Vigna, G., Understanding Code Mobility, IEEE Transactions on Software Engineering, VOL.24, 1998

Ghezzi, C. and Vigna, G.,1997, Mobile Code Paradigms and Technologies: A case Study, Proceedings of the First International Workshop of Mobile Agents, pp. 39-49, Springer, Berlin, Germany, April, 1997

Harrison, C.G., Chess, D.M., Kershenbaum, A.K., Mobile Agents: Are they a good idea? IBM Research Report, see http://www.cetus-links.org/oo_mobile_agents.html ,1995

Huhns, M.N., Singh, M.P., Readings In Agents, San Francisco, CA, Morgan Kaufmann, 1998

IBM Tokyo Lab, see http:/./www.trl.ibm.co.jp/aglets/spec11.html, 1999

Jain, A.K., Aparico M., and Singh M. P., Agents for Process Coherence in Virtual Enterprise, pages 62-69, March 1999-Volume 42, Communications of the ACM, 1999

Johansen, D., Mobile Agent Applicability, 1998, Proceedings of Second International Workshop, MA'98, , Rothermel, K. and Hohl, Fritz, (eds.), Springer, 1998

Kiniry J. and Zimmerman, A hands-on look at Java Mobile Agents, IEEE Internet Computing, July/August, 1997

Knapik, M. and Johnson J., Developing Intelligent Agents for Distributed Systems, McGraw-Hill, New York, 1998

Lange, D.B., Mobile Objects and Mobile Agents: The Future of Distributed Computing? see http://www.genmagic.com/asa/danny/, 1998

Lange, D.B., Present and Future Trends of Mobile Agent Technology, see http://www.genmagic.com/asa/danny/, 1998

Lange, D.B. and Oshima, M., Seven Good Reasons for Using Mobile Agents, Communication of ACM, pp. 88-89, March, 1999

Milojicic, D., Breugst, M., Busse, I., Campbell, J., Covaci, S., Friedman, S., Kosaka, K., Lange, D., Ono, K., Oshima, M., Tham, C., Virdhadriswaran, S., and White. J., see http://www.fokus.gmd.de/research/cc/ima/masif/entry-old.html, 1998

ObjectSpace Voyager, General Magic Odyssey, IBM Aglets: A Comparison by ObjectSpace, ObjectSpace, 1997

Petrie, C.J., Agent-Based Engineering, the Web, and Intelligence, see http://www-cdr.stanford.edu/NextLink/Expert.html, IEEE Expert, December, 1996

Rothermel, K., Hohl, F., (eds.) Mobile agents : Proceedings of the Second International Workshop, MA '98, Stuttgart, Germany, September 9-11, Springer, 1998

Rus, D., Gray, R., and Kotz, D., Transportable Information Agents, in the International Conference on Autonomous Agents, Feb, 1997, in Readings In Agents, pp.283-291, Huhns, M.N., Singh, M.P., (eds.), 1997

Stamos, J. and Gifford, D., Remote execution in ACM Transactions on Programming Languages and Systems, 12(4): pp. 537-565, October, 1990

Sun Microsystems, Inc. see http://java.suun.com/products

Vaughn, L., Client/Server System Design & Implementations, McGraw-Hill Inc. New York, NY, 1994

Vigna, G., (ed.),   Mobile Agents and Security, Springer , 1998

White, J.E., Mobile Agents, see
http://www.genmagic.com/technology/techwhitepaper.html , 1994