INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

A Multimedia Transportable Agent System

by
F.J.Ziade

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

M.A.Sc
in
Electrical Engineering

Ottawa-Carleton Institute of Electrical and Computer Engineering

Department of Electrical & Computer Engineering
Faculty of Engineering
University of Ottawa
Ottawa, Ontario

September 1999

Table of Contents

Abstract vii
Acknowledgments ix
Acronyms x

Chapter 1 Introduction 1

1.1 Motivation 1
1.2 Objectives 5
1.3 Thesis Outline 6
1.4 Main Contributions 7

Chapter 2 Classification of Agents 9

2.1 Introduction 9
2.2 Properties of Agents 13
2.3 Classification of Agents 15
2.4 Personal Agents 16
2.5 Service Agents 20
2.6 Multi-Agent Systems 21
2.6.1 Coordination in Multi-Agent Systems 22
2.6.2 Negotiation in Multi-Agent Systems 24
2.6.3 Agent Communication Language(ACL) 25
2.7 Mobile Agents 26
2.7.1 Introduction 26
2.7.2 The Mobile Agent Paradigm 28
2.7.3 Advantages of Mobile Agents 29
2.7.4 Issues Raised by the Mobility of Agents 31
2.8 Existing Mobile Agent Systems and Tools 32
2.8.1 Open Languages 32
2.8.2 Proprietary Languages 33
2.8.3 Mobile Agent Systems 35

Chapter 3 Multimedia Information Systems and Mobile Agents 40

3.1 Introduction 40
3.2 The MEDIABASE Project 42
3.3 The News on Demand Application 42
3.4 The Multimedia News Agent System 45
3.4.1 Architecture 45

3.4.2 The Static News Agent System....46
3.4.3 The Mobile News Agent System....47
3.4.4 The Mobile News Agent Implementation....48
3.5 The Multimedia Transportable Agent System....49
3.6 Global Architecture....50
3.7 Layered Architecture of the Agent System....53

Chapter 4 Multimedia Transportable Agent Platform....57

4.1 Introduction....57
4.2 The Agent Structure....58
4.2.1 The Transport Header (TH)....59
4.2.2 The Agent Header....62
4.2.3 Summary....67
4.3 The Multimedia Agent Platform Architecture....68
4.4 The Migration Facilitator....70
4.5 The AEE....72
4.6 Implementation....74
4.6.1 The Migration Facilitator....75
4.6.2 The Reception Manager....77
4.6.3 The Departure Manager....89
4.6.4 The AEE....91

Chapter 5 Multimedia Agent-Based Applications....94

5.1 Introduction....94
5.2 The Service Agents....95
5.3 The Agent Communication Language....96
5.3.1 New Tcl commands in the ACL....99
5.4 The GUIs and Mode of Operations....101
5.4.1 The Agent Properties....103
5.4.2 The Choice of an Application....104
5.4.3 The Agent Constructor....106
5.4.4 The Agent Activation, Cycle and Return of Results....108

Chapter 6 Conclusion....112

6.1 Summary....112
6.2 Suggestions for Future Work....114

List of Figures

Figure 2.1 General architecture of information filtering agents.....18

Figure 2.2 General architecture of information retrieving agents.....19

Figure 2.3 Architecture of the Visitor Bot agent system.....21

Figure 2.4 Architecture of centralized multi-agent systems.....24

Figure 2.5 The Remote Procedure Call model.....27

Figure 2.6 The mobile agent communication model.....28

Figure 2.7 The Telescript architecture model.....35

Figure 2.8 The architecture of the ARA system.....36

Figure 2.9 The Agent migration in Tacoma.....38

Figure 2.10 The layered architecture of Agent-tcl.....39

Figure 3.1 The architecture of the News Application.....43

Figure 3.2 The main interface of the News Application.....44

Figure 3.3 The layered architecture of the agentized news system.....46

Figure 3.4 The general architecture of the Multimedia Transportable Agent System.....51

Figure 3.5 The layered architecture of the agent system.....54

Figure 4.1 The agent structure.....59

Figure 4.2 The agent migration.....59

Figure 4.3 The Transport Header 1 (TH1).....60

Figure 4.4 The Transport Header 1 (TH2).....61

Figure 4.5 The agent header.....63

Figure 4.6 The UI header.....63

Figure 4.7 The MD header.....64

Figure 4.8 The AP header....65

Figure 4.9 The AS header....66

Figure 4.10 The complete agent structure....68

Figure 4.11 The functional architecture of the Multimedia Agent Platform....69

Figure 4.12 The flow chart of the migration facilitator for incoming agents....71

Figure 4.13 The flow chart of the migration facilitator for departing agents....72

Figure 4.14 The flow chart of the AEE....74

Figure 4.15 Hardware set-up for the implementation....75

Figure 4.16 Main program of the Reception Manager....80

Figure 4.17 Implementation of the init_WinSock_DLL_Version() function....81

Figure 4.18 Implementation of the init_TCP_Socket() function....82

Figure 4.19 Implementation of the wait_for_client() function....84

Figure 4.20 Implementation of the set_server_connection() function....84

Figure 4.21 Implementation of the read_TH1() function....85

Figure 4.22 Implementation of the read_TH2() function....86

Figure 4.23 Implementation of the read_file() function....88

Figure 4.24 Implementation of the inform_AEE() function....89

Figure 4.25 Extract from the implementation of the Departure Manager....91

Figure 4.26 The simplified agent header....92

Figure 4.27 Extract from the implementation of the AEE....93

Figure 5.1 The extented Tcl interpreter....97

Figure 5.2 The implementtion of a new Tcl command....99

Figure 5.3 The main interface of the agent system....102

Figure 5.4 The error generated if trying to build an agent before specifying the mission.....103

Figure 5.5 The agent properties entry window.....103

Figure 5.6 The entry windows for the List and Search applications.....105

Figure 5.7 The file manager appearing when the "Send Document" is activated.....106

Figure 5.8 The presentation of the list of files found during the "list" mission.....109

Figure 5.9 The file manager showing the results of the "Search" mission.....110

Figure 5.10 The file manager appearing during the "send" mission to remote users.....111

List of Tables

Table 2.1 Classification of currently existing agents.....15

Table 4.1 TF possible values.....61

Table 4.2 The primitives of the migration facilitator reception's manager.....78

Abstract

Computer networks have progressed rapidly in the late eighties and nineties: Intranets and public networks like the Internet have gained more and more popularity, demand and growth. Electronic information and services available through those networks have accordingly been growing at an explosive rate. The growth rate has by far exceeded the network speed especially with the integration of various medias like images, audio and video. As a result, the simple task of accessing, managing and even locating information and services in vast networks has become a tedious job. The trend in today's communications is to find new genre of technologies that would assist the user efficiently in roaming, using and managing the electronic services. Among the new technologies, software agents are gaining more momentum and seem to be a promising solution and a possible way out of this dilemma.

This thesis addresses the emerging field of mobile agents. The mobile agent communication paradigm is viewed as the extension of the client/server communication model. Mobile agents are self-contained objects that travel autonomously in a computer network, executing on different machines until their assigned task is solved. Mobile agents offer many advantages over the client/server model such as asynchronous operation and support for mobile users and lightweight devices but also new issues and challenges such as security and privacy.

A multimedia mobile agent system is designed and implemented to asynchronously execute tasks on behalf of the user. The mobile agent system is composed of a multimedia agent platform and agent-based applications. The agent platform provides a migration facilitator, an agent execution environment, an agent transport protocol and a simple API to the basic agent

operations such as *build* and *jump*. The agent-based applications are built on top of the agent platform and offer asynchronous services such as *search*, *retrieve* and *send* multimedia documents.

Acknowledgements

I would like to acknowledge the guidance which I have received from my thesis supervisor Dr. Ahmed Karmouch throughout the program. Dr. Karmouch exposed me to the latest technologies and adviced me throught all the stages of my thesis.

I would like to dedicate this thesis to my parents and family which supported and encouraged me all along. Thank you.

Acronyms

ACL	Agent Communication Language
AEE	Agent Execution Engine
AI	Artificial Intelligence
API	Application Programming Interfaces
ARA	Agents for Remote Action
ATM	Asynchronous Transfer Mode
BSD	Berkeley Software Distribution
DAI	Distributed Artificial Intelligence
GUI	Graphical User Interface
IP	Internet Protocol
KIF	Knowledge Interface Format
KQML	Knowledge Query Manipulation Language
LAN	Local Area Network
PDA	Personal Digital Assistant
RPC	Remote Procedure Call
TCL	Tool Command Language
TCP	Transmission Control Protocol
TH	Transport Header
TK	Toolkit
WAN	Wide Area Network

Chapter 1
Introduction

1.1 Motivation

We have witnessed in the last decade a tremendous change in the world of computing. Ten years ago, computers had limited access to networks and remote services. Today, millions of users worldwide have started accessing public networks for electronic information and services. The Internet, for instance, has swelled to about 40 million users and is continuing to grow at an increasing rate [THO95]. The electronic data and services offered on such wide networks have also progressed respectively from simple text files and command line applications, that allow data browsing, to advanced multimedia distributed applications such as videoconferencing and

electronic commerce.

The wireless networks also emerged in the eighties and nineties. The wireless networks include one-way paging, cellular data, wireless packet, specialized mobile radio, etc. This growth has been caused by multiple factors [ODO94]. Firstly, the increasing competition in the latest economic recession forced corporate employees to leave their headquarters, pursue new business deals, meet their customers face to face and offer support for their products at the customer's locations. The Bureau of Labor Statistics reveals that, the United States alone has close to 50 million people who are involved in occupations away from their "home base" for a long period of time and who could benefit from mobile communications. Secondly, increased dependence on palmtops, laptops and Personal Digital Assistants (PDAs) contributes to the growing demand for wireless data services. The wireless data can be delivered over three different coexisting technologies: radio, cellular and satellite. These technologies cover all of North America and the provider can select the best method to deliver the data to the subscriber. Thirdly, wireless Local Area Networks (LANs) and Wide Area Networks (WANs) are also gaining a great deal of attention [BRG94]. Wireless LANs are considered as "in-building" solutions. They can replace existing LANs or, in other cases, can act as an enhancement for an existing network to serve specialized applications, especially in open environments.

The wireless data market has existed for a number of years over primarily private networks aimed at blue collar applications. The strategy of the vendors today is to target a broader variety of applications (i.e. white collar) thereby deploying wireless data networks in more enterprises: universities, software companies, etc. This could be the first step in the

appearance of the public wireless networks, which will connect the private wireless networks, into a large public one.

The trends in today's communications are the following:

- the integration of the public and wireless networks as they are being installed. This integration has been inspired by the vision "information at anytime, anywhere and in any form" [MAG96].
- The development of more intelligent software that are user friendly and easy to use since more and more everyday tasks are computer-based and increasingly more users are untrained.
- The development of more personalized applications that could asynchronously perform tasks on behalf of the user. Computer users spend today a tremendous amount of time locating the appropriate information and using the electronic services in the public networks. This will only get worse with the explosive expansion of these public networks worldwide.

Although very desirable, the demand for the above three requirements in today's technologies has been very slow. The following three prominent factors have been identified for slowing down this demand: the cost factor, the performance limitations, and the lack of features.

In reality, the slow deployment of the wireless networks and the shy appearance of more personalized intelligent asynchronous applications goes back to the nature of the applications and communication protocols offered by the vendors over the new telecommunications infrastructure. Indeed, despite the gigantic change in the information infrastructure, the

applications have not changed. The communication model is in fact still based on the client/ server paradigm, introduced in the seventies, in which client applications request services from a non-local server process. In this context, the software requirements to make full use of the information infrastructure falls today into two categories. The first includes more personalized and adapted applications to each user connected to the public networks. The second involves new flexible communication model which allows asynchronous interaction with non-local servers.

In the search for a solution for the above problems, the agents paradigm receives considerable attention and seems to be a promising option. Agents are intelligent processes that act on behalf of the user. Used in numerous applications and domains, such as telecommunications, medicine, and Artificial Intelligence (AI), various classes of agents have surfaced. From a software perspective, agents are computational processes, inheriting all the attributes of AI agents and can be divided into two main classes: static and mobile. Static agents are stationary processes that assume the role of personal assistants or advisors and can be customized to each user. Mobile agents are complex objects able to travel in a network, running chunks of code on different nodes, executing tasks on behalf of the user and therefore interacting asynchronously with non local servers.

The University of Ottawa's Multimedia Information Research Laboratory (MIRLab) studied various aspects of distributed multimedia systems like multimedia synchronization, database models, storage and retrieval, document architectures, video browsing and indexing. Within the above studies, several protoypes and applications have been implemented such as the

Media Player [ROD95] and the News Application [FAL95]. The deployment of agent technologies in the broadband environments is believed to have a great impact on the next generation applications. The agent paradigm serves the end user efficiently, asynchronously, and uses low-bandwidth, an asset for personalized multimedia networked applications.

1.2 Objectives

The main objectives of this thesis are to study the possibility for the mobile agents to have multimedia capabilities and to develop a multimedia mobile agent system. Besides the traditional properties of a mobile agent, new multimedia capabilities such as the handling, playback and transport of multimedia files are implemented in the agent system. Multimedia agent-based applications represent a wide door of ideas and evolutions in open and wireless networks. The multimedia mobile agent system is composed of two main parts, namely multimedia mobile platform and multimedia agent-based applications.

The multimedia agent platform provides support for agents to travel and execute on various machines in the network. The main components of the agent platform are the migration facilitator and the Agent Execution Engine (AEE). The migration facilitator implements the agent transport protocol and assumes the transport of the agent and its load which is composed of various medias such as audio and video. The AEE is the process for hosting and running the agent. The AEE activates the agent if the requested resources are available and the security of the machine is not violated.

The agent-based applications are built on top of the agent platform and offer asynchronous services to the user. Mobile agents travel in a network with missions to execute on behalf of the user. Agents can stop executing on one machine, update their internal state and load they might carry and migrate to a new machine. Agents can have different tasks such as searching and retrieving multimedia documents, and communicating with service agents through a set of predefined instructions that constitute the Agent Communication Language (ACL).

1.3 Thesis Outline

Chapter 2 summarizes the agent's concepts, issues and technologies. The evolution of AI agents from objects with some kind of intelligence to complex networked mobile agent systems is also presented. Static and mobile classes of Agents are discussed. A survey of related work and existing agent systems, languages, and tools is made.

Chapter 3 introduces the multimedia mobile agent system. The functional specifications of this system and its architecture are presented. This chapter also presents previous multimedia and agent work to date at the University of Ottawa's MIRLab: the Multimedia News Agent which was created to present personalized multimedia news in a asynchronous mode as opposed to the existing multimedia news application which is based on the client/server paradigm.

Chapter 4 details the design and the implementation of the multimedia mobile agent platform. The various interworking components of the platform are presented: the agent structure, the agent migration protocol, the migration facilitator and the AEE. These elements

provide an environment to mobile agents to travel and execute, and constitute a platform to quickly develop agent-based applications, and explore the convergence of the multimedia and agent technologies.

Chapter 5 exposes three multimedia agent-based applications built on top of the agent platform. The three applications allow to list, search, and send multimedia documents in a network. Chapter 6 concludes this thesis.

1.4 Main Contributions

This work describes the design and development of an architecture and applications that are based around the metaphor of mobile agents. The aim is to propose an infrastructure for a mobile agent system such as a communication protocol, an agent execution environment and a mobile structure and to produce a prototype that allows developers to rapidly build multimedia networked applications.

Furthermore, this research presents a study of mobility within the multimedia context. The use of mobile agents that offer asynchronous operations and low bandwidth utilization for multimedia networked applications is believed to present an attractive technology in the new open network concept, where the wireless networks are integrated, and the information are distributed over wide areas.

Consequently the originality of the work presented in this thesis relies in two main areas. Firstly, the ability of multimedia networked applications to use the mobile agents to send, transport, query and manipulate multimedia documents, while most contemporary solutions prefer static, client-server model of interaction and most agent research focuses on different

issues of mobility such as security, agent communication language, intelligence etc. Secondly, provide an abstraction and transparency for the basic mobile agent actions such as *jump* and *build*. The different levels of the software offer such transparency and allow to developers to rapidly build agent applications.

In the early stage of development, the advantages of dividing the system into two major parts have been recognized. The agent platform is the main part of the system and was built to facilitate the development of agent-based applications. It offers support for agents to migrate over TCP/IP therefore allowing development of Intranet and Internet applications. The agent platform includes several interworking processes: a migration facilitator, an agent constructor, a graphical user interface and an agent execution environment. It also defines an agent structure and an agent communication protocol used by migration facilitators to exchange agents and their loads.

Other contributions of this work are the development of agent-based applications on top of the agent platform. The applications use the simple API provided by the agent platform for basic agent commands such as *build* and *send*. Agents with various missions are sent out in the network, running asynchronously on multiple remote hosts and manipulating files of different medias like audio, video, and text.

The research work described in this thesis has been completed before June 1997.

Chapter 2
Classification of Agents

2.1 Introduction

The term agent was initially used in the field of AI and robotics in the seventies and early eighties to specify computational autonomous objects, robots, or programs which:

- act on behalf of the humans or other computational entities
- are able to analyze a situation through self-contained intelligence and make decisions based on their analysis.

AI research focuses on developing techniques that would allow computers to behave like humans [HEW77]. The research can be divided into three separate subject areas. The first area is

known as natural language processing and seeks to develop computer applications that read, speak and process inter-human languages. In the sixties for instance, the federal government of the United States spent millions of dollars on a computer system that would translate human languages. The project was a failure because computers are not able to understand and process vague meanings the human intelligence are able to. The second area focuses on robots. This research's main concern is to develop visual programs to enable robots to detect their environment and adapt to any changes as they move or perform their associated task. Finally the last area of research is known as expert systems. Those systems make use of artificial knowledge to simulate human experts. The expert systems is the most elaborated area in AI.

The decisions made by earliest agent systems were rule-based, reflecting the scientists earlier beliefs that the human mind works according to specific logical rules and that all statements could be reduced to a combination of the IF-THEN-ELSE conditional rules and the logical operations AND, OR, and NOT. Rule-based agents are driven by a collection of user-programmed conditional rules in solving a task or making a decision. Such agents make use of traditional or conventional programming and could be developed employing common languages like C and BASIC.

Although intuitive, this approach has many drawbacks. Firstly, it requires a complete understanding of the agent role and behavior prior to program the rules. Secondly, knowledge is explicit and does not dynamically change. Rules and codes require to be constantly changed and updated if the habits of the agent are to change. Finally, the scientist's more recent discoveries proved that the human's reasoning recognizes the validity of a sentence or an idea, through more

than just IF-THEN-ELSE schemes and logic operations. In reality, the human mind functions in analyzing an idea, according to the logic operations and more importantly through some insight into its meaning.

Knowledge-based agents were introduced as an enhancment to rule-based agents and to better reflect the human's mind. AI scientists discovered that humans do not always solve problems or make decisions in purely logical ways and that the mind most likely functions by applying approximate solutions to problems. Solving problems in a particular domain requires general knowledge of the objects in the domain and knowledge of how to reason in that domain, embracing therefore the rule-base techniques. Knowledge became the most important characteristic of AI problem solving.

The main drawback of knowledge-based agents is the complexity and difficulty of representing knowledge: solving an apparent simple problem to humans like reading one line requires an extensive amount of knowledge. Consequently, knowledge-based agent systems require a large repository of knowledge that has to be maintained by an engineer with an expertise in the agent domain. Such systems cannot then be customized to every user, and therefore are not suited for personalized applications.

Faced with the problems faced above, AI research on rule and knowledge-based agents failed to deliver the advanced intelligent systems that it promised in the early eighties and the agent technology remained with little consideration for over ten years.

Recently, the agent technology gained new interests and considerable increasing momentum. Academia, software companies and businesses from all over the world show increasing interest in agents in a wide range of applications: user interfaces, network management, information retrieval and filtering, advertising etc. The reasons for such attention are summarized here below:

- the need for new technologies and applications to exploit the gaps left by the tremendous advances in the telecommunications infrastructure: the ongoing convergence of the telecommunications, computing and wireless worlds has not been accompanied by parallel major advances or improvements in applications and telecommunication protocols. Therefore, the vision "information any time, at any place, in any form"[MAG96] will not be affordable without the deployment of new solutions to serve and assist the end user efficiently in roaming and using the electronic services of the humongous public networks.
- the need for intelligent software programs to assist users in everyday's tasks. Today's computers are passive entities that respond to the user's direct manipulations. The trend is to find new applications where computers and people are engaged in a cooperative process to execute complex tasks, like information gathering, that consumes a large proportion of the user's time. This need is becoming apparent for three main reasons. First, more and more everyday tasks are computer-based. Secondly, increasingly more users are untrained and finally, the information infrastructure is dynamically changing, expanding, and evolving towards a gigantic unstructured repository of data.

Among the numerous technologies and solutions considered to fulfill the above needs, software agents are believed to be a remarkable option and may have a great impact on

tomorrow's applications. More than just simple objects with some intelligence, today's agents have been given various properties and functionalities ranging from simple shell programming to self learning personal assistants to sophisticated intelligent mobile programs that navigate a network looking for a specific task to perform. Although the massive deployment of agents in different projects and despite their diverse definitions and approaches, agents are still easily recognized by the way they are exposed. Indeed, despite all the changes and evolutions, agents still inherit most of their attributes and properties from the old agent technology.

2.2 Properties of Agents

The following attributes describe the typical properties of an agent system:

- Autonomy: agents are able to perform their tasks with minimum intervention of humans as opposed to direct manipulation. Agents have also control over their internal state which they constantly update.
- Reactivity: agents perceive the environment and adapt to any change. In case of mobile agents for example, faced with network congestion, the agent should adapt and try to migrate to other destinations.
- Asynchronous operation: one of the most attractive attribute of agents, particularly mobile agents in public and wireless networks. Agents can execute their tasks decoupled from the user and other agents.
- Mobility: agents with the ability to move to a new location for a specific task while preserving their internal state. On each location they visit mobile agents run a script,

communicate with the user, other agents, update their state, and migrate autonomously to another location.

- Intelligence: hard to define, as it has been the subject of many discussions in the field of AI, and no clear answers have yet been found. The following definition of what makes an agent intelligent is taken from [IBM95]: "Intelligence is the degree of reasoning and learned behavior: the agent's ability to accept the user's statement of goals and carry out the task delegated to it. At a minimum, there can be some statement of preferences, perhaps in the form of rules, with an inference engine or some other reasoning mechanism to act on these preferences. Higher levels of intelligence include a user model or some other form of understanding and reasoning about what a user wants done, and planning the means to achieve this goal. Further out on the intelligence scale are systems that learn and adapt to their environment, both in terms of the user's objectives, and in terms of the resources available to the agent. Such a system might, like a human assistant, discover new relationships, connections, or concepts independently from the human user, and exploit these in anticipating and satisfying user needs."
- Communication: agents interact with the user and communicate with other agents. The interaction with the user to seek additional information on a delegated task or to inform the user about useful information such as the end of a task, a threshold attained by a stock on the Internet etc. Interactions with the user can take several forms. It could be simply in the form of statements printed on a command line window, through a user interface or even through voice recognition. The purpose of the communication between agents is to exchange knowledge and coordinate goals and skills to jointly solve a problem. Agent communication is a wide research area and has not reached a mature phase yet. Many of the protocols

proposed for agent negotiation are based on the concepts of contract nets and speech act theory.

2.3 Classification of Agents

This section is a detailed classification of the various agent technologies that exist today. Every class of agents is illustrated by an example of a system in that class. Agents are sorted, in the communications world, into two main classes: static and mobile.

Static Agents			Mobile Agents	
Personal Agents	Service agents	Multi-Agent systems	Remote execution	Fully mobile
• Information filtering and retrieval	• Service to other agents and processes	• Jointly take actions to solve a problem	• Agents finish the execution before the migration	• Agents can stop the execution and migrate while active

Table 2.1 Classification of currently existing agents

A static agent is an intelligent process running on a fixed node without any mobility attributes. Static agents in their turn are sorted into three different types depending on the type of service they offer (Table 2.1). The first type of static agents, known as personal agents or user agents, act as automated secretaries and perform tasks on behalf of the user. The second type of static agents are called service agents. These agents offer services to other processes, particularly

to mobile agents looking for a service on a remote host, and do not collaborate with users. The third type are called collaborative agents. They form a family of agents that communicate and coordinate in solving a task.

In contrast to static agents, mobile agents move between different nodes in solving a task. Such agents enable remote execution and asynchronous operation and are considered as the extension of the client/server communication paradigm. On every node they visit, mobile agents run a program/script and communicate with service agents, other mobile agents or humans. Two sorts of mobile agents are recognized: agents enabling remote execution only such as java applets or operating systems remote batch jobs and agents with full migration such as Telescript agents [WHI96] which may migrate while they are active.

2.4 Personal Agents

Personal agents act as automated secretaries and provide services directly to the user. The first form of services, through advices and advanced help, is done by the personal advisory agents which do not take actions on behalf of the user. On the other hand, personal assistant agents cooperate with the user in daily tasks and perform tasks on his behalf. These agents are more powerful, complex and are equipped with more intelligence.

Personal agents live on the user's workstation and run close to him. They also learn his habits and profiles and, by doing so, provide a big benefit: the agents can run in the background of the user's workstation even if he is logged off, execute some tasks on his behalf and

consequently transform the passive personal workstation which remains idle most of the time into an active processing entity [ORE91].

From a network perspective, two types of personal agents are differentiated: interface agents which access local resources only and networked agents which can access local and remote resources.

Interface agents assist the user in many different ways. They act asynchronously and perform daily tasks on the user's behalf such as mail filtering, local disk management etc. Even though there has been a great amount of research in interface agents for many years now, current available user agents are far from the human-like intelligent assistants. The most common example on interface agents is the information filtering agent. Figure 2.1 illustrates the general architecture of filtering agents.

Filtering agents are very common because they are the simplest class of agents. Their role simply consists of filtering a large amount of information and present what is relevant to the user. Interface agents are personalized to every user: by watching the choices and the habits of the user, they build a customized user profile that is constantly consulted and updated. The work of Jennings and Higushi is an interesting example of user interface agents [Jen93]. Their agent uses an artificial neural network to classify newsgroups into interesting and irrelevant ones and watches the user rejecting articles which are given as feedbacks to the neural network. Another example on user agents is Stanford's SIFT system [Yan96].

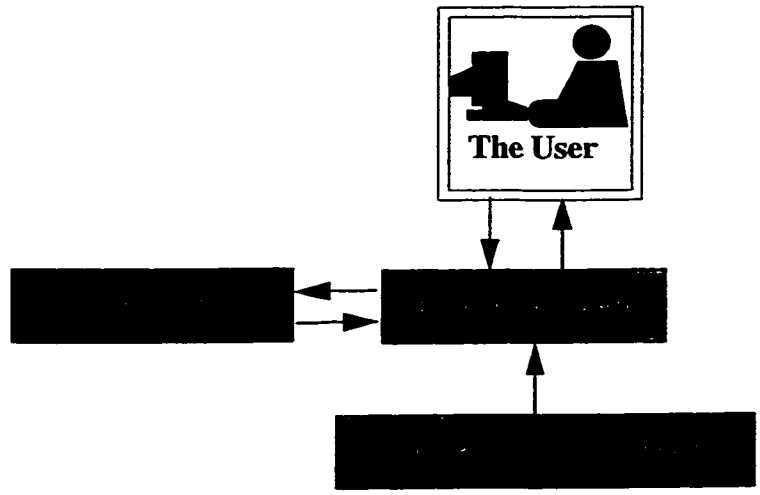

Figure 2.1 General architecture of information filtering agents

Networked agents have the ability to access local and remote resources. Such agents make extensive use of information and services distributed in large networks. The most common application of networked agents are the information retrieval agents which complement the information filtering agents discussed above (Figure 2.2).

The architecture of networked information retrieval agents, may appear similar to the architecture of filtering agents. However the extra arrow indicating a flow of information from the retrieval agents to the information sources reflect a major difference in their concept and practice. Networked agents are proactive in accessing information in the network and do not wait passively for the information to be presented to them. Networked agents build a network profile where the network infrastructure and service sites related to the user's preferences are kept and updated.

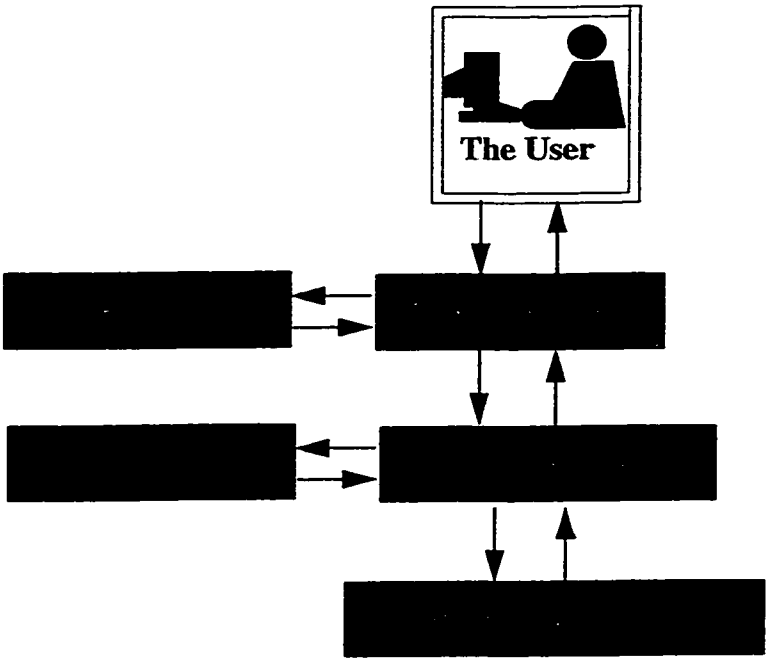

Figure 2.2 General architecture of information retrieving agents

Networked agents are particularly attractive to the Internet. Among many examples, a virtual newspaper agent which, based on the user's interests, puts together a personal newspaper by accessing a number of articles from different sources [URL1]. Another example is an agent that monitors specific variables, like stock prices, and thresholds on specific sites in the network and notifies the user of any changes that might interest him.

In order to assist users in daily tasks and to be able to make decisions on their behalf such as sort incoming mail, schedule a visit, personal agents (user and networked) need a substantial customization to each particular user. We recognize 3 approaches that have been considered in order to customize such agents. The first approach employs user programming engineering and the second approach is based on knowledge engineering. The third and most recent approach

employs machine learning techniques and therefore addresses the problems of the rule-based and knowledge-engineered approaches. Agents could "automatically customize to individual users, by learning through experience" [MIT94]. Agents learn by observing the behavior of the user and detect any regularities in his behavior. There are few learning methods that could be used among them the decision tree induction and artificial neural networks [MIT94].

2.5 Service Agents

In contrast to user agents which live close to the user, learn his habits and interact with him, service agents provide services to other agents and do not communicate with the user. Service agents are used typically for security measures to control the access of mobile agents to restricted resources: in mobile agent systems, a mobile agent is not allowed to access any information on host machines or to write directly to the host hard disk for security reasons. Instead, a local service agent with access privileges communicates with the mobile agent and processes the services requested, assuring the node's security and privacy.

An example of service agents is the VisitorBot [KAU94], a process that assists user agents in scheduling a meeting (Figure 2.3). The following systematic steps are undertaken by the visitorbot in scheduling a meeting. The first step is to constantly listen to requests from user agents to schedule a meeting, on a specific date and time and to record them. The second step of the service agent is to contact all user agents asking them if the user is free on that specific date and time otherwise to provide a suitable time. The last step is to analyze all the answers and to schedule the meeting on a date that is favorable to the majority.

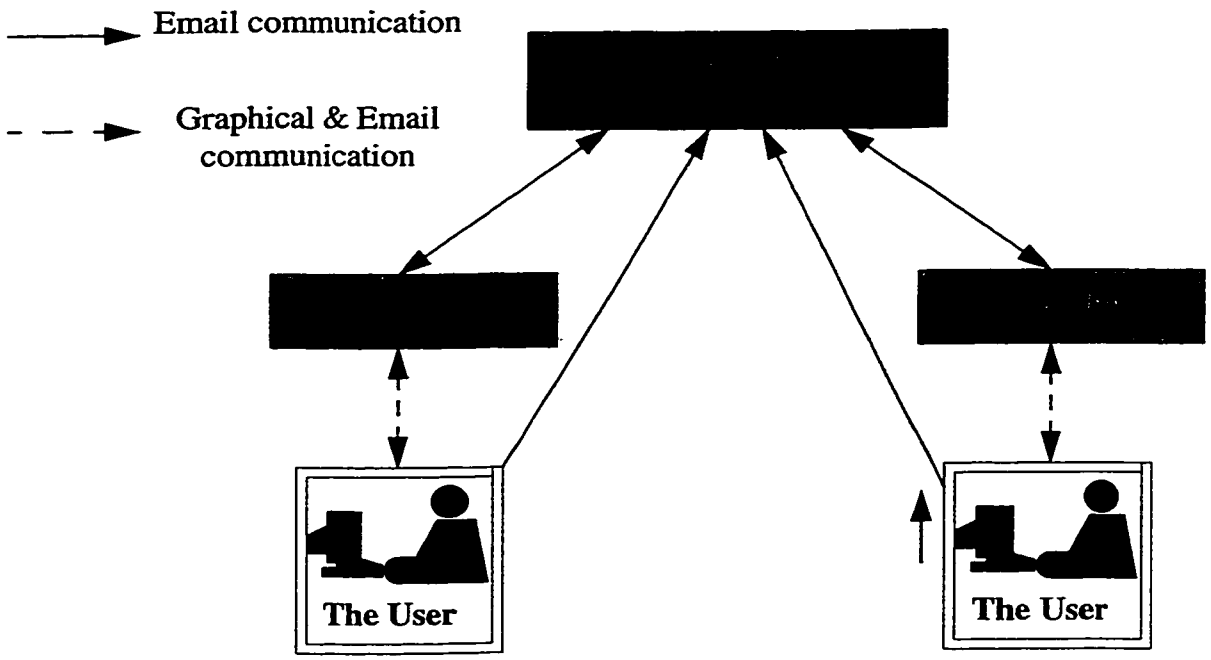

Figure 2.3 Architecture of the Visitor Bot agent system.

2.6 Multi-Agent Systems

Multi-Agent systems or Distributed Artificial Intelligence aim at developing agents that jointly take actions to solve a problem. Such agents communicate with users, local and remote resources and exchange information and services with other agents to solve a problem that cannot be solved alone. The motivations for multi-agent systems are the following:

- Provide solutions for inherently distributed systems
- Solve problems that are too large for a centralized single agent
- Enhance speed, parallel processing and reliability

According to Bond and Gasser [Bon88], and depending on the degree of cooperation

undergone between the individual agents, the following two types of multi-agent systems can be recognized:

- cooperative multi-agent systems, where all interacting agents are designed by one designer with the only goal to enhance the performance of the total system. Such agents are considered cooperative, and their behavior is not concerned with the performance of the individual agents.
- self-interested multi-agent systems: such interacting agents are normally designed by independent designers with the vision of agents interacting for their own benefit, and not for the overall performance of the system. Self-interested agents are competitive and exhibit antagonistic behavior.

The research in multi-agent systems is mainly concerned with the design of agents that manifest cooperation, coordination and communication with other agents. These three ingredients are essential for successful distributed problem solving, and hence are discussed individually in the next three sections.

2.6.1 Coordination in Multi-Agent Systems

Coordination is essential in multi-agent systems and in any other system where multiple elements act simultaneously to prevent interference, chaos, and therefore failure. Similarly to the human brain which is an excellent example of a coordinator, and harmonizes the actions of all the human body parts, a society of agents must have ways to coordinate the behavior of every agent in the system.

There are two main types of agent coordination architectures, centralized and distributed:

- In centralized multi-agent systems as seen in Figure 2.4, a master agent with a wide perspective of the stem goals and architecture, gathers the plans of all the agents in the group, finds conflicting plans, rearranges and creates new plans, and finally assigns the tasks to the agents to ensure global coherence and harmony. This architecture employs the blackboard coordination technique, which is simple but criticized as impractical and contrary to the distributed artificial intelligence paradigm. Werkman's DFI system [Wer90] and the Sharp Multi-Agent Kernel (SMAK) system [KER94] utilize the blackboard architecture to achieve coordination.
- In distributed multi-agent systems, the master agent is non-existent. Agents therefore communicate with their peers and update their own plan until all conflicts are removed. The most eminent coordination technique for task and resource allocation among agents is the Contract Net Protocol [SMI81], [DAV83]. The advantage of direct communication is that agents do not rely on any program to communicate with peers. Disadvantages of the distributed approach is cost and complexity, especially if there is a large number of agents. Every agent will contain the code to support negotiation, update of plan, and a global view of every agent plan in the system. An example on distributed multi-agent systems is Lesser and Corkill's Distributed Vehicle Monitoring Testbed (DVMT) [Les83] [Les81].

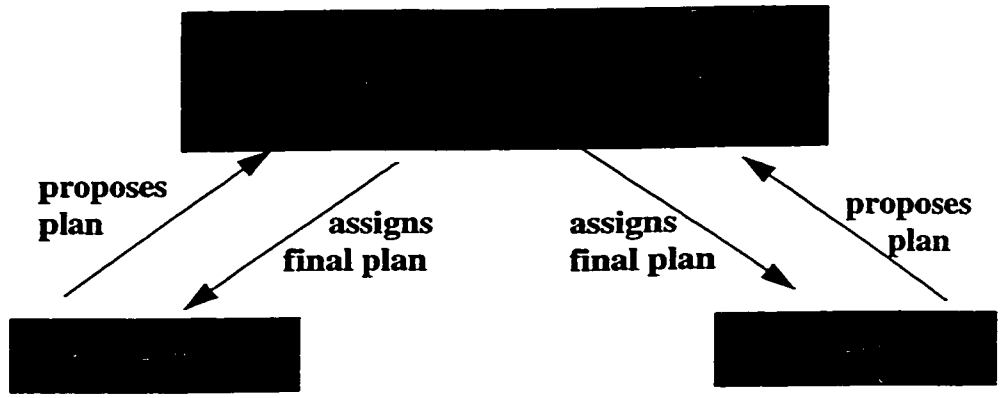

Figure 2.4 Architecture of centralized multi-agent systems.

2.6.2 Negotiation in Multi-Agent Systems

According to Bussman and Muller [BUS1996], "...negotiation is the communication process of a group of agents in order to reach a mutually accepted agreement on some matter." The agreement might be about price, joint action, joint objective, assigned task, etc. Depending on the cooperative behavior of the agents, negotiation can be competitive or cooperative:

- Competitive negotiation occurs between agents that have conflicting plans and not a common goal. Such agents need to find an intermediate plan, otherwise the conflicting plans are not realized. This scenario can occur between selling and buying agents. If the lowest price negotiated is still out of the buyer's budget, the sale cannot resume. KASBAH [Cha96] and MAGMA [Tsv96] are examples of agent systems, where agents representing buyers and sellers perform negotiation in a virtual marketplace.
- Cooperative negotiation is used between agents that have a common goal. Such agents are not competitive and therefore are willing to back down for the greater good of the system.

2.6.3 Agent Communication Language (ACL)

To achieve coordination, agents need to negotiate, interact and exchange information, and, consequently, agents need to communicate. The four models of communication in multi-agent systems range from those involving no communication, to primitive communication, to ad-hoc ACLs, and finally to standard ACLs. These four models are further explained below:

- No communication: in some cases, agents know about their peer's plans without communicating with them. Tubbs S. terms this *tacit bargaining* [TUB94] and points out that it works best in cooperative multi-agent systems.
- Primitive communication restricts the agents to exchange a finite set of signals with fixed interpretations. Consequently, negotiation is practically non existent and coordination between agents limited. Private communication has been applied by Georgeff in his multi-agent approach to coordination [GEO83].
- *Ad Hoc* ACLs: within this category, many agents employ an ad-hoc set of performatives within the *ad hoc* ACL. Many others communicate by depositing information in some shared data structure. The *blackboard* is the model of shared memory that is often used.
- Standard ACLs: we distinguish two popular approaches in the design of such a language. The procedural approach and the declarative approach. The declarative approach is the approach used by scripting languages such as the Tool Command Language (Tcl) [OUS94] and Telescript [WHI94]. They allow programs to transmit not only commands but entire procedures and programs. The advantages of such an approach are simplicity and power. The disadvantages are that the communication is unidirectional and the messages are difficult to merge. It is difficult for an agent to receive multiple scripts that should run and interfere

together. The declarative approach states on the other hand that agents have a better communication by exchanging declarative statements (definitions, assumptions etc.). One of the standards proposed is the knowledge Query Manipulation Language (KQML) [FIN92]. KQML consists of three layers: the content layer specifies the actual content of the message, the message layer is made of a set of performatives provided by the language and the communication layer is the protocol for delivering the message.

2.7 Mobile Agents

2.7.1 Introduction

Mobile or itinerant agents are the most attractive and challenging types of agents. A mobile agent is a task oriented program that can migrate from one machine to another in a network, running if necessary on the nodes visited. A mobile agent is an "intelligent" process that runs asynchronously from the user: the agent chooses when and where to migrate, suspends the execution at an arbitrary point, transports itself to another machine and resumes execution. Designed to fit in large distributed networks, mobile agents introduce a new communication model with many advantages, new applications, and more services.

Mobile agents may be viewed as an alternative to the Remote Procedure Call (RPC) model introduced in the seventies in which clients communicate with a non-local server (Figure 2.5).

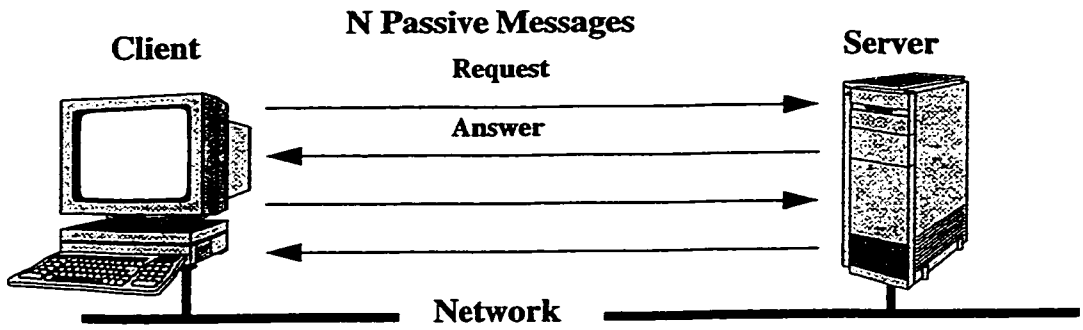

Figure 2.5 The Remote Procedure Call model

In the RPC model, clients send requests to the server through the network. A typical request consists of a procedure call and the procedure arguments. Upon receival of the message, the server performs the request internally by running the appropriate function and sends back the results to the client.

Although simple and popular, this communication model has a few limitations: firstly, the client/server interface is static and constraints the client to a limited set of queries provided by the server. This limitation leads in some cases to inefficient interaction and bandwidth utilization. If a client, for instance, needs to sort a large repository of remote data, and the server does not provide such capability, the client has two choices: make multiple intermediate requests or transfer all the data and sort it locally. The second limitation of the RPC is that interaction between clients and servers requires permanent network connections and could not support systems that do not have permanent network connections such as mobile computers and PDAs.

The limitations of the RPC have driven the research to find a new communication model which serves the needs of end users more efficiently and uses the network resources more intelligently.

2.7.2 The Mobile Agent Paradigm

The communication paradigm based on the mobile agent paradigm enables computers not only to exchange data and call for procedures but also complete programs. Chunks of program code can travel around a network and execute at different nodes (Figure 2.6). The traveling programs are called mobile agents.

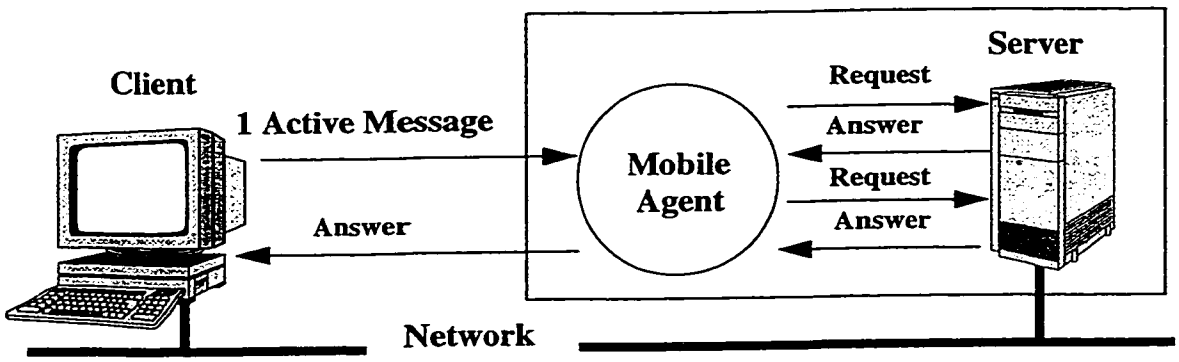

Figure 2.6 The mobile agent communication model

The idea of sending programs for remote execution is rather old. It has been called "remote (batch) job processing" in the seventies and "function shipping" or "remote evaluation" in the eighties. The main motivations of the remote evaluation were the lack of capacity to execute large programs locally and for load balancing. An enormous contrast with the new mobility concepts which aim for open environments, the integration of the wireline/wireless

networks, and a more effective communication model. Another major difference subsists in the mobile agent structure: apart from its code and data, an agent has an internal execution state which allows to the agent to suspend the program, transport itself to another node and resume execution from where it left off. Finally, agents have an autonomy and an intelligence which make their behavior and characteristics boundless. Some mobile agents for instance are able to create and fork new agents which enables task distribution and parallel execution.

2.7.3 Advantages of Mobile Agents

While many researchers defend strongly the mobile agent communication model and fight to see it more deployed in the networks, others believe that the traditional RPC has better preformance, and is more safe and stable. In front of the ongoing debate on whether mobile agents have better performace than the RPC, It is probably more appropriate to state that the comparison between these two technologies cannot be radical: the performance of each model depends strongly on the application, the network size, the size of the agent etc.

In reality, the RPC and the mobile agent model should coexist in such to offer the most effective solution to the applications which are making more and more use of the public networks. If the RPC has many limitations, it is suited for small applications that run only on the wireline infrastructure, make little use of the networks, and need a small bandwidth. On the other hand, the agents migration is complex and causes communication overhead, therefore, is desirable where interactive applications make regular use of the network, connect to many nodes, send large packets, etc. In other words, mobile agents are desirable is some applications where

the RPC model results in poor performance and high bandwidth cost. A such example is an application which involves downloading large files from different locations to a mobile computer. In such case, It is more efficient to have a mobile agent asynchronously travelling to the remote locations, grouping the files and downloading them all together.

What follows is a summary of some of the prominent advantages of deploying the agent technology and a variety of purposes that mobile agents could serve in distributed systems. A more detailed discussion of pros and cons of mobile agents can be found in [HAR95]:

- Unlike the client/server model where ongoing interaction requires ongoing communication, mobile agents offer asynchronous interaction by sending one active message to the service site. This characteristic of reducing network communication is very attractive in a wireless network, where communication is unfortunately very costly. Consider for example a service consisting of filling out several forms. In the traditional client/server model, a client sends normally a request to fill out the forms. The forms are sent one at a time. Once one form is completed, the client sends it back to the server and asks for the next form to be sent. The more effective alternative would be to send one agent with all the information necessary to fill out locally all the forms, reducing therefore the communication to only the transmission of the agent to the server.
- The ability to assign different tasks to different mobile agents and send them simultaneously to one or multiple nodes allows highly dynamic and parallel computation.
- Mobile agents provide a potent means for dynamically watching a regularly changing conditions in a network site, such as watching the fluctuation of stock market prices and then

notifying the user when certain thresholds are reached.

- In centralized network management systems, mobile agents could be delegated specific management tasks and therefore could decrease the pressure on the central operations system.
- Mobile agents can be an excellent tool for advertising and service representatives. Remote service providers could delegate a large number of agents with a marketing mission. Once arrived at the client machine, the agent starts a dialog with the user and provides interactive information about the services being promoted.

2.7.4 Issues Raised by the Mobility of Agents

Mobile agents are a new technology and face many issues and challenges. The following is a description of two prominent issues the agent technology is facing.

- Safety is issued to protect receivers from agents, agents from receivers and agents from agents. Some steps have been undertaken towards the protection of machines from incoming hostile agents: Firstly, agents are written in interpreted scripting languages which makes easier to scan the code and to detect malicious instructions and programs. Secondly, receivers can authenticate incoming agents, and refuse to run any code originated from untrusted sources. Thirdly, receivers can limit the resource consumption of agents such as CPU time, memory and disk space.
- Secrecy arises since the user who receives an agent may violate the privacy of the sender by examining its hidden agenda and modify the code. Therefore, it should be impossible for the

clients to examine the agent code or to modify it. It is difficult to achieve such secrecy in software, and one solution proposed is to implement the interpreter in hardware and to encrypt the agent code.

2.8 Existing Mobile Agent Systems and Tools

This section reviews some of the existing agent systems and tools that help building mobile agents: Java and Tcl/Tk in open languages, Telescript as a propriety language and finally three existing agent systems: Ara, Tacoma and Agent-Tcl.

2.8.1 Open Languages

- **Java**: open language created by Sun Microsystems as a software development tool for distributed computing [SUN95]. Java is a simple object-oriented language based loosely on C++. It provides multi-threaded features and synchronization between threads. Java is interpreted and designed to be compiled to architecture-independent byte codes. These byte coded instructions then execute on a "virtual machine" which produces machine code for execution on the underlying architecture.

Taken individually, the characteristics discussed above can be found in a variety of software development environments. The advantage of Java is the innovative manner in which it combines these characteristics into a powerful programming system which supports distributed computing

- **Tcl/Tk:** The Tool Command language [OUS94] is a high level scripting language accompanied by the X windows toolkit for building window widgets. Tcl is a string-based command language with little syntax which makes it easy to learn and use. Tcl is interpreted and provides a list of primitive commands for generic programming such as variables and loops and procedures. As a scripting language Tcl is similar to other UNIX shell languages such as the Bourne Shell (sh) and the C Shell (csh). The main difference is the possibility to add a Tcl interpreter to any application and to call the Tcl commands from the application. Tcl serves in such as an extension language. One more characteristic of Tcl/Tk is the possibility to extend the Tcl interpreter by writing new commands in C and call them as if they were Tcl built-in commands. Another advantage of Tcl is the contribution of the Tcl community which made public new versions of Tcl and new commands that can be accessed downloaded and used.

2.8.2 Propriety Languages

- **Telescript:** The Telescript technology [WHI96] introduced by General Magic Inc. in 1994, was the first commercial implementation of transportable agents and was designed with the vision of the electronic marketplace, where consumers and providers transact business electronically through public networks. Telescript introduced the concepts of agents, places, travel, meetings, connections, authorities and permits. An agent is a mobile entity that can travel from place to place, a place being a computer that offers a service to the mobile agent that enters it. A mobile agent can meet another stationary agent. A meeting lets agents call

one another' s procedures. Telescript allows also two agents located in different computers to connect in order to communicate and exchange information.

The Telescript technology has three main components: The language, the engine or interpreter and the communication protocol. Telescript is a pure object-oriented language for writing mobile agents. Like any other object-oriented language, a Telescript program is a collection of hierarchically organized classes, with sub-classing and multiple inheritance. General classes such as agents are predefined by the language, while application specific classes have to be defined by the programmer. The language is complete, applications could be written entirely in Telescript, although they could be written partly in C. The engine deals with the low-level interpreted language, and a compiler translates between the two. Telescript's engine executes places and agents and communicates with the computer resources through three APIs: a storage API, a transport API and an external API which allows the interaction with C.

Telescript engines communicate in order to transport agents between them. The migration protocol can operate over a wide variety of transport networks including those based on the TCP/IP protocols, the X.25 interface of the telephone companies, or even electronic mail. Telescript also features numerous safety precautions which include interpretation, credentials and permits. Interpretation means that agents are processed by an interpreter that allows run-time checking of agent actions. Credentials allow agents and their owners to be identified, reducing the likelihood of anonymous virus-like agents. Permits define restrictions on agents capabilities such as lifetime, maximum size, and incurred cost. Telescript's weaknesses

remain in that it requires high-end workstations or special-purpose hardware (UNIX or PDA version respectively) and does not provide explicit support for reasoning and sophisticated agent cooperation mechanisms. Furthermore, Telescript remains a proprietary commercial product and is restricted to large, enterprise networks.

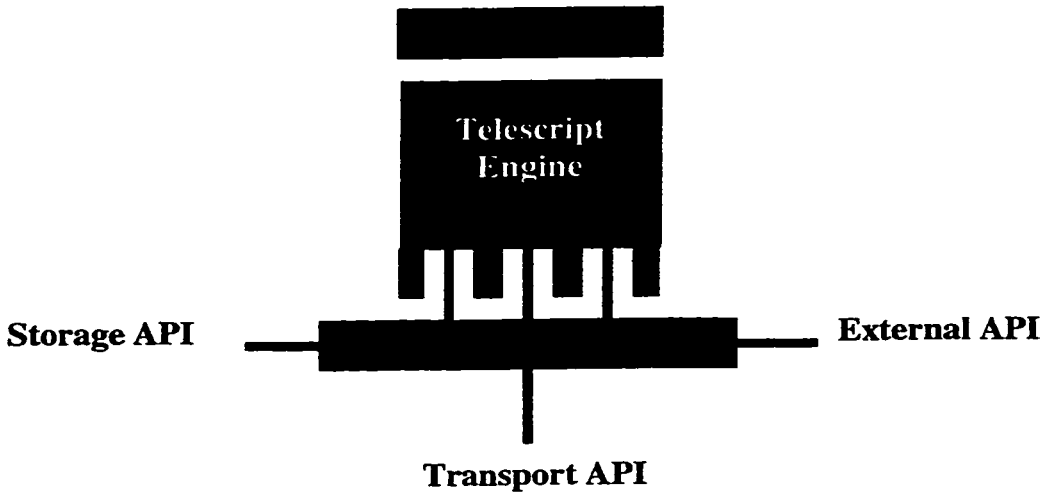

Figure 2.7 The Telescript architecture model

2.8.3 Mobile Agent systems

- **Agents for Remote Actions (Ara)** project [ROT97] was funded in 1994 by the Distributed Systems group, Dept. of Computer Science, University of Kaiserlautem, Germany. The design rationale of the Ara platform is to add mobility to the well-developed world of programming, and to find a new communication model as a substitute to the client/server model which is unsuitable for mobile computers. The agent model was chosen for the benefits of asynchronous interaction, local access to services, on-site processing of data and

dynamic remote configuration. Ara defines an agent as a program written in an interpreted language with the ability to migrate to new places and to communicate with other agents. A place is a software object that can host an agent after its migration and may impose specific security restrictions on the agent staying at that place. The agent in a place could call specific services or communicate with other agents. The Ara's architecture, as can be seen in figure 2.8, provides security through interpretation and portability through a generic core. The functionality of the system core allows to employ several interpreters for different programming languages at the same time, unlike the restrictions of systems built with Java or Tcl.

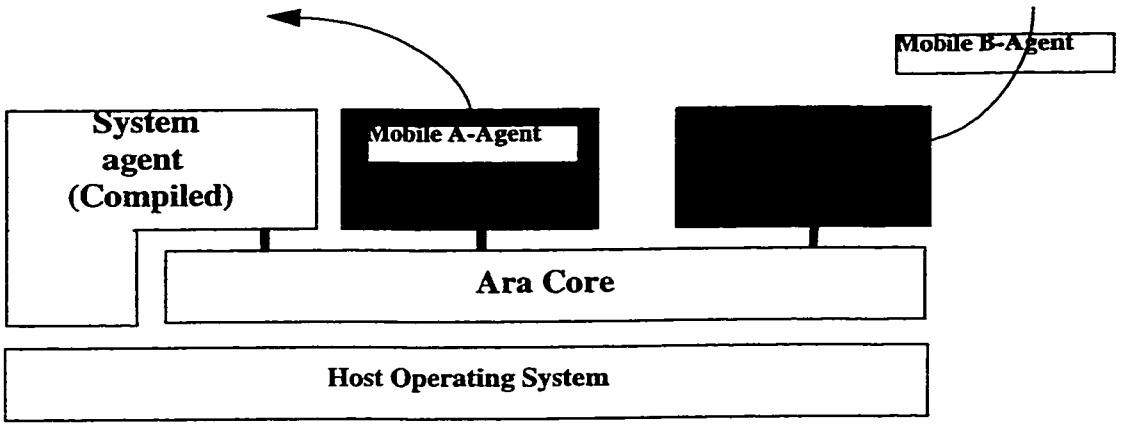

Figure 2.8 The architecture of the ARA system

- **Tacoma** [JOH95] is a joint project between university of TromsØ and Cornell university. Tacoma focuses on operating system support for mobile agents in distributed systems. Some of the issues examined are:

  1. Identifying and implementing the abstractions that operating systems should support for agent based applications. This is partly done by identifying the structure of agent-based applications and their needs.

2. Providing increased fault-tolerance of agents

3. Accounting: including applicability of electronic cash

4. Security: protect the environment from agents and agents from the environment

5. Management: monitoring and control of agents

The Tacoma architecture introduces the notion of folders, briefcases and file cabinets. A folder is a unit of data accessible by agents. For instance, a CODE folder contains the source code of an agent and a DATA folder contains data that can be associated with the agent in the CODE folder. A briefcase is a collection of folders carried by the agent such as CODE, DATA. A file cabinet is a collection of stationary folders for permanent data repository purposes.

Agents can leave data in a file cabinet, read or remove data from it. Agents can move in a network by invoking the *move* operation, meet by using the *meet* operation, and exchange information. In order to move, agents are wrapped in a briefcase. At the host site, the *tac_firewall* agent receives the briefcase and hands it to *tac_exec* agent which will extract the agent from the briefcase and sets up the execution environment for the guest agent. As can be seen in figure 2.9, Agent A represented by the briefcase is received by *tac_firewall* at site 2 and executed by tac_exec. Agent A can then meet with Agent B, or with Ag_local and exchange a briefcase or simply read and write to the local file cabinet.

Tacoma is built using Tcl-TCP a Tcl extension supporting TCP communication, and runs under HP-UX, Solaris, BSD Unix and Linux. Agents may be written in C, Tcl/TK, Scheme,

Perl or Python.

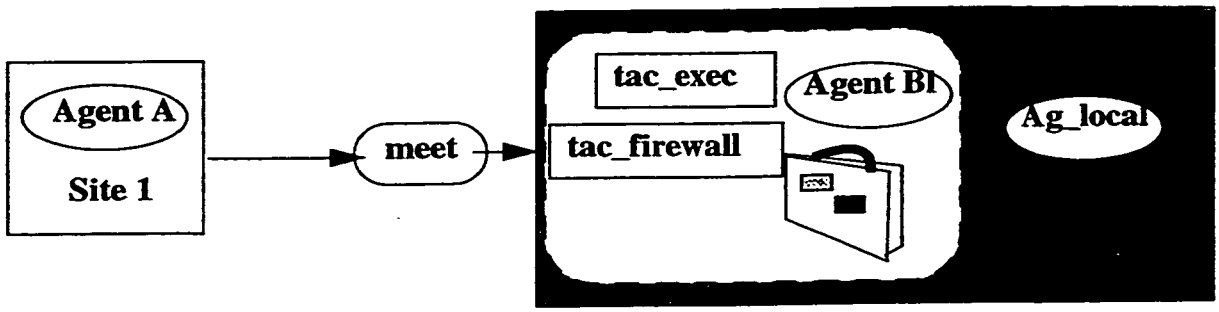

Figure 2.9 The Agent migration in Tacoma

Agent-Tcl is a mobile agent system that is under development at Darmouth College. The architecture of the system consists of four layers (Figure 2.10). The lowest level is an API for the available transport mechanism. The second level is a server that keeps track of agents running on its machine, authenticates incoming agents and passes every agent to the right interpreter, and allows agents on local and remote hosts to communicate by sending and receiving messages. The third level consists of one interpreter for each supported language. Each interpreter consists of the interpreter itself, a state capture that allows agents to stop executing at any arbitrary point and migrate to a new site, a security module that protects the environment from malicious agents and finally an API that interacts with the server for migration and communication purposes. The architecture of Agent-Tcl has not been completely implemented, the current version supports a single language Tcl and a single communication mechanism which is TCP/IP.

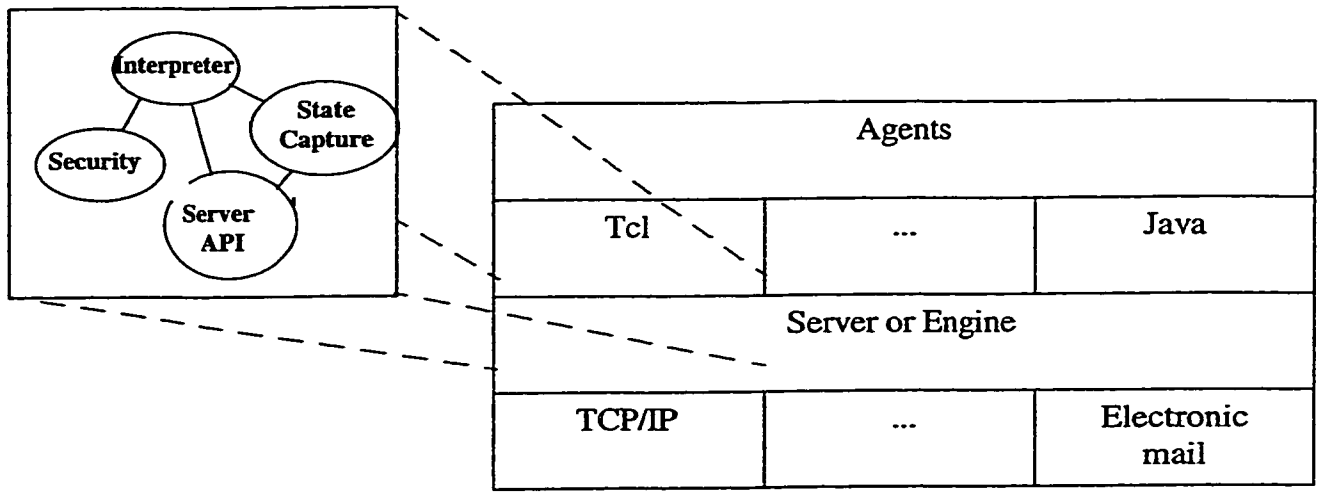

Figure 2.10 The layered architecture of Agent-tcl

Chapter 3
Multimedia Information Systems and Mobile Agents

3.1 Introduction

Multimedia information systems emerged in the eighties with the progress of broadband networks, storage and computer processors. Multimedia services such as video-on-demand and video conferencing are widely available and have been received with enthusiasm by computer users around the world. Multimedia improves the communication between people by integrating different types of media. The media could be of discrete type such as text, images and graphics or of continuous type such as audio and video..

The media types supported by multimedia systems have various storage and playback requirements. Storage of text consumes approximately 2kB per page while CD quality audio requires 176 KB per second of data. Also, in a similar comparison, visual animation playback rate is 2.5 MB/s while voice playback rate varies between 6 and 44 MB/s. As a consequence of such diversity, multimedia information systems have to support the various storage and playback conditions of each media for a successful integration. With the advances in compression techniques, storage, real-time processing and networking, multimedia systems have reached some level of maturity in the nineties: video-conferencing and networked real-time high graphical resolution applications such as Doom and Quake are possible over low-bandwidth networks such as the Internet.

The application of mobile agents in the multimedia systems remains inexploited. Multimedia agents however can provide asynchronous access to multimedia data which can be of discrete and/or continuous type and can be used to build sophisticated interactive remote user interfaces. Multimedia agents are also a strong candidate for mass advertising and marketing technologies that providers of electronic services are looking for in public networks. We believe that the integration of those two leading edge technologies will change the face of computing: multimedia mobile agents will enable very powerful mobile service clients, and will also open new doors for new applications.

This chapter briefly reviews MEDIABASE [KAR93], a project created and developed at the MIRLab to study different aspects of distributed multimedia information systems like synchronization, document architecture, storage and retrieval. Within the MEDIABASE project

many applications were developed. The Multimedia News Application [FAL95] in particular will be described. The News Application is based on the client/server model. Users distributed geographically can access a main database where documents are stored and archived. This chapter also presents the Mobile News Agent [KWA97], an agent based system developed to agentize the multimedia news application. The Mobile News Agent provides personalized news and asynchronous services. Finally, this chapter introduces the Multimedia Transportable Agent Sytem a prototype inspired by the Mobile News Agent and built to study more in depth the issues of agent mobility. The system addresses the roles, issues and advantages of mobile agents in the world of computing and combines the multimedia and the agent technologies.

3.2 The MEDIABASE Project

The following various elements were developed within the MEDIABASE platform: a database model and architecture, a database query language, a multimedia file system, information storage and retrieval models, a document model, a data model, communication protocols, and a media player. The platform has been developed on Unix workstations connected through the MIRLab LAN by both Ethernet and Asynchronous Transfer Mode (ATM) technologies. Three main applications were built on top of the MEDIABASE platform: the Multimedia Newspaper, Multimedia Groupware and Telelearning Coursware. The next section describes the Multimedia Newspaper.

3.3 The News on Demand Application

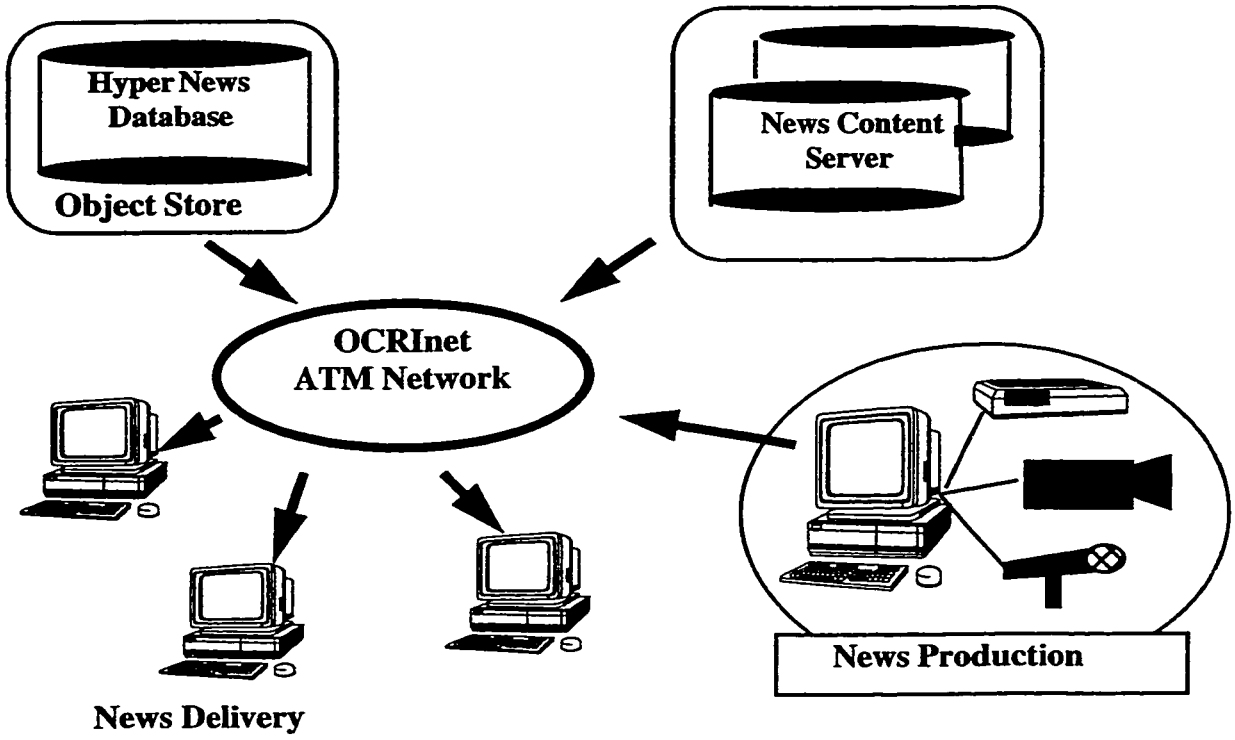

Figure 3.1 The architecture of the News Application

The generic architecture of the News Application is shown in figure 3.1. The application includes four main distinct components: a production center for document authoring, a hypernews database for hypermedia news documents, a content database for authors, and news clients. All the components of the system communicate through OCRInet, an ATM network in the Ottawa region. The hypernews database contains completed documents. It is updated on a daily basis, and supports document aging and archive management. The production server supports collaborative document creation. Editors from geographically distributed locations can cooperate in creating documents whose contents are drawn from the news content server. The news documents are written in the HTML format, and contain multiple media type like text, graphs, audio and video.

The main goal of the News Application is to provide the user with a powerful and friendly tool to browse the news documents in the hypernews database. Consequently, the main interface of the system, shown in figure 3.2, offers to the user the ability to search the database, initiate a video browsing session through the video-browser, create and maitain a personal profile, see the keyword tree of his database, follow-up on the current viewed document and therefore be presented with all documents related to the current article. Additionally, the HTML documents offer the flexibility to view the news articles with any type of the Internet tools such as Mosaic and Netscape and therefore can contain all the common media formats: JPEG, MPEG, GIF.

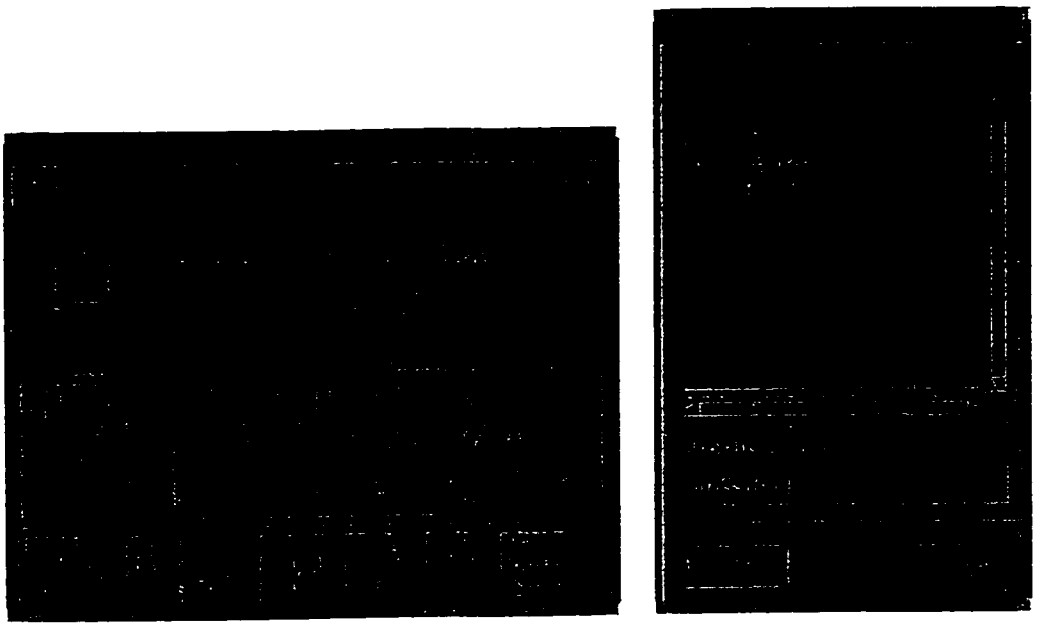

Figure 3.2 The main interface of the News Application

The multimedia news application represents an attractive application for the deployment of agents for several reasons. Firstly, the news applications is a client/server application and

therefore bounded by the RPC limitations. Deploying mobile agents in retrieving and transporting the news documents offer all the advantages of mobiles agents in networked applications: asynchronous operations, more flexibility, bandwidth efficiency etc. Secondly, the multimedia news is a highly dynamic environment where documents keep being updated daily therefore customized user agents can have a role in creating and updating the user's profiles. Thirdly, the news application is a multimedia application and is optimal in complexity, hence, it minimizes the efforts to build an agent prototype and allows to study quickly the convergence of the multimedia and agent technolgies and all the advantages and issues to resolve.

Driven by the three advantages described above, The Multimedia News Agent System was developed to agentize the Multimedia News Application. The new agentized application provides personalized news to the user and combines the use of static and mobile agents. The following section describes the architecture and implementation of the news agent system.

3.4 The Multimedia News Agent System

3.4.1 Architecture

The agentized news system design offers two possibilities when bringing in news articles. The first possibility is attained through two static agents: one for filtering and retrieving, the second for the presentation to the user. The second possibility involves the use of a mobile agent. In this scenario, the mobile agent moves itself to the news database and takes care of the filtering, retrieving and then moves back to the user machine for the presentation. The general

layered architecture of the sytem is shown in figure3.3

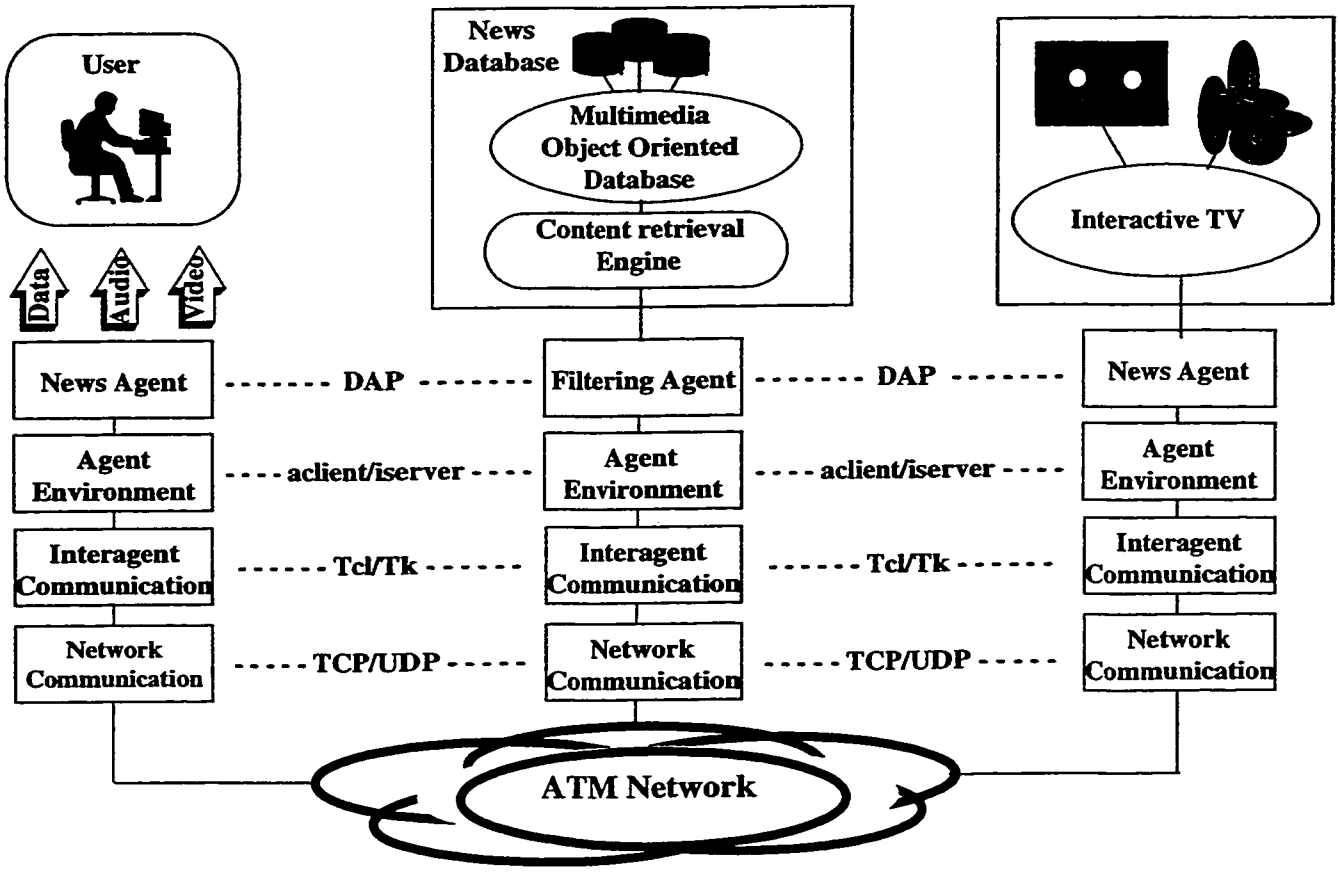

Figure 3.3 The layered architecture of the agentized news system

3.4.2 The Static News Agent System

The top layer is the agent layer which has the total control of the system. On the user site, the agent layer is also known as personal/user agent layer. This is where the habits and profiles of the user are observed, stored and updated. Also optimizations to every user are performed at that level. The agent layer communicates with the lower layers the preferences of the user. This layer also plays the role of presentation layer when the news articles are retrieved from the

database. The format of the articles are represented according to the user profile.

On the service site, the top layer is known as filtering agent. This agent filters the database for relevant articles and monitors constantly the content updates of the news. The filtering agent communicates with the database server for searching and retrieving of the news files through a database communication protocol.

The news and filtering agents communicate through the interagent communication layer, which defines the inter-agent communication language, based on Tcl/Tk scripts.

3.4.3 The Mobile News Agent System

The role of the layers in the mobile agent system differ enormously from their respective role in the static agent system. Once responsible for the presentation of articles in the static agent system, the top layer is now responsible for the negotiation of the resources required to present the articles that are brought and displayed by the mobile agents. The second layer from the top, known as the agent environment layer, had a minor role in the static agent system but an extensive role in the mobile news agent system. This layer allows hosting and running of incoming mobile agents and similarly dispatches migrating agents. The communication protocol between two agent environment layers is based on the aclient-iserver paradigm. This paradigm is an evolution of the client-server paradigm, the client being now asynchronous and the server interactive. The aclient-iserver model facilitates the migration of agents. The iserver has the ability to become a client to the service provider upon reception of an agent. The service

provider in the news application is the news database.

3.4.4 The Mobile News Agent Implementation

The mobile news agent implementation includes the agent environment, the mobile agent and the news application. The news database and the iservers were implemented on SPARC 10 while the GUIs and aclients were implemented on SPARC IPCs and IPX workstations. All workstations are connected via 10 Mbps Ethernet and some workstations connected by OC3 155 Mbps ATM network.

The mobile agent is made of Tcl scripts. The communication and the environment for agents to respectively travel and run are based on the aclient/iserver model. The aclient is a process written with the C language for handling the migration of the agent to the iserver. The aclient also listens for messages reporting the mission status of agents. Those messages might be the report of a failure or a success. Upon reception of the agent, the iserver creates a Tcl interpreter to interpret the script in the agent. The script is interpreted line by line. During the execution of the script, the iserver becomes a client to the news database: the script being interpreted is actually an interaction with the news database server. The interaction involves searching and retrieving documents from the database. When the agent completes the interaction with the database, it is sent back to the user. The results of the database queries are then presented to the user in a pop-up window. The results could also at the user's request be emailed back at any email address.

The Mobile News Agent System opened new research directions. It also represented a good base and motivations for the research work described in this thesis for the following reasons:

- Firstly, the Mobile News Agent System has a running prototype in the lab and therefore demonstrates the difficulties to face in building agent systems, improvements to develop, existing features to enhance and new features to consider.
- Secondly, the news agent system did not look in depth into the agent mobility and the role of agents in multimedia applications and communications. The system's main focus was to study the possibility of "agentizing" an existing client/server application and quickly build a prototype to overview the advantages of agent based systems such as personalized services, asynchronous operations etc. Issues such a multiple migrations within one mission, transport of multimedia files remain inexploited.

3.5 The Multimedia Transportable Agent System

The Multimedia Transportable Agent System is developed to study in depth the mobility of agents and the role of agents in the multimedia communications and applications. Currently, most of the research on mobile agents focus on the mechanism to allow the migration of agents, and existing projects allow for mobile agents to capture their internal state, migrate to a new machine and run from where they left off. So far, none of the projects really studied mobile agents and multimedia systems. What characterizes the research described in this thesis is the support that the agent system provides for agents to manipulate multimedia data. The agents can travel in a network, query, retrieve, and transport documents of various media as part of their

load.

The integration of mobile agents and multimedia systems opens a new page in the telecommunications research. Agents can be applied in new applications that take full advantage of their characteristics. Multimedia agent-based applications that offer asynchronous services and efficient network utilization by decreasing network data traffic can be of enormous use.

3.6 Global Architecture

Figure 3.4 shows the generic architecture of the agent system. The system includes users and service sites distributed in an Intranet. The system also connects the Intranet to the Internet and therefore to other Intranets.

Mobile agents are launched in the network from the users' stations or from the service sites. The sites visited by agents can be sorted into four main categories or classes:

- service sites: such services are the equivalent of the remote servers in the RPC model. These sites offer access to resources such as the news database or to other networks such as the Internet. In the RPC model, multiple requests are made to the servers through messages containing calls for procedures and the corresponding arguments. The procedures are processed by the server and the results returned to the client.

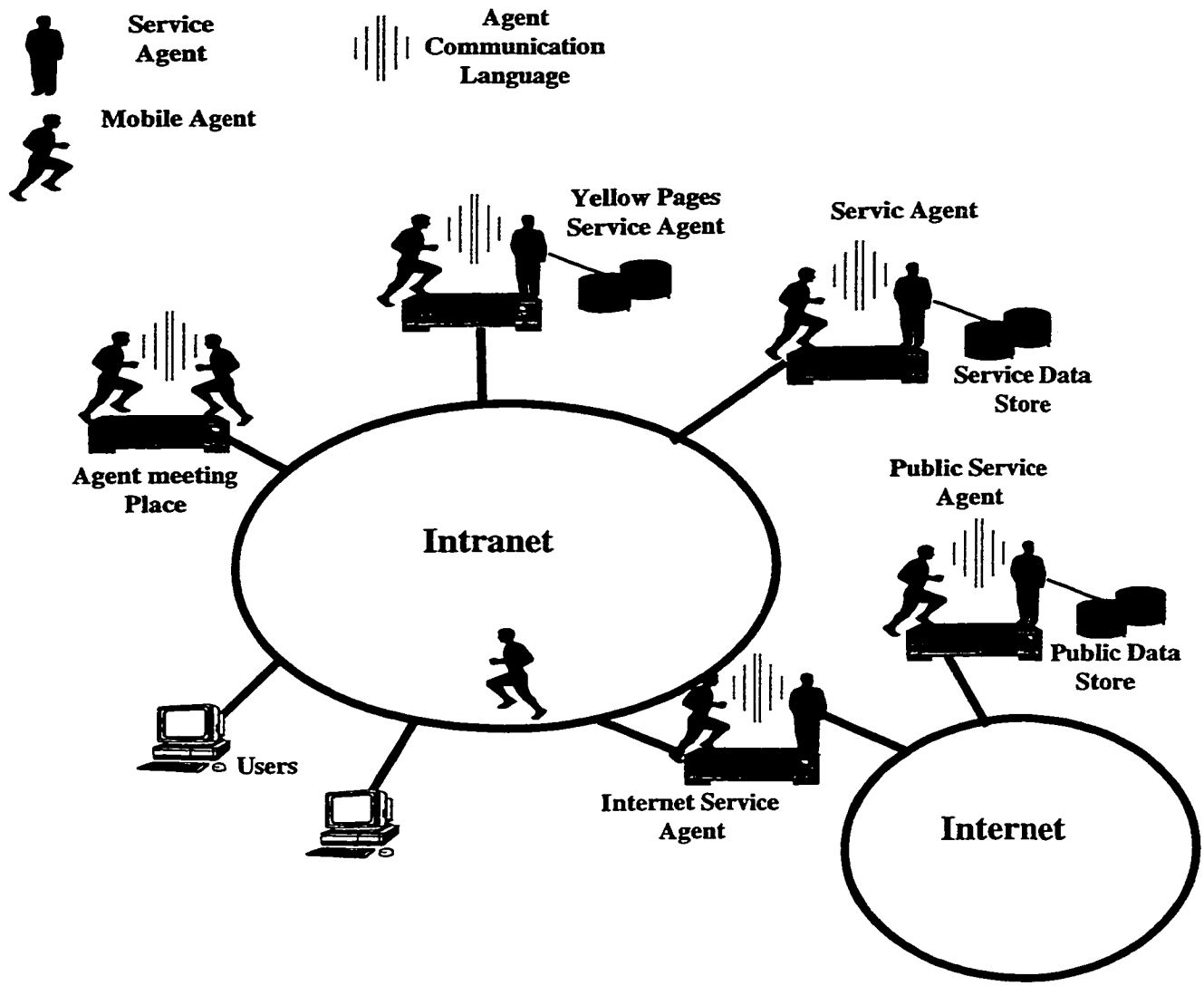

Figure 3.4 The general architecture of the Multimedia Transportable Agent System

In the agent model, the client makes only one request to the service sites by sending all the requests in a complete program. Unlike the traditional clients, The agent runs on the same machine where the service is provided, interacts asynchronously with the resources and migrates independently to other service sites. The agent returns to the initial machine with the results of its interactions.

- yellow pages: a large repository of information available to the agents as a guide to start off their mission. When the agent is launched in the network, It normally has little or no information on where to start in the network and the list of service sites relevant to its mission. The yellow pages are typically the first destination in every agent's itinerary. The yellow pages server inspects the agent's mission and returns a list of sites that will help the agent to start the migration.
- User's sites: Being an excellent tool for remote multimedia playback, a mobile agent is therefore a powerful messenger between users and an excellent tool for marketing in public networks. Also, the ability of a mobile agent to transport several files of various media and to display them according to the agent script without the human intervention, brings a lot of dynamism to the communication applications. For instance, most messaging and file transfer today are done using the electronic email system which services are considered primitives when compared to the advances in networking and processing. The electronic mail simply stores the messages received and the attached files in the mail repository and awaits the user to look for them. Also, the attached files have no intelligence for playing back, and the user might have to launch several applications to view the attached files that can be of different formats. The mobile agent, in contrast, can notify the user of its arrival to his machine and asks for permission to show the attachments. This is a perfect example on the new type of applications needed to fill the gap left by the computing and the telecommunication technologies.
- Agent meeting points: public servers for agents to meet and interact. The agent meeting points are the equivalent to the newsgroups in the internet. The difference in agentized networks is that the agent does all the interactions on behalf of the user. Normally, the user

has to post an ad, and regularly check the newspages for possible replies. Agents interact with another agent by calling each other's public procedures. Besides its internal state, script and load, every agent contains a list of public procedures that can be of use for other mobile or static agents. These procedures are the only public interface of the agent, which does not exhibit its internal structure and private agenda.

3.7 Layered Architecture of the Agent System

The layered architecture of the agent system is shown in figure 3.5. It consists of three symmetrical layers on top of TCP/IP. This architecture is applied to all the places visited by the mobile agent in the network.

The agent layer is the top layer and the central control of all the system. This layer includes the following elements on the user's side:

- the user agent where the user's preferences and habits are constantly monitored, evaluated and updated in the user's profile. The user agent provides also the interface to the user to make service requests which are eventually delegated to a mobile agent. The user interface can vary from simple command line interface to advanced voice recognition mechanism.
- The agent constructor builds the agent file according to the mission and the user profile. The agent file consists of the agent header and the agent script. The agent header contains the mission description, the user's identification, the agent properties, the load carried, etc. The agent code is a list of commands from the ACL.

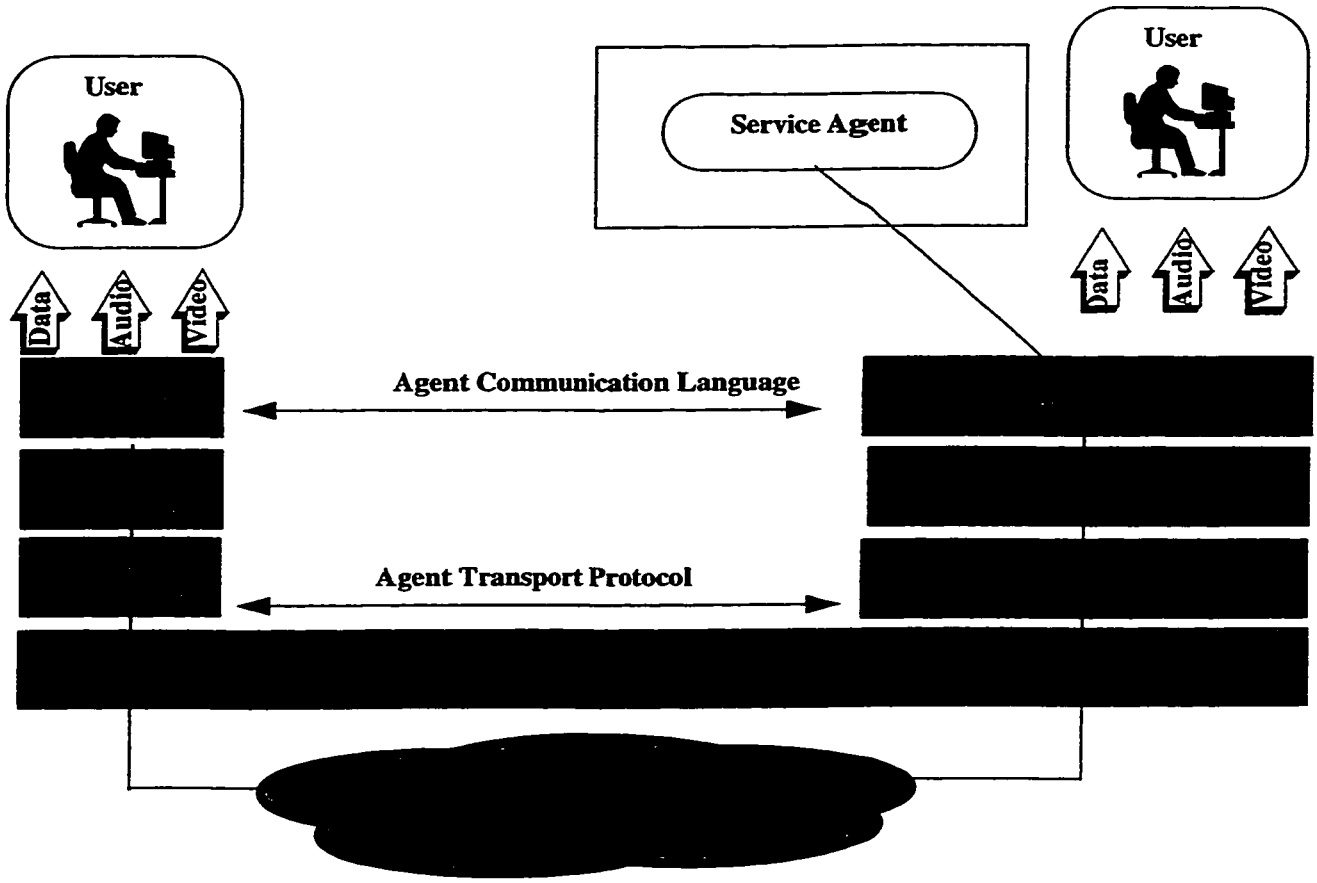

Figure 3.5 The layered architecture of the agent system

On the service site, the agent layer corresponds to the layer of the service agent. When the mobile agent runs at the service site, the service agent communicates with the mobile agent and executes internally the agent code. In fact, the service agent transparently translates the high level commands in the agent code to a lower level service specific commands and returns the results of the commands executed to the mobile agent.

The Agent Environment layer is the second layer in the agent layered architecture. This layer consists of the following two components:

- The agent interpreter which executes the high level agent commands. The agent interpreter is in its simpler form an ordinary script interpreter like Tcl and Perl. These interpreters however have a limited set of commands and are not suited for agents with missions that require many interactions like database queries or communication with an another service or mobile agent. In such cases, the interpreter needs to be enhanced to support high level specialized set of commands . The enhancement in the interpreter reduces the size and the complexity of the mobile agent, the computation is therefore done at the interpreter level.
- The AEE is responsible for running the mobile agents. This process is equivalent to the *tac_exec* in the TACOMA distributed system. For every agent, the AEE retrieves the agent header, and then grants the permission or the denial to the agent to run. It also sets the maximum allocation of computing resources the agent can consume. In order to allow to an agent to run, the AEE reads first the agent info, particularly info on the sender of the agent and ensures the sender is not banned the access to the local services. Secondly the AEE examines the mission info of the mobile agent and looks for the computing resources and detailed service type requested and if they are available. In case of a paid service, the AEE examines the credit of the agent and compares it to the cost of the service requested. Finally, the AEE scans the agent code for malicious commands, and provides the environment for the agent to run.

The network communication is the third layer of the agent architecture. This layer is implemented by one process, the migration facilitator and provides transparent agent migration to the top layers. This process constantly runs in the background and listens for incoming and departing agents. In case of an incoming agent, the migration facilitator communicates with the

migration facilitator at the remote machine through an agent transport protocol to successfully receive the agent. For departing agents, the migration facilitator makes the request to the migration facilitator at the remote machine and sends the agent according to the transport protocol discussed in the next chapter.

The design and implementation of the Multimedia Transportable Agent System is divided into two major parts. The first part, called the Multimedia Agent Platform consists of the two bottom layers in the agent layered architecture. The second part, makes use of the Agent Platform to build and create Multimedia Agent Applications.

The next chapter presents the design architecture and the implementation of the platform. The platform is developed first, because it is common to all the agent-based applications built on top of it. The agent platform provides support for the mobile agents to migrate to and run on remote hosts and facilitates the development of agent based applications.

Chapter 4
Multimedia Transportable Agent Platform

4.1 Introduction

The architecture and the implementation of the multimedia agent platform are presented in this chapter. The presentation will focus on the various components of the platform which deal with the agent migration, agent structure, agent hosting and execution.

The multimedia agent platform's main focus is to facilitate the constrruction of agent-based appplications. The platform provides an interface between the agent-based applications and

the lower layers of the existing network systems that are optimized for applications of the RPC type. The platform has the following three main goals:

- Provide abstraction for agent based applications. The system should provide transparency for the basic agent actions: for instance, the network migration is reduced to the *jump* command.
- Support for multimedia applications. The transport protocol should recognize files of various media such as audio, video, and text and support moving these files when the agent requests to move to a new location. The AEE must also provide support for active agents to request and receive multimedia files from service agents. The mobile agent can then update their load with the received files.
- Allow the rapid developpement of applications. The support provided by the platform for the migration, and execution of multimedia enabled agents, helps rapid protoyping of agent applications. Developers do not have to worry about the basic agent actions and therefore can focus on the agent-based applications.

4.2 The Agent Structure

The structure of the agent (Figure 4.1) is divided into three main parts. The transport header, the agent header and the agent script. Each part corresponds exactly to one layer in the layered agent architecture. Every application built on top of the agent platform must apply this structure to its corresponding mobile agents.

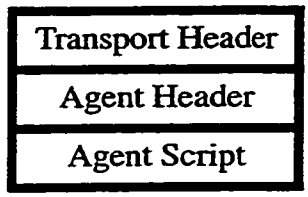

Figure 4.1 The agent structure

4.2.1 The Transport Header (TH)

The Transport Header corresponds to the migration layer in the system layered architecture. It is composed of two headers: the transport header 1 (TH1) and the transport header 2 (TH 2). TH 1 encapsulates the total agent package, including the agent file and all files in the agent load, and is sent only once with every agent migration. TH2 envelops every file sent in the network. When agents request moving to a new location, the migration facilitator connects to the migration facilitator at the destination machine and transfers the agent and load files, encapsulating each of those files with TH2 and envelops the whole with TH 1. The migration of agents is illustrated in figure 4.2.

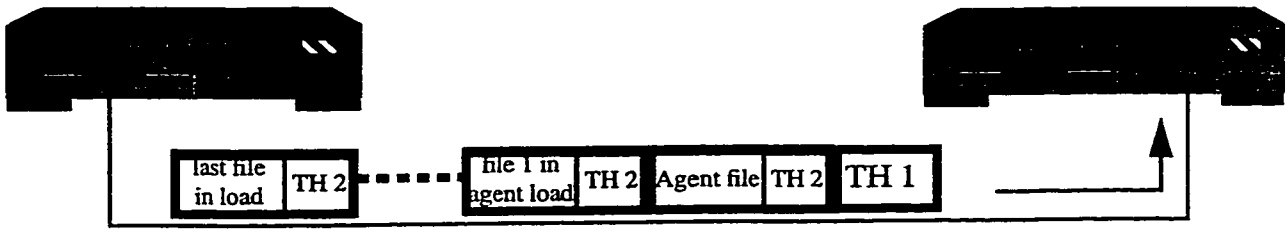

Figure 4.2 The agent migration

On the destination machine, upon receival of an incoming agent, the migration facilitator

decodes the transport header, and reads all the files in the agent load from the network. The migration facilitator reads first TH1, then for every file decodes TH2 and receives its corresponding file.

The TH1 contains general information of the agent package and has only two elements: The agent identification (AG-ID) and the number of files in the agent load (LL). TH1 is shown in Figure 4.3.

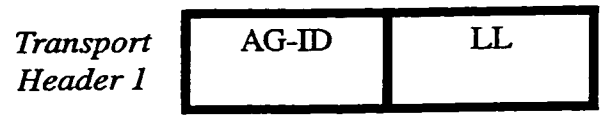

Figure 4.3 The Transport Header 1 (TH1)

- The AG-ID is unique for every agent in the network. It is set by the agent constructor before the agent executes its first hop in the network and never changes during the agent mission. The AG-ID has two parts. The first part is a number or a string that identifies the sender of the agent. the second part is a simple count that identifies mobile agents sent by the same user simultaneously in the network. If a user has an identification equal to A and he decides to send in the network simultaneously two agents with similar or different missions, the corresponding AG-IDs would have the values of A1 and A2 respectively. Upon receipt of the agent, the migration facilitator creates a directory AG-ID in the agent store and stores the agent and its load in that directory. Once the receival of the agent enclosure is completed, the migration facilitator informs the AEE of the new incoming agent and continues listening to more incoming agents.
- The LL indicates the total number of files in the agent load, including the agent file. This field

is dynamically updated by the agent script when the load is updated. For instance, if the agent collects n files during a mission, this field is equal to n+1.

The TH2 is sent with every file sent in the network and contains the necessary information for the migration facilitator to sucessfully read the file. TheTH2 consists of the following three elements: The type of the file (TF), the name, and the size. TH2 is shown in figure 4.4.

Figure 4.4 The Transport Header 2 (TH2)

- The TF determines the media type of the file. It is used to distinguish between agent files and the different media files in the agent load. The following table summerizes the type of media files recognized by this header field and the corresponding value of FT.

Type of File	*TF value*
agent	0
text	1
image/graph	2
audio	3
video	4

Table 4.1 TF possible values

- The name of the file is used by the migration facilitator to name the file being read from the

network and stored in the local agent repository.

- The size of the file is very important since all files belonging to one agent are transported sequentially in the same network socket. The migration facilitator reads one file at a time, starting by reading its TH2. It then creates a local file, pipes the file from the socket to the new created file, and then reads the next TH2 if any.

4.2.2 The Agent Header

The agent header corresponds to the agent execution layer. This header is built by the agent constructor before the agent migrates in the network. The agent header is divided into two parts: the user info and the mission info, and contains information related to the agent execution and migration such as the complete sender identification, network routing table, load size, hops completed, and required resources. Unlike the transport header, the agent header is part of the agent file only and does not enclose any of the the other files carried in the agent enclosure.

After successful migration, the migration facilitator informs the AEE of the complete receival of the agent. Before activation of the agent, the AEE extracts the agent information and the conditions and resources requested by the agent to run. If the requests meet the conditions of the hosting machine, the agent is provided permission to run.

The agent header contains the following 5 components: the user info (UI), the mission description (MD), the agent properties (AP), the route info (RI) and the agent state (AS).

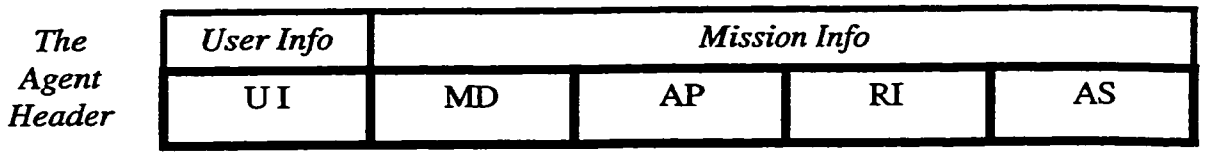

Figure 4.5 The agent header

- The UI contains information on the owner and sender of the mobile agent. As can be seen in figure 4.6, the UI is a structure with the following elements: user's name, company, AG-ID, machine IP address, and email address.

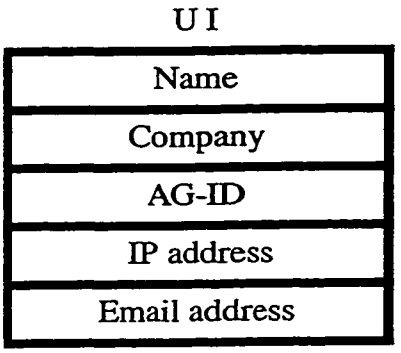

Figure 4.6 The UI header

- The MD contains the purpose of the agent's visit to the destination, and summerizes the goal of the agent script. The MD has 3 subfields (Figure 4.7): the mission, the keyword and the preferenced media. In the example of a mobile agent with the mission to search the Intranet for video documents on Africa, the different fields of the MD would be as follows: Mission: Search, Keyword: Africa, Preferenced Media: Video. The MD has a simple structure but could be enhanced in the future if applications running on top of the agent platform build agents

with more sophisticated missions.

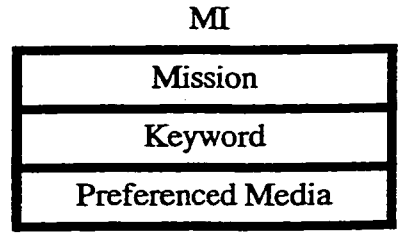

Figure 4.7 The MD header

- The AP header (figure 4.8), is divided into two main parts. The first part is addressed to the hosting AEE, and describes the resources and the conditions needed by the agent to run. These conditions are compared by the AEE with the policies of the host. If there are no conflicts, the agent is given the permission to run. The second part is a list of thresholds not to exceed by the mobile agent. These values never change during the agent journey. If the agent finds itself about to exceed any of these values, the execution is aborted, the agent requests to return to the home machine.

Addressed to the Hosting AEE: resources and conditions requested by the agent to run	Min Memory
	Min Disk Space
	Min Time to Run
	Interpreter type
	Credits
Thresholds and values not to exceed by the agent script	Max Number of docs to collect
	Max number of sites to visit
	Max number of bytes to carry
	Max time to spend on mission

Figure 4.8 The AP header

- The RI is a list of addresses to guide the agent during its mission. In case of an IP network, the RI is simply a list of IP addresses.
- The AS is the last category in the agent header and one of the most importants. It represents one of the major differences between the mobile agent technology and the "remote (batch) job processing" and "remote evaluation" both known in the software world for decades now. The agent state is part of the "intelligence" of the mobile agent, acting in the role of a virtual memory, in reminding the agent in the work that has been done and therefore guiding the agent in the future steps to do.

  In general, there are two levels of agent states that need to be differentiated:

  Firstly, If the application requires from the mobile agent to stop the execution at any point,

even in the middle of a transaction, and resume the execution after its migration from where it left off, then the agent has to capture the state at the interpreter level, saving the values of the stack and rebuilding the exact copy of the state of the agent and the interpreter in the new environment.

In the second case, similar to the agent system described in this thesis, the mobile agent does not need to stop the execution and to migrate in the middle of a transaction. At the end of any transaction, if the agent needs to migrate, there is no need to capture the state at the interpreter level, but only the state of the agent, information such as the number of bytes in the agent load, number of sites visited etc. The AS is shown in figure 4.9.

NS:Number of sites visited		
Sites	**Credits Spent**	
Site 1	Credit 1	
. .	. .	
Site NS.	Credit NS.	
NF: Number of files collected		
name	**size**	**type**
file 1	size 1	type 1
. .	. .	. .
file NF	size NF	type NF

Figure 4.9 The AS header

The first variable indicates the number of sites (NS) visited by the agent. This number is

updated by the agent everytime the migration to a new machine is successful. For every site visited, the Sites array records the network address, and the credits array contains the credits spent. The sites array permits to trace the agent itinerary while the credits array allow to compute the total credits spent so far by the agent. This number is compared to the max number of credits in the agent properties and determines if more payable services are possible.

Next in the AS, is the number of files (NF) collected in the agent load. For every file in the agent load, the Name array keeps its name, the Size array keeps its size in bytes and the Type array indicates the media type of the file.

4.2.3 Summary

Collecting all the different pieces described in the above sections, the agent structure as a whole is presented in figure 4.10.

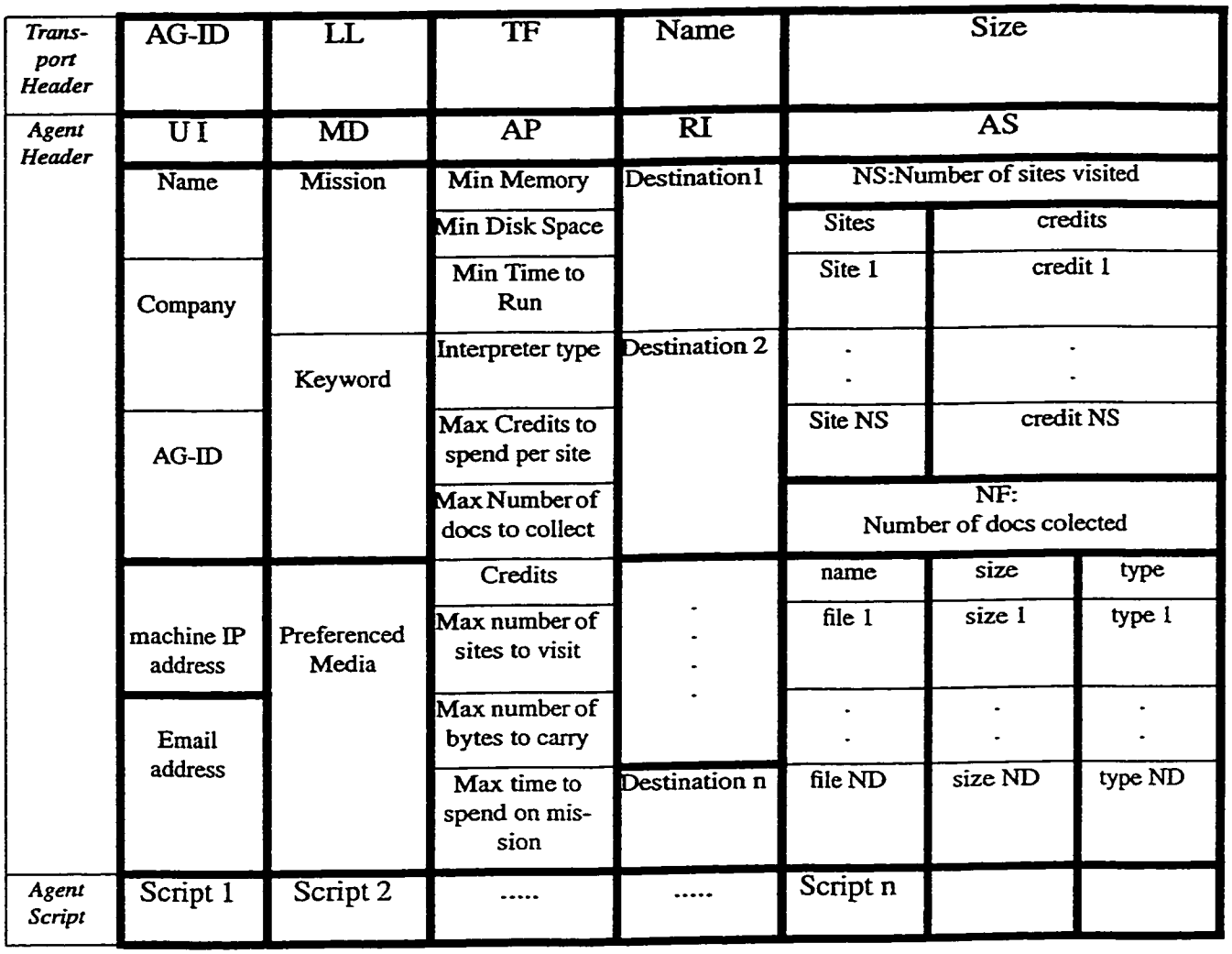

Figure 4.10 The complete agent structure

4.3 The Multimedia Agent Platform Architecture

The architecture of the platform (Figure 4.11), has three main components: the migration facilitator, the service agent and the AEE.

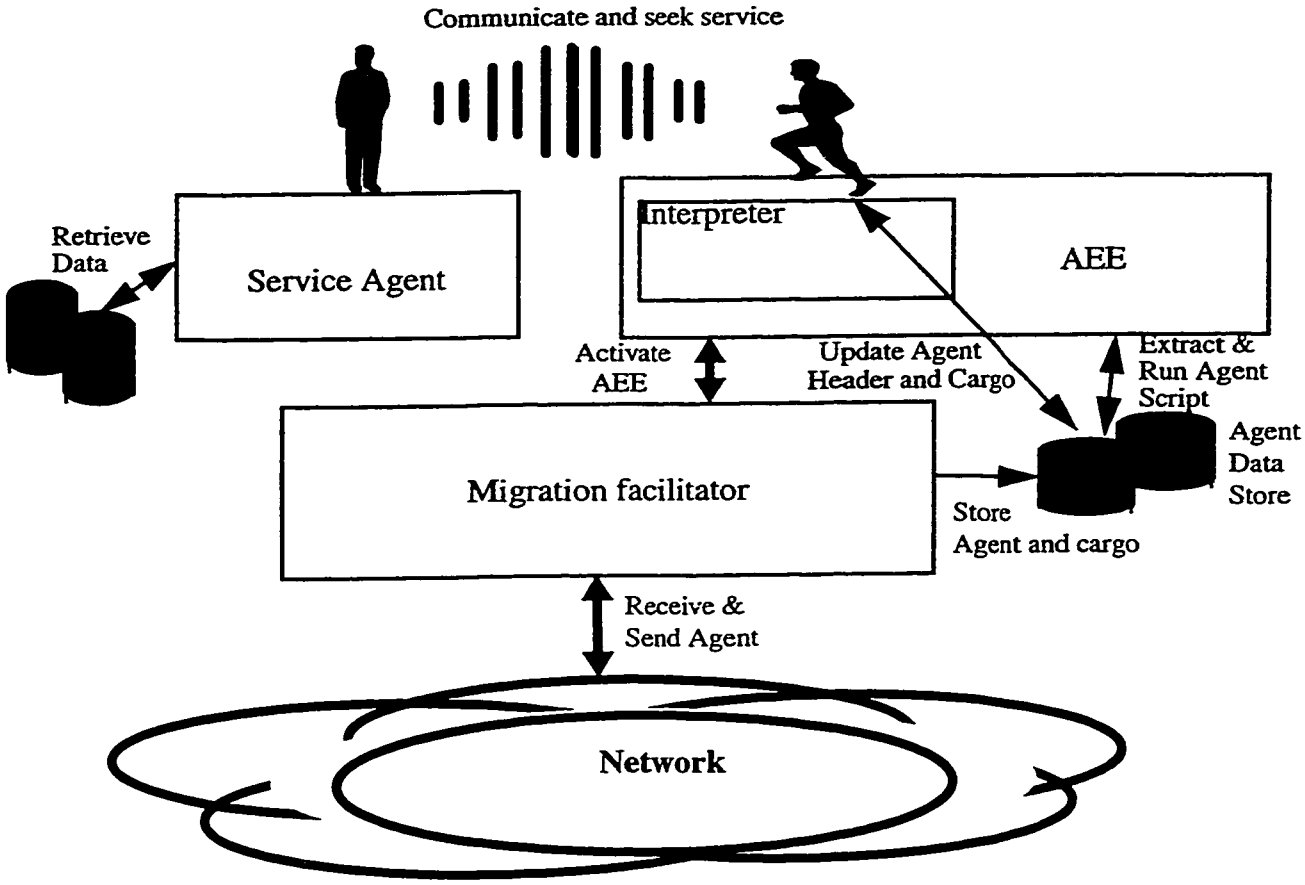

Figure 4.11 The functional architecture of the Multimedia Agent Platform

Along with the display of the processes, the arrows illustrate the interaction between the various processes and the comments besides the arrows explain the nature of the interactions. The migration facilitator for instance interacts with the network by constantly listening for incoming agents. Incoming agents are stored along with their cargo in the agent data store. If the receival is successful, the migration facilitator activates the AEE and goes back to the idle state which is listening for requests from migrating agents. The AEE extracts the agent header and script from the agent repository, and after examining the header and scanning the agent script, grants permission for the agent to run or denies the execution. If the permission is given, the

AEE starts an interpreter and allocate the computing resources for the agent script to run. The mobile agent communicates with the service agent and updates the agent load with retrieved files or info and updates the agent header accordingly.

4.4 The Migration Facilitator

The migration facilitator assumes the transport of the agent and its load. The agent load can include many files of different medias such as audio, video images and text.

The migration faciltator is the only process that constantly runs in the background. It is always active and listens on the network for incoming agents and internally for departing agents.

For every incoming agent, the migration facilitator reads the transport header according to the transport protocol, creates a directory in the agent repository and stores the agent and load files. If the migration of the agent is successful, the migration facilitator activates the AEE to look after the inactive agent and waits for new incoming agents. The flow-chart of the migration facilitator in the case of incoming agents is illustrated in figure 4.12.

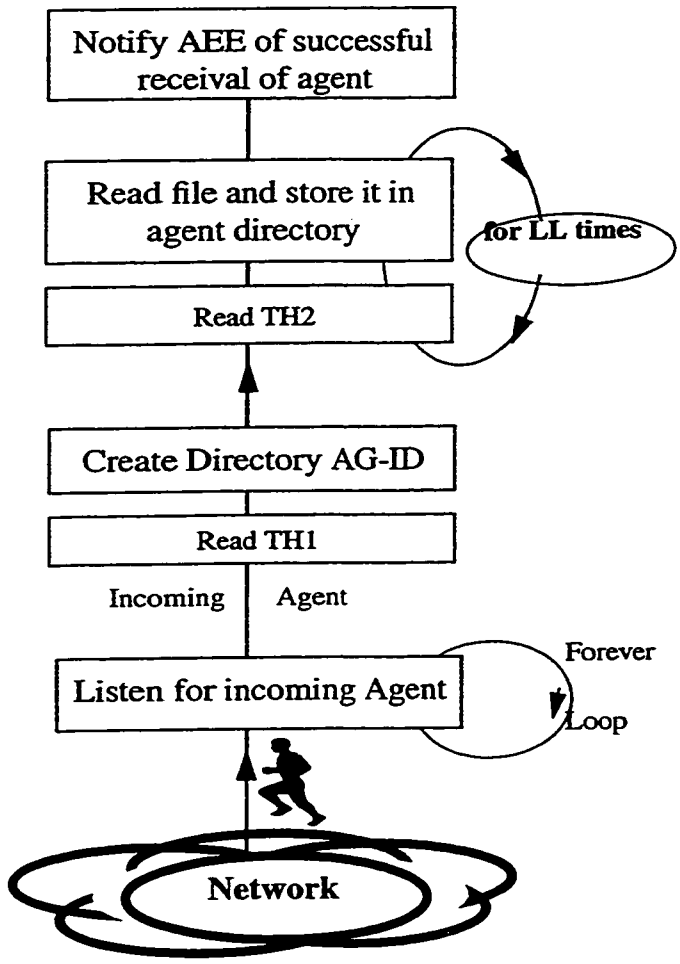

Figure 4.12 The flow chart of the migration facilitator for incoming agents

For every departing agent, the migration facilitator connects with the migration facilitator at the destination requested by the agent. If the connection is successful, the migration facilitator transfers the agent file and the files in the load to the new agent destination. Similarly to the receival of agents, the transfer of agents is done according to the transport protocol. TH1 is sent first, then TH2 is sent with every file sent in the network. Figure 4.13 illustrates the flow chart of the migration facilitator for departing agents.

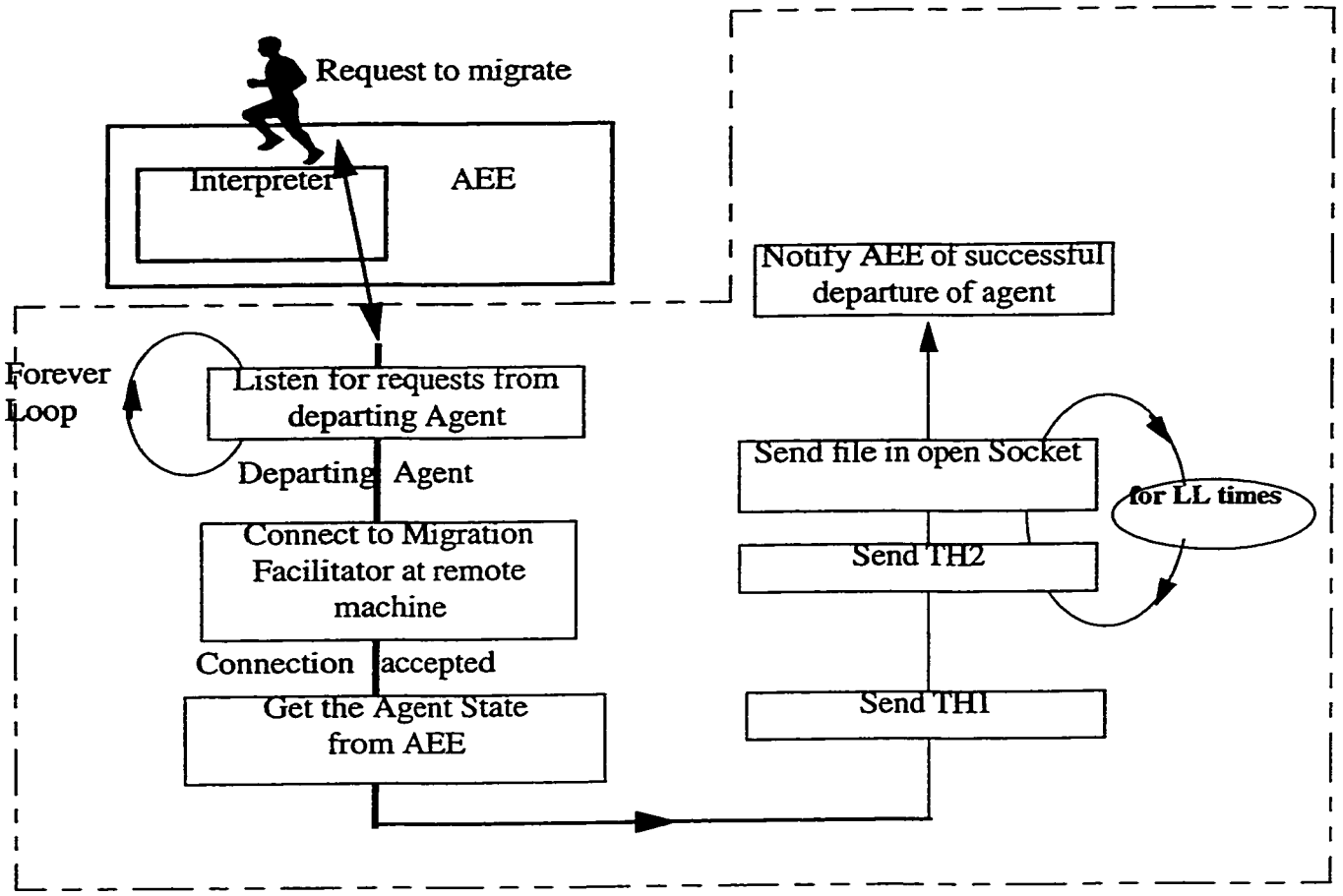

Figure 4.13 The flow chart of the migration facilitator for departing agents

4.5 The AEE

The AEE is notified by the migration facilitator, when the receival of an incoming agent is successful, to assume the activation and execution of the agent. At that time, the mobile agent is simply a directory created by the migration facilitator in the agent repository, grouping the agent file and the files of various media type carried in the agent load.

The AEE extracts the agent file and seperates the two main parts put together initially by

the agent constructor and updated by the agent script along the route: the agent header and the agent script. Before running the agent, the AEE has to ensure the following requirements:

- the security of the workstation
- the availability of the service requested by the agent
- the possibility to allocate the resources requested
- the credits in the agent account if the service requested is not free.

If the agent does not meet the requirements of the AEE, the agent is prohibited to run on the machine, and is simply asked to leave. The flow chart of the AEE is shown in figure 4.14.

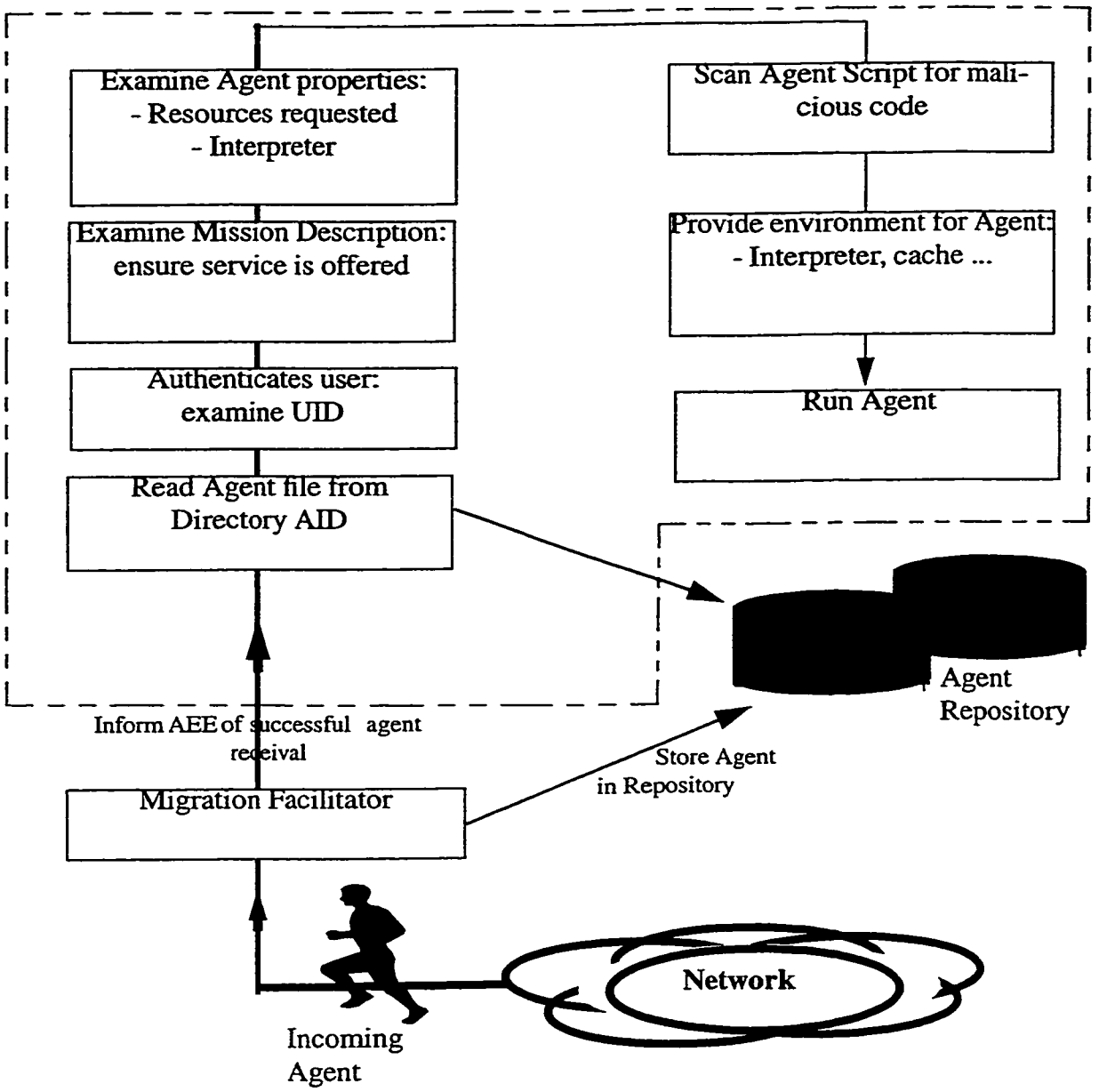

Figure 4.14 The flow chart of the AEE

4.6 Implementation

The implementation of the migration facilitator and the AEE which together constitute

the multimedia agent platform is presented in this section. The migration facilitator is the lowest layer in the platform layered architecture and therefore is explained first.

The first prototype of the agent platform is implemented on Pentium Pro 200 Mhz and Pentium 133 Mhz PCs running Windows 95 operating system. Each computer has a 10 Mbps Ethernet connection and a 155 Mbps ATM connection. See figure 4.15.

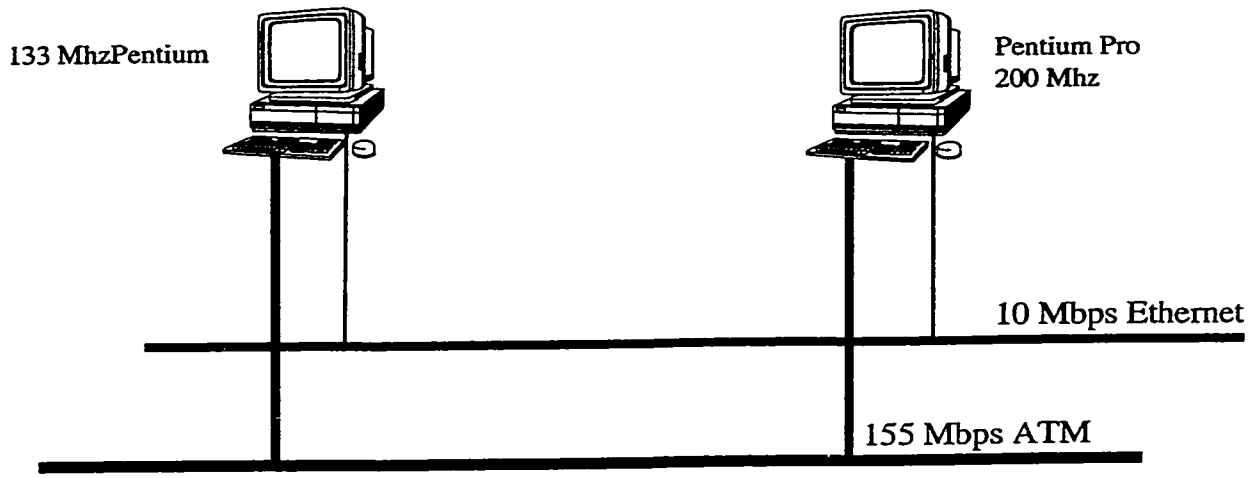

Figure 4.15 Hardware set-up for the implementation

The programming language used to implement the migration facilitator is C using the compiler Borland C++ version 5.0. The AEE is written in Tcl version 8.0, which is also used to define the agent script. Tcl is a string-based command language and is freely available from the Internet.

4.6.1 The Migration Facilitator

This section discusses the implementation of the migration facilitator. This process handles the migration of agents between machines. Agents can carry a load of different media files and need support when leaving a machine, and reception at the destination machine. The migration facilitator is composed of two independant processes. The first process is the reception manager (Figure 4.12), and the second (Figure 4.13) is the departing manager.

The first implementation of the migration facilitator was written in Tcl 8.0. Tcl 8.0 supports communication and transfer of files over the network through TCP/IP sockets. A Socket is an endpoint of communication to which a name may be bound. TCP/IP has been selected for its wide use in Intranets. Also, the target of agent applications is the public and open networks which seem to converge towards a common base protocol: the IP. Tcl was chosen for its easy syntax, and for rapid prototyping. In the testing of this first prototype, the reception manager and the departing manager are able to open sockets and to communicate and exchange information, but the migration process fails in the transfer of the agent load. When an agent requests to migrate to a new destination, the departing manager opens a network socket and connects successfully to the reception manager that constantly listens for incoming agents. During the transfer of the mobile agent, the transport headers are filled and sent correctly, however a problem occurs in the transfer of the binary files in the agent load. Further testing and research prooved that Tcl is a string-based language, and has no support for binary files. The files in the agent load are of different media types and cannot be treated as text files. After opening the files of binary formats like videos, audios and images, the departing manager is unable to read and send the files correctly.

In the second prototype, the reception and departing managers are written in C and use the Windows Sockets (winsock) for the communication and transfer of files. The Windows Sockets specification defines a network programming interface for Microsoft Windows which is based on the socket paradigm popularized in the Berkeley Software Distribution (BSD) from the University of California at Berkeley. the Berkeley Sockets programming model is the de facto standard for TCP/IP networking.

4.6.2 The Reception Manager

The reception manager is responsible for constanly listening in to a particular port for incoming agents. Agents make requests to migrate to the new machine through the departing manager, which contacts the reception manager at the destination and transfers the agent and its load according to the transport protocol.

The primitives of the migration facilitator reception's manager are listed in table 4.2. These primitives are offered to all the layers above the migration facilitator in the agent system layered architecture: the AEE, the mobile agent and the agent-based applications.

primitive	Description	parameters
wait_for_client	Waits on a socket for a connection request from a client. and determines the status of the socket.	int Sock;
set_server_connection	Creates a new socket for the communication with the client and returns a handle to the new socket	int Sock;
read_TH_1	reads from a socket the tigration header 1	int Sock; char *agid, *agnb; int *cargo;

read_TH_2	reads from a socket the tigration header 2	int Sock;
create_agent_dir ectory	Creates a diretory for the agent in the Agent repository	char *agid, *agnb;
read_file	reads from a socket a file and stores it in the agent directory	int Sock;
init_WinSock_D LL_Version	initialization	-
init_TCP_Socket	initialization	-

Table 4.2 The primitives of the migration facilitator reception's manager

Figure 4.16 shows the main program of the reception manager which is built according to the flow chart illustrated in figure 4.12 and makes use of the primitives in the table 4.2. If the initializations of the WinSock library through **init_WinSock_DLL_Version()** is successful and the socket to listen for incoming connection through **init_TCP_Socket** is established, the reception manager waits for incoming agents through an infinite while loop. For every incoming agent, a new socket is created and returned by **wait_for_client,** the mother socket being used only for incoming requests and not for transfer of agents. **read_TH1** reads from the new socket created, the agent identification and the number of files in the agent cargo. **create_agent_directory** creates in the agent repository a directory to receive the agent entity. Then, in a while loop, for every file in the agent entity **read_TH2** is invoked to read the TH_2 and **read_file** is invoked to read and store the file in the agent directory. Finally, the new socket is closed and **inform_AEE** is called to activate the AEE after the reception of the agent.

```
void main()
{
SOCKET sock, newsock;
char agid[5], agnb[3], char doctype[2], char docname[21], char doclength[11];
int recu, count, code, *cargo;

printf("Reception Manager: init in process...\n");

if ( init_WinSock_DLL_Version == 0 )
      printf("WinSock initialization complete\n");
else
{
      printf("error in WinSock initialization, exiting ...\n");
      exit(0);
}

sock = init_TCP_Socket;
if (sock == -1)
{
      printf("error in Socket initialization, exiting ...\n");
      exit(0);
}

printf("Listener: initialization completed!\n");

/* Infinite loop, wait for Departure Managers requests to connect....  */

while(1)
   {
      newsock = wait_for_client(sock);
      switch (newsock)
      {
            case error1:
              printf("invalid socket.Listener: Can't establish connection\n");
              break;
            case error2:
              printf("Can't establish connection. Timed out\n");
              break;
            default:
/* Read Migration Header 1 */
              read_MH_1(newsock, agid, agnb, cargo);
/* Create Agent Directory */
              create_agent_directory(agid, agnb);

              count = 0;
              while (count < cargo)
              {
                 read_MH_2(newsock, agid, agnb, doctype, docname,  doclength);
                 read_file(newsock, agid, agnb, doctype, docname,  doclength);
                 count++;
              }

            printf("closing socket\n");
            close(newsock); /*** close child_sock in parent ***/
```

```
            inform_AEE(agid, agnb);
            break;
        }
    }
}
```

Figure 4.16 Main program of the Reception Manager

The function **init_WinSock_DLL_Version()** (Figure 4.17) calls the windows function WSAStartup(). This function must be the first Windows Sockets function called by an application or DLL. It allows to an application to specify the version of Windows Sockets API required and to retrieve details of the specific Windows Sockets implementation. The reception manager may only issue further Windows Sockets API functions after a successful WSAStartup() invocation. Upon entry to WSAStartup(), the Windows Sockets DLL examines the version requested by the application. If this version is higher than the lowest version supported by the DLL, the call succeeds and the DLL returns in wHighVersion the highest version it supports and in wVersion the minimum of its high version and wVersionRequested. The Windows Sockets DLL then assumes that the application will use wVersion. If the wVersion field of the WSADATA structure is unacceptable to the reception manager, it calls WSACleanup(), fails to initialize and returns -1.

```
int init_WinSock_DLL_Version()
{

WORD wVersionRequested;
WSADATA wsaData;
int err;

wVersionRequested = MAKEWORD( 1, 1 );
err = WSAStartup( wVersionRequested, &wsaData );

if ( err != 0 )
```

```
{
/* Tell the user that we couldn't find a usable  winsock.dll.  */
    printf("could not find a usable winsock.dll, exiting...\n");
    return -1;
}

/* Confirm that the Windows Sockets DLL supports 1.1.Note that if the DLL
supports versions  greater than 1.1 in addition to 1.1, it will still return
1.1 in wVersion since that is the version we requested. */

if ( LOBYTE( wsaData.wVersion ) != 1 || HIBYTE( wsaData.wVersion ) != 1 )
{
      /* Tell the user that we couldn't find a useable  winsock.dll. */
      printf("could not find a usable winsock.dll, exiting...\n");
      WSACleanup( );
      return -1;
}

/* The Windows Sockets DLL is acceptable. Proceed. */
return 0;
}
```

Figure 4.17 Implementation of the init_WinSock_DLL_Version() function

The function **init_TCP_Socket()** is presented in figure 4.18. This function is a typical function for applications/servers that initializes a socket to listen to a port on a TCP/IP network. This function calls the three main functions for registering a socket, **socket()**, **bind()** and **listen()**. **socket()** allocates a socket descriptor of the specified address family, data type and protocol, as well as related resources. When a socket is created with **socket()**, it exists in a name space (address family), but it has no name assigned. **bind()** establishes the local association (host address/port number) of the socket by assigning a local name to an unnamed socket. **listen()** applies only to sockets that support connections, i.e. those of type SOCK_STREAM. The socket is put into "passive" mode where incoming connections are acknowledged and queued pending acceptance by the process. listen is typically used by servers that could have more than one connection request at a time. init_TCP_Socket() returns the socket description upon success or -1 in case of an error in creating and initializing the socket.

```
SOCKET init_TCP_Socket();
{
struct sockaddr_in server_def;
SOCKET sock;
int err;

memset( (char*)&server_def, 0, sizeof(server_def));

/* Initialize sockaddr_in struct with server(local) socket name */
server_def.sin_family = PF_INET;
server_def.sin_port = htons(3555);
server_def.sin_addr.s_addr = IN_ADDR_ANY;

/* Ask for a stream TCP/IP socket */
sock = socket(PF_INET, SOCK_STREAM, 0);

if (sock == INVALID_SOCKET)
      {
      printf("error in opening socket = %u\n",sock);
      err = WSAGetLastError();
      printf("error = %d\n", err);
      return -1;
      } else {
      printf("socket succeeded, value = %d\n",sock);
      }
}
/* Name the local socket with the values in the sockaddr_in structure */

if (bind(sock, (struct sockaddr *)&server_def, sizeof(server_def)) < 0)
      {
      perror("bind");
      close(sock);
      return -1;
      }

/* The server prepares to accept a connection attempt from a TCP client
      by calling listen() function to the named unconnected socket */

if    ( listen(sock, 5) == SOCKET_ERROR)
      {
      perror("listen");
      return -1;
      }

/* socket established to listen for incoming connection. */
return sock;
}
```

Figure 4.18 Implementation of the init_TCP_Socket() function

The function **wait_for_client**, displayed in figure 4.19, is called inside the infinite main loop of the departure manager. This function listens to the socket created and returned by **init_TCP_Socket()**. To determine the status of the main socket, a call for the winsock function **select()** is done. **select()** returns:

- if no error occurs, the total number of descriptors which are ready and contained in the fd_set structure which indicates the set of sockets for which a given status is requested.
- 0 if the time limit expired.
- SOCKET_ERROR if an error occurred.

In case of an error or a time out, **wait_for_client** returns with the error number: -1 for SOCKET_ERROR and 0 if the time limit expired. The default case, when no error is returned by select, **wait_for_client** calls the function **set_server_connection** (Figure 4.20).

set_server_connection accepts a connection on a socket by calling **select**. This routine extracts the first connection on the queue of pending connections on a socket s, creates a new socket with the same properties as s and returns a handle to the new socket, otherwise, a value of INVALID_SOCKET is returned. The accepted new socket may not be used to accept more connections. The original socket remains open.

```
int wait_for_client(int sock)
{
fd_set sock_set;
struct timeval wait;
int desc;

wait.tv_sec = TIME_OUT;
wait.tv_usec = 0;
FD_ZERO(&sock_set);
```

```
FD_SET(sock, &sock_set);

switch(select(FD_SETSIZE, &sock_set, (fd_set *)NULL, (fd_set *)NULL, NULL))
      {
      case SOCKET_ERROR:
            perror("select in wait_for_client");
            desc = -1;
            break;
      case 0:
            desc = 0;
            break;
      default:
            if (FD_ISSET(sock, &sock_set))
                  desc = set_server_connection(sock);
            else
                  {
                  printf("select terminated, ???");
                  desc = -1;
                  break;
                  }
      }
return(desc);
}
```

Figure 4.19 Implementation of the wait_for_client() function

```
int set_server_connection(int sock)
{
struct sockaddr_in pc_addr;
register int  desc;
int length;

length = sizeof(pc_addr);
memset((char *)&pc_addr, 0, length);
if  ((desc  =  accept  (sock,  (struct  sockaddr  *)&pc_addr,  &length))==
INVALID_SOCKET)
      {
      perror("accept");
      return(-1);
      }
else
      return(desc);
}
```

Figure 4.20 Implementation of the set_server_connection() function

Once the connection with the departure manager is established through the new socket

created, **read_TH_1** is called to read the TH1. **recv** is used to read data from the socket. agid and agnb have fixed length, 4 and 2 bytes respectively. The agid and agnb variables are used by **create_directory** to create a directory in the agent reporsitory to store the agent. The cargo variable which is equivalent to the number of files in the agent load, can have variable length depending on the number of files in the agent cargo. As a consequence, the cargo_header has a preheader of a length equal to 1 byte to indicate the number of bytes necessary to read the cargo header.

```
read_TH_1(SOCKET newsock,char agid[5], char agnb[3], int *cargo)
{
char cargo_header_len[2], cargo_header[3];

/* Read Agent-ID = 4 bytes and AG-NB = 2 bytes */

recu = recv(newsock, agid, 4, 0);
agid[4] = NULL;
recu = recv(newsock, agnb, 2, 0);
agnb[2] = NULL;

/* Read the number of docs sent with the agent file */
recu = recv(newsock, cargo_header_len, 1, 0);
cargo_header_len[1] = NULL;
len = atoi(cargo_header_len);
recu = recv(newsock, cargo_header, len, 0);
cargo[len] = NULL;
*cargo = atoi(cargo_header);
}
```

Figure 4.21 Implementation of the read_TH1() function

Once the TH1 is read and the directory to store the agent in the agent repository is created, the main program enters the main loop. For every file in the agent load, including the agent file, **read_TH2** is called prior to call **read_file. read_TH2** reads the type , the name and the length of the file. These three variables constitute the TH2 of the agent.

```
int read_TH2(int sock, char agid[5], char agnb[3], char doctype[2], char
docname[21], char doclength[11] )
{
int rec, entier;
char  len_name[4], len_name_len[2], len_len[2];

/*  Read and determine the type of file. 1 byte */
rec = recv(sock, doctype, 1, 0);
doctype[1] = NULL;

/*  Determine the name of the file
    The Length of the name length = len_name_len = 1 byte
    The name length = name_len = (len_name_len) bytes
    docname = document name = (name_len) bytes */
rec = recv(sock, len_name_len, 1, 0);
len_name_len[1] = NULL;
entier = atoi(len_name_len);
rec = recv(sock, len_name, entier, 0);
len_name[entier] = NULL;
entier = atoi(len_name);
rec = recv(sock, docname, entier, 0);
docname[entier] = NULL;

/*  Determine the length of the file
    Length of length = len_len = 1 byte
    Length of file = doclength = (len_len) bytes */
rec = recv(sock, len_len, 1, 0);
len_len[1] = NULL;
entier = atoi(len_len);
rec = recv(sock, doclength, entier, 0);
doclength[entier] = NULL;
}
```

Figure 4.22 Implementation of the read_TH2() function

After the receival of the TH2, all the information to read the file in the agent load are captured. **read_file** (Figure 4.23) creates first in the agent repository a file to store the information received from the socket. The receival of the file is done in packets of 1000 bytes. The number of packets, represented in the variable *nb_segments* is calculated by dividing the length of the file, *len*, by 1000. Since the length of the file is not necessarly a multiple of 1000, *rest* represents the number of bytes to receive in the last packet, after the receival of *nb_segments* packets of 1000 bytes each.

The packet's size of 1000 bytes has been chosen after several testings and trials. TCP does not guarantee when an application sends n bytes in a socket the receival of the exact amount of bytes at the receiving end. It is the responsability of the application to ensure the success of the data delivery, especially when large files are being tranferred over a busy network. The first intuition was to read all the file, *len* bytes at once, using one call to **recv** . This method did not result in the complete receival of the large files. The second trial consisted of the reading of the file byte by byte, in a for loop, making *len* calls to **recv**. Although successful, this second trial resulted in very poor performance. After several trials and errors, the packet set at 1000 bytes seemed the most stable, performant and reliable.

```
read_file(int sock, char agid[5], char agnb[3], char doctype[2], char
docname[21], char doclength[11])
{
char filename[300], *arr;;
int len, value, nb_segments, rest, index, recu;
FILE *agf;

/* For every file, fill filename[300] with the right path (agent directory) */
strcpy(filename, "c:\\users\\francois\\firstimp\\service\\");
strcat(filename, agid);
strcat(filename, agnb);
strcat(filename, "\\");
strcat(filename, docname);
/*  check if the file is an agent and add .agt if yes */
if ( strcmp(doctype, "0") == 0 )
strcat(filename, ".agt");
/*  Open the file in the agent directory */
agf = fopen(filename, "ab");
if (agf == NULL)
{
      fprintf(stderr, "Cannot open input file.\n\n");
      return -1;
};

len = atoi(doclength);

/* Read the file in segments of 1000 bytes. TCP does not guarantee delivery of
```

```
large packets */
value = 1000;
// create and initialize buffer "arr"
arr = (char FAR *)malloc((int)value*sizeof(char));
for (index=0; index++; index<value)
arr[index] = 0;
arr[0] = 0;

nb_segments = len/value;
rest = len - value*nb_segments;
index = 0;

for (index=0; index<nb_segments; index++)
      {
      recu = recv(sock, arr, value, 0);
      recu = fwrite(arr, value, 1, agf);
      };

if (rest > 0)
      {
      recu = recv(sock, arr, rest, 0);
      recu = fwrite(arr, rest, 1, agf);
      }

if (recu == 0)
{
      printf("the connection has been closed, couldn't read sock\n");
      fclose(agf);
      free(arr);
      return -1;
}
else if (recu == SOCKET_ERROR)
{
      printf("socket error, couldn't read sock\n");
      fclose(agf);
      free(arr);
      return -1;
}
else
{
      printf("data read successfully");
};

fclose(agf);
free(arr);
return 0;
}
```

Figure 4.23 Implementation of the read_file() function

Once all the files in the agent entity are successfully received and stored in the agent repository, the connection with the machine that previously hosted the agent is closed. The agent

is ready now to be activated, a call is made to **inform_AEE** (Figure 4.24).

inform_AEE creates a Tcl interpreter by calling **Tcl_CreateInterp**, and launches the AEE, which is written in Tcl, by calling **Tcl_EvalFile.** The AEE exctracts the script to activate and creates a new file with the script extracted. The new file created, which is the mobile agent, is then activated by calling **Tcl_EvalFile.**

```
inform_AEE(char agid[5], char agnb[3])
{
Tcl_Interp *inter;
int code;
char dirname[300];

/* Create an interpreter, and then run the Agent Execution Environment AEE */
inter = Tcl_CreateInterp();
code=Tcl_EvalFile(inter, "c:\\users\\francois\\firstimp\\serveur\\AEE.tcl" );

if (code != TCL_OK)
   {
     printf("An error occured in the TCL script1, error = %s\n",inter->result);
     return -1;
   }

printf("calling the agent.tcl which is the agent\n");
strcat(dirname, "\\");
strcat(dirname, agid);
strcat(dirname, ".tcl");
code = Tcl_EvalFile(inter, dirname);
if (code != TCL_OK)
   {
     printf("An error occured in the TCL script3, error = %s\n",inter result);
     return -1;
    }
}
```

Figure 4.24 Implementation of the inform_AEE() function

4.6.3 The Departure Manager

The departure manager is responsible for listening to requests from active agents to migrate to a new machine. The departure manager then contacts the reception manager at the destination and transfers the agent and its load according to the transport headers and protocol.

The request to migrate is made by the active agent through a command in the agent script. As a consequence, the request has been developed as a new Tcl command and has been added as a command to the set of commands in the agent communication protocol described in section 5.3. The active agent communicates this command to the departure manager as a request to migrate to a new machine.

The implementation of the departure manager is very similar to the implementation of the reception manager with few changes. The initialization of the WinSock DLL and the socket are done with the same functions, **init_WinSock_DLL_Version()** and **init_TCP_Socket()**. The departure manager does not have an infinite loop for listening at a socket, requests come internally from the mobile agent. After a request for migration, the migration facilitator initializes a TCP socket, ans uses **connect** to open the communication with the reception manager at the remote machine. Unlike the reception manager, one socket only is used for the migration of the agent.

Once the communication succeeded, the transport manager sends the TH1 for the agent then sends, for every file in the agent repository, the TH2 and the file. The **send** command is used to send information in the socket. The following code is an extract from the departure manager code. The code in figure 4.25 opens a file in the agent repository, and sends it the remote machine.

```
//open the file
handle1 = fopen(path, "rb");
      if (handle1 == NULL)
      {
            printf("Cannot open input file.\n");
            return 1;
      }

// allocate a buffer and initialize it to 0
buf = (char FAR *)malloc((int)(length+1)*sizeof(char));
for (ind=0; ind++; ind<=length)
buf[ind] = 0;

// read into buf, from handle1, rlen number of bytes
rlen = fread(buf, 1, length, handle1);

// Send the file buffered in buf, in the socket s
sent = send(s, buf, rlen, 0);
free(buf);
fclose(handle1);

closesocket(s);
```

Figure 4.25 Extract from the implementation of the Departure Manager

4.6.4 The AEE

The AEE is responsible for running the inactive mobile agent, received and stored in the agent repository by the reception manager. In order to do so, The AEE has to ensure first, that the agent meets all the requirements of the receiving host. The AEE access the agent file, and examines the agent header. The agent's intentions, requirements and code are observed prior to run the agent.

The AEE code is written completely in Tcl. The AEE does not handle binary files, and in such case, Tcl is practical and powerful in handling text file, and strings. The AEE implementa-

tion is directly related to the agent header. Since the mobile agent is running only in the MIRL Intranet, and not in public networks, the agent header has been simplified, to simplify and speed up the implementation of the AEE. The agent header in the current prototype looks like the following figure.

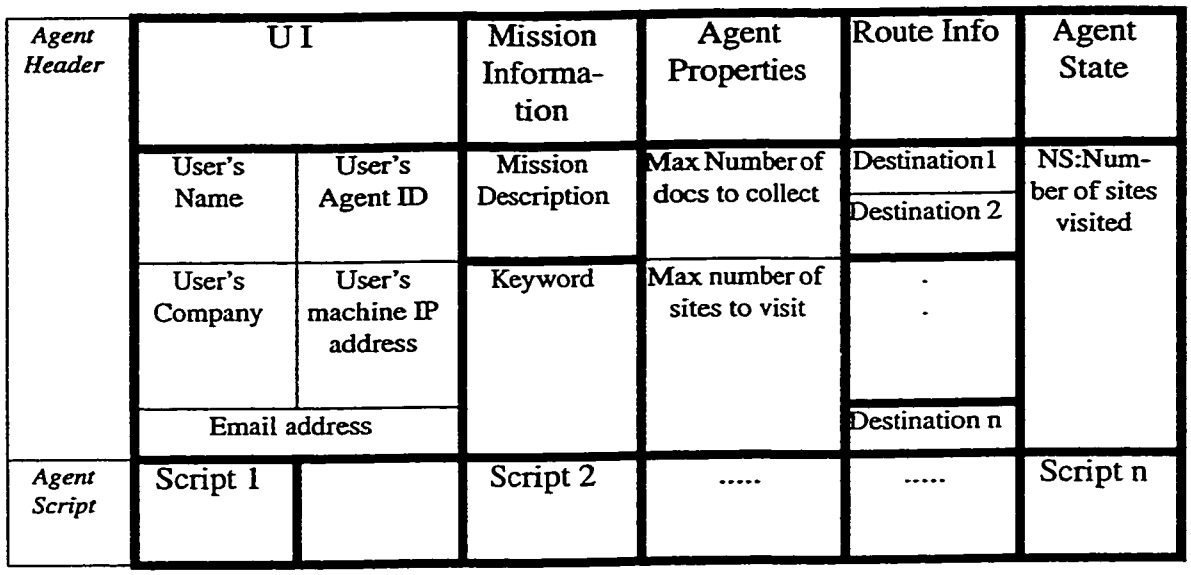

Figure 4.26 The simplified agent header

After the complete receival of the agent, the AEE examines the simplified Agent header. After ensuring that the user is not restricted access to the services (by comparing the user's info to a file of banned users and companies) and that the mission is available (by looking at the Mission Description) the AEE extracts the appropriate script and loads it into the extended interpreter for execution. The applicable script for the local machine, among the n scripts in the agent file, is found after the reading of the number of sites visited, NS. For example, If the agent has visited 2 machines before the current host, then the AEE exctracts Script 3 for execution. The code in Figure 4.27 is an extract from the AEE code. This code opens the agent file and extracts some of the

info from the agent header and then extracts the agent script to be executed.

```
# Open the Agent file in order to read the info needed
  set agfile [open "$agentdir/$agid$agnb.agt" r]
# agstring contains the total agent structure
  set agstring [read $agfile]
  seek $agfile 0

# Read the following information:
# Mission, keyword, max-docs, docs-found, max_sites, sites-visited

while { [gets $agfile line] >= 0} {
      foreach attribut { mission max-docs AG-ID docs-found max_sites sites-
visited} {
            set comparer [ string last $attribut $line]
            if { $comparer >= 0 } {
                  set longueur [string length "$attribut"]
                  set agent($attribut) [ string range $line [expr $comparer +
$longueur +2] end]
                                    }
                                                        }
                                 }
close $agfile

## Check that all the conditions to let the agent execute, otherwise send the
agent home
....
# All the conditions are met, Execute the right script,
# Read first the script for this site and write it to a file
set agscript [open $agentdir/$agid.tcl w]
set number [expr $agent(sites-visited) + 1]

set comparer [ string last "sos$number" $agstring]
if { $comparer >= 0 } {
                  set longueur [string length "sos$number"]
                  set startindex [expr $comparer + $longueur]
                        } else {
                  puts stdout "sos$number not found"
                  return TCL_ERROR
                              }

set comparer [ string last "eos$number" $agstring]
if { $comparer >= 0 } {
                  set longueur [string length "eos$number"]
                  set endindex [expr $comparer - 1]
                        } else {
                  puts stdout "eos$number not found"
                  return TCL_ERROR
                              }
set script [string range $agstring $startindex $endindex]
```

Figure 4.27 Extract from the implementation of the AEE

Chapter 5
Multimedia Agent - Based Applications

5.1 Introduction

This chapter describes the implementation of three multimedia agent-based applications. These applications are developed using the mobile agent paradigm, and allow to users to asynchronously list by keyword, retrieve and send multimedia documents. The applications are built on top of the Multimedia Agent platform presented in chapter 4, and therefore employ the agent structure defined in the platform and use the services of the migration facilitator and AEE for the transfer and the hosting of agents.

This chapter is organized as follows. The service agents used at the service sites to com-

municate with mobile agents through the ACL and execute their requests are first presented. Then, the ACL used by the service agents and mobile agents to communicate is explained. Further, the Agent Constructor developed to build the agent according to the agent structure and depending on the agent mission is detailed. Finally, the GUIs for building and sending the agents, the mode of operation, and the presentation of the agent mission results are presented.

5.2 The Service Agents

The service agents have been mentionned briefly in the section 4.3, in the platform architecture. The service agents interact locally with the active mobile agent and provide their services. In agent-based applications, the service agents replace the servers, used in the client/server model.

In the mobile agent paradigm, the requests for services are all done locally on the same machine and the remote procedure calls are replaced by a set of commands that constitute the ACL. The service agent executes the commands requested by the mobile agent and returns the results.

The service agents are also used as a security tool. Mobile agents cannot access resources directly on remote machines, especially that today's technology does not guarantee detecting all the malicious code that an agent could carry by scanning the binary code.

Service agents can have an infinite role in public networks: database queries, WEB file servers, yelow pages administrators. In the case of the three applications described in this chapter,

the service agents have the role of searching a file system, retrieval of files and presentation of documents.

The service agents are not implemented as stand alone processes, but part of the interpreter. When the AEE exctracts the script to execute from the agent file, it loads the commands written in C into the Tcl interpreter created for the agent, creating a service agent as part of the Tcl interpreter.

5.3 The Agent Communication Protocol (ACP)

This section describes the interaction between active agents. This interaction can occur between two mobile agents or between a mobile agent and a service agent. The interaction is done through a set of commands that constitute the ACP. Every command sent from Agent A to Agent B, is a call to a procedure in the Agent B, which executes internally the procedure and returns the results to Agent A.

The ACP developed in this thesis is uni-directional and is limited to the interactions between the active mobile agent and the service agent. The ACP is the list of all the commands in the agent script which are sequentially executed and communicated to the service agent. The commands in the agent script are chosen from the list of all the commands in the Tcl package, plus all the additional service specific commands that are created in C and incorporated into the Tcl interpreter.

One of the main advantages of Tcl, and the reason for choosing it as the base for the agent communication language, is the possibility to extend the core Tcl features with additional application-specific commands in C. The Tcl parser of the extended interpreter can thus execute all the core Tcl commands plus the service specific commands requested by the mobile agent. Figure 5.1 shows how the new commands called by the mobile agent script are parsed in the extended Tcl interpreter. First, the functions that implement the new commands are initialized and loaded. Then, the agent script is passed through a command loop for parsing. The parser has access to the core Tcl commands plus all the new service specific commands loaded into the interpreter.

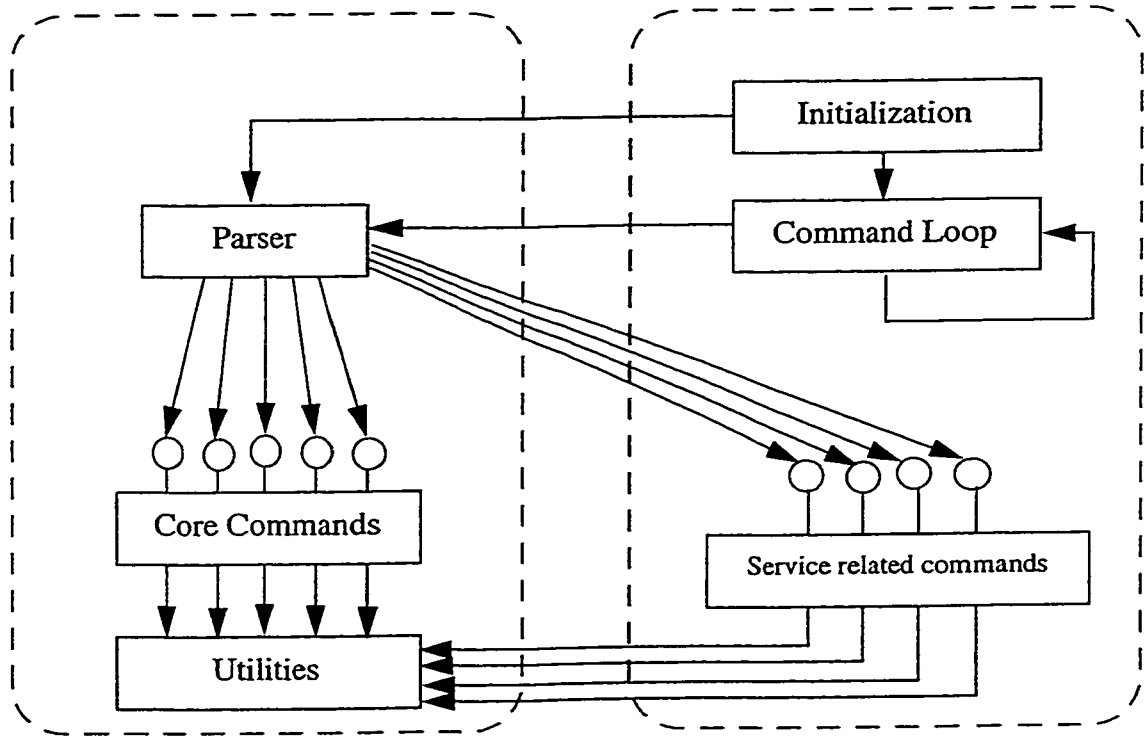

Figure 5.1 The extented Tcl interpreter

The code in figure 5.2, implements a new Tcl command, **Ag_update**. The C code is written in Borland C++. All the new other commands of the communication protocol, created and added to the tcl interpreter, have a similar structure.

```
/* AG_Update.c --
   This file is an Example of a Tcl dynamically loadable extension. */

/* List all the include files here */
#include <tcl.h>
        .
        .

/* Declarations for functions defined in this file. */
static int  AG_UpdateCmd _ANSI_ARGS_((ClientData clientData,
                        Tcl_Interp *interp, int argc, char **argv));

EXTERN EXPORT(int,AG_Update_Init) _ANSI_ARGS_((Tcl_Interp *interp));
/*----------------------------------------------------------------------
 * AG_Update_Init --
 * This procedure initializes the AG_Update command.
 * Results:A standard Tcl result.
 * Side effects: None.
 *----------------------------------------------------------------------*/

EXPORT(int,AG_Update_Init)(interp)
        Tcl_Interp *interp;
{
        Tcl_CreateCommand(interp, "AG_Update", AG_UpdateCmd, NULL, NULL);
        return Tcl_PkgProvide(interp, "AG_Update", "1.0");
}

/*----------------------------------------------------------------------
 * AG_UpdateCmd --
 *    This function implements the Tcl "AG_Update" command.
 * Results:A standard Tcl result.
 * Side effects: None.
 *----------------------------------------------------------------------*/

/* Declarations of variables */
.......

static int   AG_UpdateCmd(clientData, interp, argc, argv)
ClientData clientData;
Tcl_Interp *interp;
int argc;
char **argv;
{
/******   FILL IN CODE FOR NEW AG_Update COMMAND HERE *******/
```

```
return TCL_OK;
}
```

Figure 5.2 The implementation of a new Tcl command

Once the above code is compiled, a dynamic linked library (DLL) file is created. The DLL file is loaded into a Tcl interpreter, using the Tcl **load** command. After it is loaded into the interpreter, the **AG_Update** command can be used like any other Tcl core command.

5.3.1 New Tcl commands in the ACP

- **AG_Update**
  This command updates the agent state in the agent file stored in the agent repository. In the simplified agent structure shown in figure 4.26, the agent state is reduced to the number of machines visited by the mobile agent. AG_Update simply increases the number of machines by one.

- **AG_Search**

  AG_Search is a filtering and a retrieval command for a database or a file system. AG_Search takes a number of arguments and allows to search by keyword, authors, subject, date and media preference. AG_Search browses and retrieves multimedia documents, with several files of different media. An example on such a document, is a documentary video on the great wall of China, with english subtitles. This document could be made of two files, one being a video of binary format and the second a text file containing the english subtitles. The current prototype, allows to search by keyword a file system. The type of files supported are of HTML format. AG_Search finds and retrieves the HTML file and all the files referenced, in the HTML

file. The referenced files could be of any format supported by HTML browsers such a MPEG, GIF... After their retrieval, the files are placed in the agent repository, and become part of the agent load.

- **AG_List**

  If the user wants to query and retrieve a list of document names in a database or a file system, without retrieving the actual files, this command is used. AG_List is very similar to AG_Retrieve, except it retrieves only the document name and the name of the files that compose the document but not the actual content. It allows to list by keyword, authors, subject, date and media preference. The current prototype, allows to list by keyword a document in a file system. The type of files supported are of HTML format. AG_List finds and retrieves the HTML file name and the name of all the files referenced in the HTML file.

- **AG_Migrate**

  This command is used to send the agent from one machine to another. AG_Migrate is actually a call to the departure manager, discussed in chapter 4. AG_Migrate takes two arguments, the IP address of the remote machine and the Agent identification of the agent. The Agent identification is used by the departure manager to find the agent directory in the agent repository. Every file in the agent directory is sent to the reception manager at the remote machine according to the transport protocol discussed in chapter 4.

- **AG_Show_Search_Res**

  When the agent returns to the initial machine that constructed and sent him in the network, the

multimedia documents in the agent load, are stored in the local agent directory, and shown to the user that assigned the search mission to the agent. AG_Show_Search_Res creates a pop-up-menu with the list of documents found during the agent mission. The user can choose the documents to display. The Netscape browser is used to display the multimedia document.

- **AG_Show_List_Res**

  AG_Show_List_Res creates a pop-up-menu with the list of documents found during the AG_List agent mission. The user can look at the list of documents found, but cannot display any of them since they are not retrieved and transfered to the user workstation.

- **AG_Display**

  This command is typically used in agents with missions to send documents to other users. Once received at the destination computer, AG_Display notifies the user of the intention of the visiting agent, the name of the sender, and the names of the documents sent in a file manager similar to the file manager created by the command AG_Show_Search_Res. If the user agrees to see the documents sent to him, the netscape browser is used to display the documents.

- **AG_Clean**

  AG_Clean is a request to the AEE to delete the agent directory in the agent repository. This request is made by the agent after the completion of its execution and about to migrate to a new destination.

5.4 The GUIs and Mode of Operations

This section presents the GUIs and mode of operations for the three agent-based applications. Operations such as choose the mission, build and send the agent are presented. Also, the agent constructor is discussed and the mobile agent complete cycle, from the moment it is built to the return and the display of the results.

Figure 5.3 displays the main window of the agent system. All the buttons are functional except the "Load Profile" and "Search the Internet" buttons which are diplayed for future features. The buttons in the main window allow to set and change the agent properties, choose the mission and enter the associated parameters, build the agent, and finally send the agent to execute its mission. All the GUIs are built using Tk.

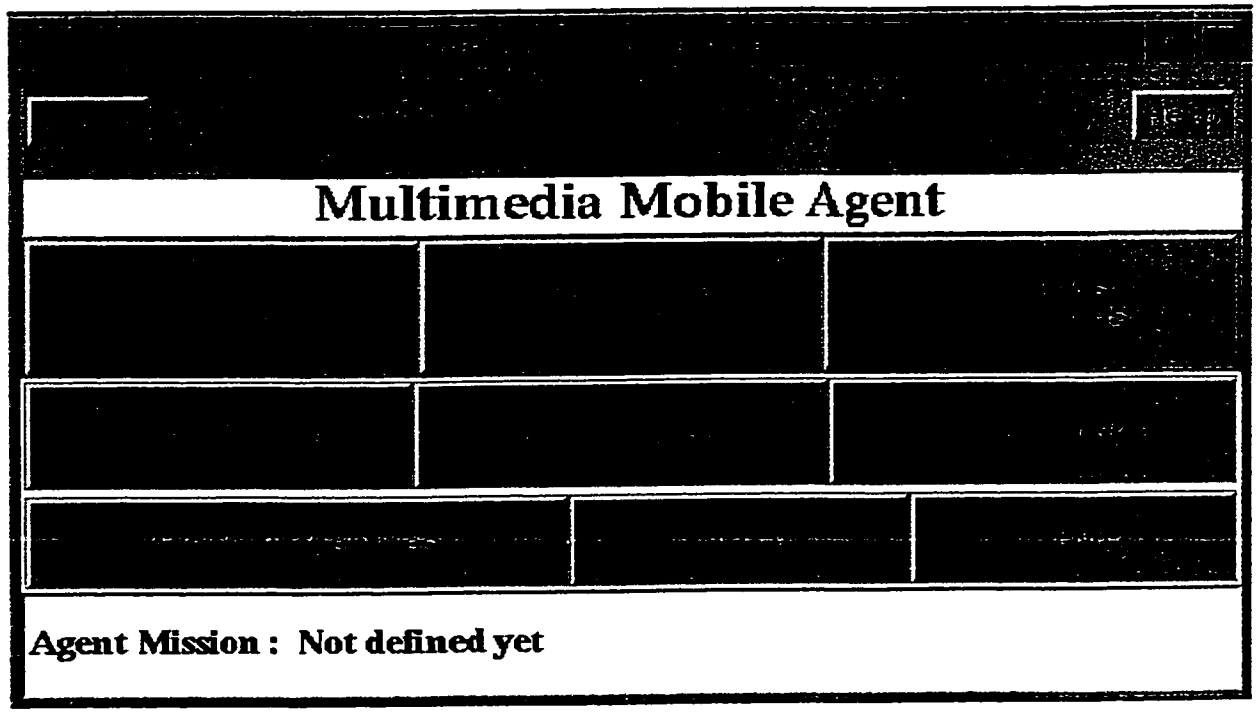

Figure 5.3 The main interface of the agent system

The bottom part of the main window illustrates the agent mission. If the user hasn't specified the mission, activating the "BUILD" or the "SEND" buttons generates the error displayed in figure 5.4.

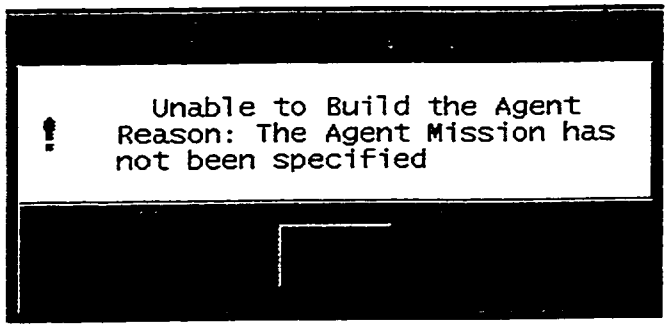

Figure 5.4 The error generated if trying to build an agent before specifying the mission

5.4.1 The Agent Properties

The "Agent Properties" button activates the window shown in figure 5.5.

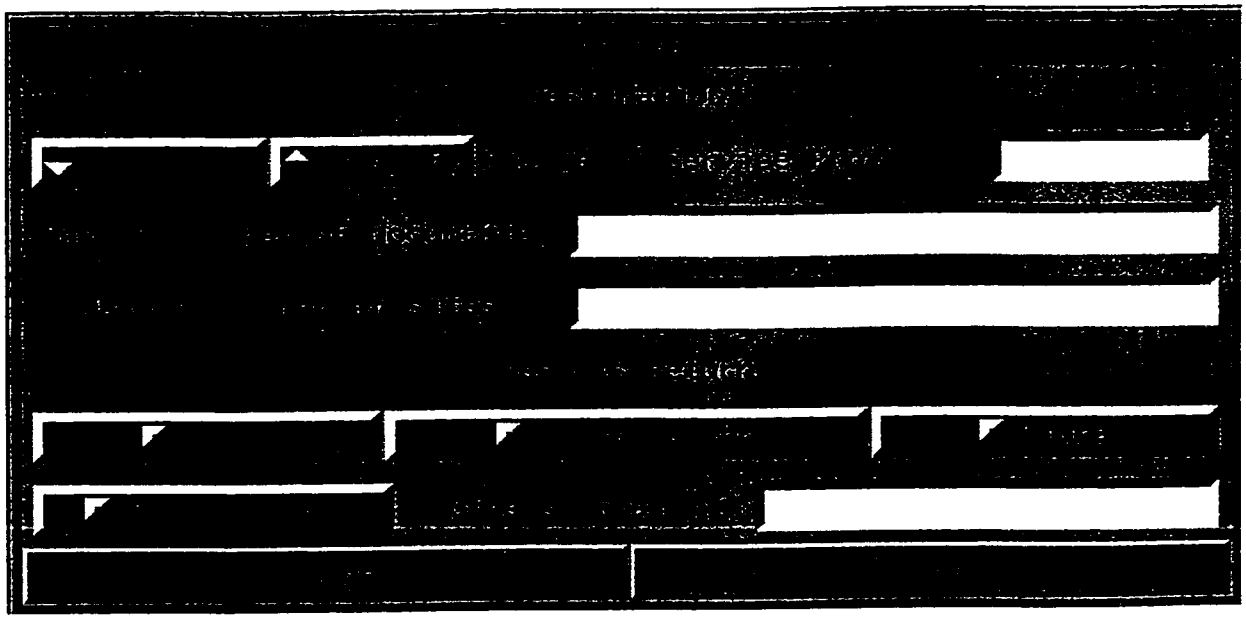

Figure 5.5 The agent properties entry window

This window allows to collect the properties that the user chooses the agent to have. The destination of the agent can be specified, or automatic. Also the maximum number of documents to collect and the maximum number of sites to visit, when the automatic destination is chosen, can be specified. Finally, the preference of the agent results presentation can be chosen: email, live presentation or storage on the hard drive.

The information collected in this form are saved in a file, when the "change" button is pressed. This file is kept and used to determine the properties of all agents, until the file is over-written by a new change of properties.

5.4.2 The Choice of an Application

Three buttons on the main interface allow the user to click on the application of his choice: the "Search the Intranet", "List Docs by Keyword" and "Send documents" buttons open each a new window to collect more information on the mission the agent is about to be assigned.

If the user clicks on "Search the Intranet", or "List Docs by Keyword" button, a similar type of window appears (Figure 5.6). Both applications require simply the keywords of the documents that the user is asking to do a list or a search for.

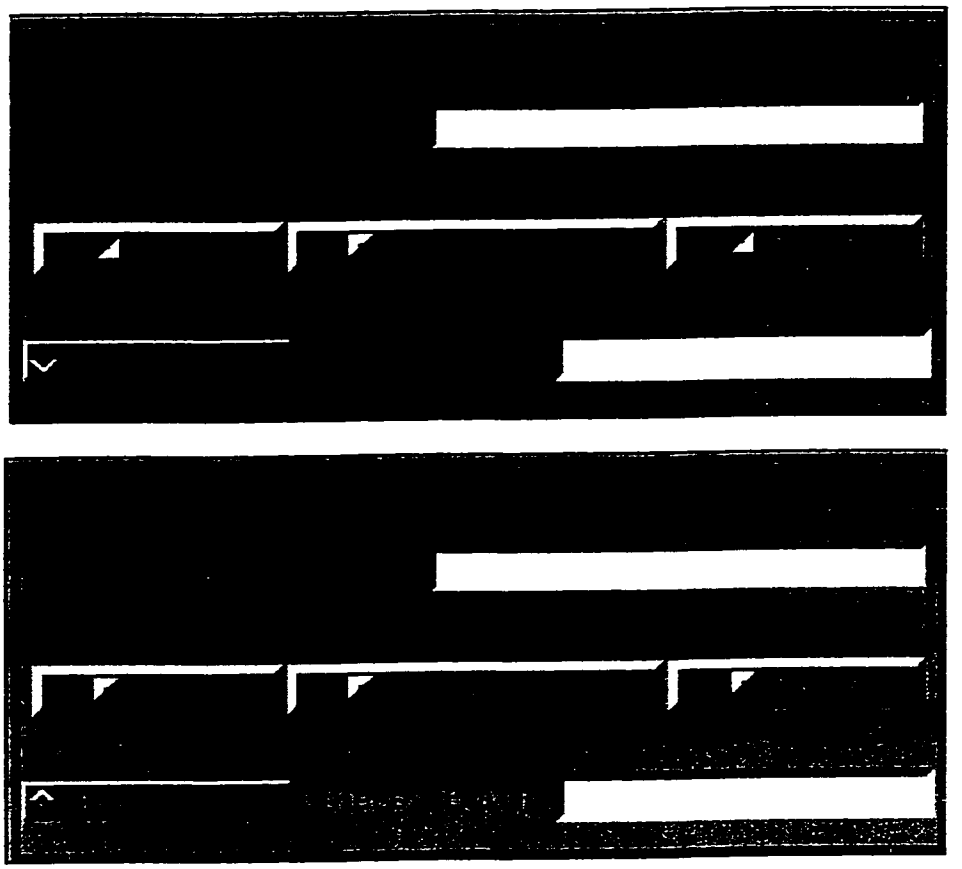

Figure 5.6 The entry windows for the List and Search applications

The user can also overwrite in the search and list entry windows the medium of delivery of the agent's mission results.

If the user clicks on the "Send Document" button, the window in figure 5.7 pops-up.

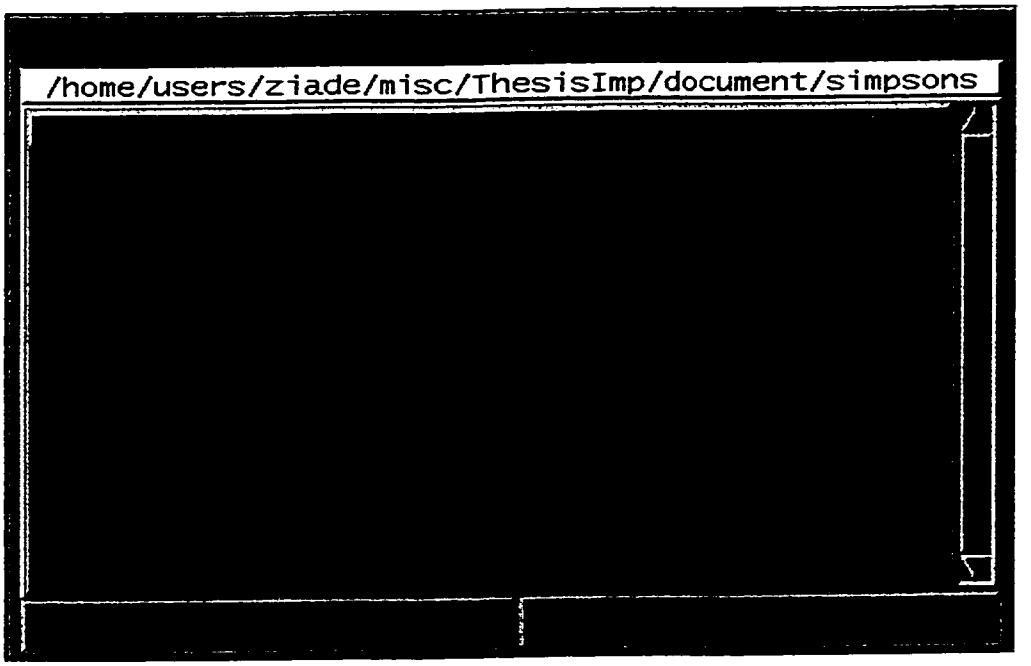

Figure 5.7 The file manager appearing when the "Send Document" is activated

The new window is a simple file manager, that permits to navigate in the user's home directory, and to choose the list of files for the mobile agent to send to the destination. Figure 5.7 shows a good example of a multimedia document on the Simpsons made of three files of various media types. Adding a file to the list is done by simply highlighting the file and pressing on the "ADD" button.

5.4.3 The Agent Constructor

After the choice of an application, the agent file is constructed by pressing on the "BUILD" button, which calls a Tcl program called the agent constructor. This program ensures that all the information required to build the agent file is available otherwise fails to build the

agent.

The information needed to build the agent can be arranged into three groups: The user info, the mission info and the agent properties. The user info and the agent properties are stored in different files in a directory that acts like the yellow pages. The mission info has to be chosen manually everytime an agent is constructed. From the main window, the user has the choice of the mission, a pop-up window allows to enter the corresponding parameters. In case of the search or list missions, the parameter is the keyword. For an agent with the send mission, the user has to choose the list of documents to send.

If the three groups are available, the agent constructor creates the agent file, according to the structure in figure 4.26. Firstly, The User information is filled by copying the info directly from the user file in the agent yellow pages. Secondly, the mission description is filled according to the entry of the user. Then, the agent properties are filled according to the file in the yellow pages (which can be changed with the window in figure 5.5). The route info is filled according to whether the user has a particular address or if the destination is automatic. if the address is specified, then its IP address is the first address in the list and the user workstation is the second address. The last address indicates to the agent the address to return the results to, unless the mission is to send documents. In this case the agent does not need to return to the sender workstation. If the destination is automatic, the agent constructor fills the list with addresses found in a file that contains information on the network in the agent yellow pages. The agent state is set to null since the agent has not started running. The last entity to build and the most important one is the agent script. The agent script is divided into sub-scripts, each intended to run on a workstation. The

number of sub-scripts always equal the number of destinations.

5.4.4 The Agent Activation, Cycle and Return of Results

The "SEND" button activates the agent after its construction. A request is then made to the migration facilitator to send the agent to its first destination. Except for the send mission, the agent entity consists in its first migration of only the agent file. The agent entity is received at the destination by the reception manager, that creates for the agent, a directory in the agent repository. After succesful migration, the AEE exctracts the script and provides an environment for the agent to run.

In case of the List or the Search missions, the script is a dialog (request for service) between the mobile agent and the service agent. At the end of the dialog, the script updates the agent state, since the sites visited has increased by one, and migrates the agent entity, to the next destination in the agent itinerary. During a search mission, the agent entity could expand, since new multimedia files can be extracted if they match the queries performed during the service dialog. Similarly to its first migration, the agent continues to migrate to all the sites in the agent itinerary, seeking service from the service agents. The last machine visited before the death of the agent is the user's workstation where the results of the mission are shown.

In case of the list mission, the list of files found are shown in a window similar to the window shown in figure 5.8. This window is created through a call to the **AG_Show_List_Res** command in the ACP. The files in the window cannot be seen because they are not extracted during the

list mission, the user is only offered a "OK" button to destroy the window.

The results of the search mission are shown in a window similar to the window in figure 5.9. The window is created by calling the command **AG_Show_Search_Res**, and resembles the window created by the list application. The exception is that the user is able to actually see the document, by selecting the file and then clicking on "open". The Netscape browser is launched to show the content of the files.

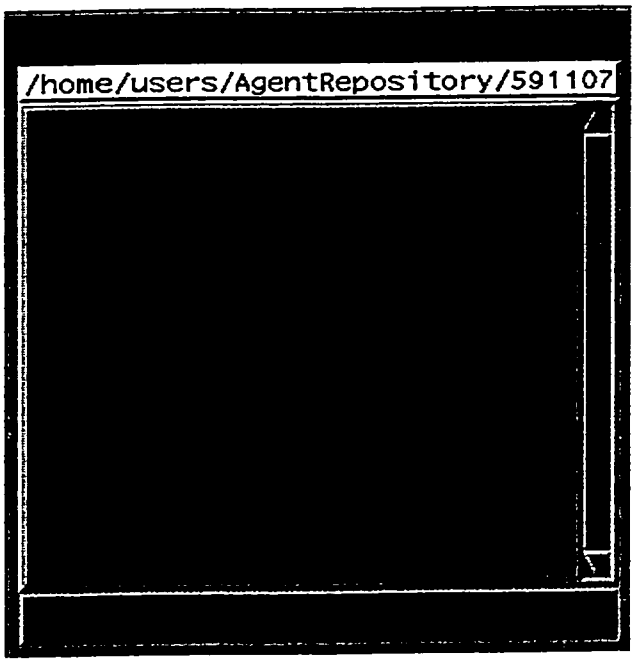

Figure 5.8 The presentation of the list of files found during the "list" mission

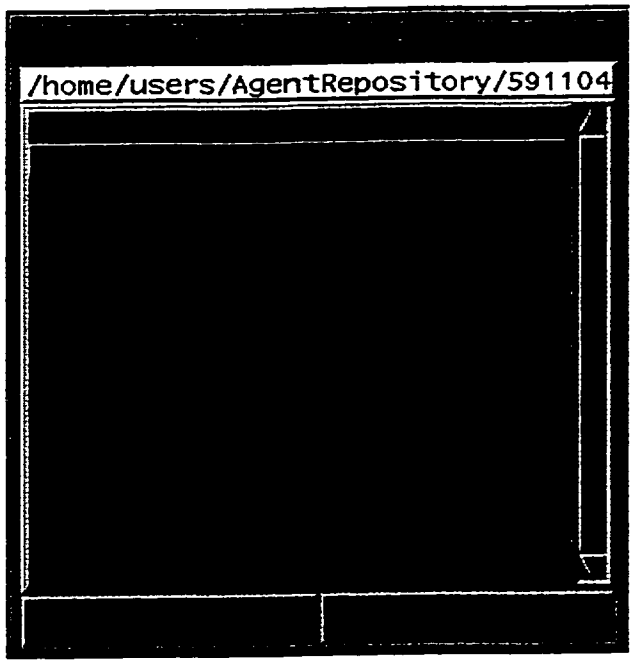

Figure 5.9 The file manager showing the results of the "Search" mission

During "send" missions, the agent entity does not vary and includes before its first migration the agent file and all the files in the agent load that are chosen by the user to be transported and displayed by the mobile agent at remote destinations. Additionally, upon receipt of the agent on a hosting machine, the agent script does not communicate with service agents, but only requests from the AEE resources to run and show the documents carried to the user.

After the receipt of the agent and the execution of the agent script, the display of the received documents is done through the **AG_Display** command, which creates the window displayed in figure 5.10. Furthermore, before waiting for the user's decision to view, save or delete the documents, the agent script updates the agent state and migrates the agent entity, to the next

destination in the list. If the user chooses to see the documents intended to him, the netscape browser is used to display the documents.

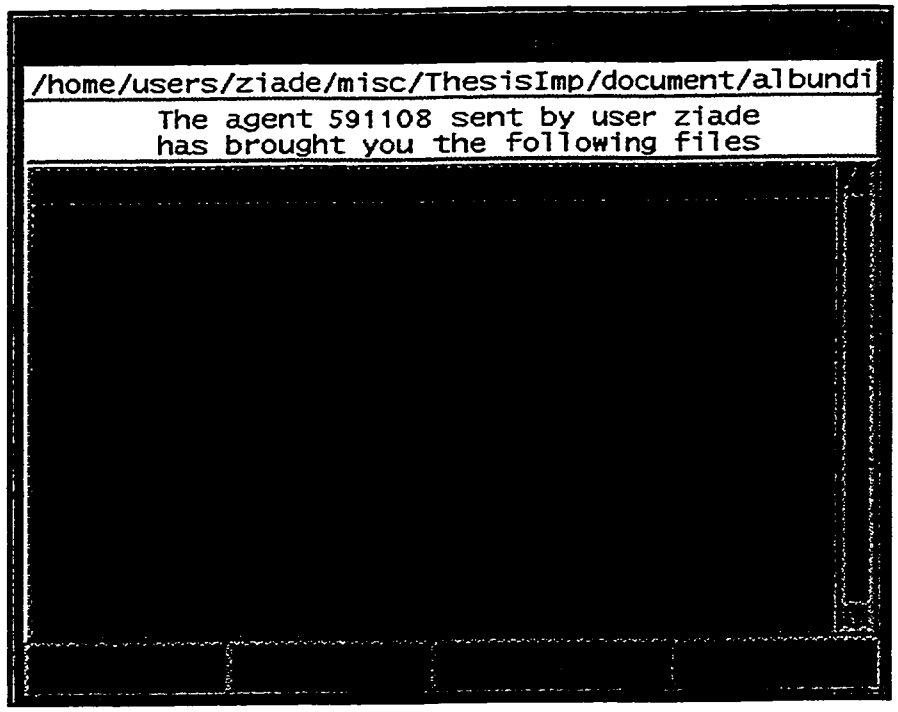

Figure 5.10 The file manager appearing during the "send" mission to remote users

Chapter 6
Conclusion

6.1 Summary

This thesis presented the design and the implementation of a multimedia transportable agent system. The goal of the system is to integrate the multimedia and the agent technologies and to address the need and roles of mobile agents in the new public and open telecommunication infrastructure. The work has been divided into two separate components, the multimedia agent platform and the multimedia agent-based applications.

The agent platform is a common base software for agent-based applications. The main

goal of the platform is to provide a transparent support for mobile agents to migrate between machines in a network, run on a remote host, and carry multimedia documents. Two main components constitute the agent platform: the migration facilitator and the AEE.

The migration facilitator is the lowest layer in the system layered architecture, built on top of TCP/IP. Two processes, written in C, deal with the migration of agents: The first process, the reception manager, constantly listens on the network for incoming agents. The agent is received along with all the multimedia files in the agent load, and stored in the agent repository. The agent is then activated by the AEE. The second process, the departure manager, listens for migration requests from active running agents. When this happens, the departing manager connects to the reception manager at the destination requested and sends the agent and the files in the load to the remote machine.

The AEE's responsability is to provide the environment to the mobile agent to run, after a successful migration. The AEE is entirely written in Tcl, and occupies the layer above the layer occupied by the migration facilitator. The AEE ensures the security of the workstation. If the permission is granted to the mobile agent to run, the AEE creates the service agent by loading the new service related tcl commands, written in C, into a Tcl interpreter and runs the mobile agent script.

An agent structure and a transport protocol has been proposed and applied in the prototype. The agent entity composed of the agent file and all the multimedia files in the agent load are encapsulated by the TH1. Then every file in the agent entity is preceeded by the TH2. The trans-

port headers are transmitted from departure managers to reception managers for a successful transfer of the mobile agent.

The agent file is formed of the agent header and the agent script. The agent header has information on the user, the mission and properties of the agent, the state of the agent, and the route of the agent. The agent file is unpacked by the AEE, which examines the agent header, and determines if the permission is granted to the agent script to execute.

Three multimedia agent-based applications are built on top of the agent platform. The applications allow to asynchronously, search, list by keyword, and send multimedia documents in the MIRLab Intranet. The mobile agents running on remote hosts in the network, communicate with service agents through a set of commands that constitute the agent communication protocol. The new commands are written in C and added to the Tcl interpreter. A GUI, built in Tk, permits to users to define the properties of their mobile agent, to determine the designated mission, build and send the agent. To build the agent, a Tcl program named the agent constructor is written. This program constructs the agent file by filling the agent header and constructing the agent script (according to the user, agent properties and mission).

6.2 Suggestions for Future Work

The work described in this thesis combined mobile agent and multimedia technologies by introducing a multimedia agent platform and agent-based applications that allow to asynchronously search, list by keyword, and send multimedia documents in a TCP/IP Intranet. This con-

vergence of technologies opens new doors for the future research. This section suggests improvements for the system design, implementation, features and capabilities.

The agent system is currently operational only in the MIRLab Intranet. Therefore, the public networks, which is the aim of the mobile agents, have not been exploited. There are two possibilities, mentioned briefly in chapter 3, section 3.6, for the agent to access the Internet. The first possibility is to install the system in various public WEB servers, and write a new service agent to access the services at these WEB servers. The Mobile agents can therefore visit the WEB servers and access the services offered by the service agents. The second possibility, which is more feasible, is to implement one Internet service agent, within the MIRLab Intranet to access the public internet. Mobile agents can then access the WEB through the Internet service agent which will translate the commands in the agent communication language into http queries.

The security issues have not been dealt with sufficiently. The mobile agent is not protected while migrating and running, and its private agenda can be violated and changed by malicious users. Furthermore, in addition to the basic steps implemented to protect machines from malicious agents, such as authentication and restriction of direct access to resources, the agents can still damage a computer by running an infinite loop which makes high usage of the computer resources. To resolve some of these security issues, mobile agents can be monitored by an external software (The AEE for instance) that restricts use of resources if some thresholds are reached. Also, mobile agents can be encrypted in migration and de-encrypted only by the AEE for the execution of the agent.

The efficiency in the transport of the multimedia files in the agent load can be somewhat improved. When the agent load becomes large in size and the agent has many migrations remaining, carrying all the files can become a substantial overhead. If the agent can in fact decide, based upon the load size and remaining route, to send the documents collected to the user workstation before continuing its journey in the network, then the overhead of carrying the multimedia documents, which uses a large bandwidth, and slows the agent in its migration, can be avoided.

The system can further be improved by inserting more intelligence. The first level of intelligence is at the interface level where the current control is totally in the hand of the user. An intelligent interface would watch the user and build a profile by representing and averaging his habits. The profile can therefore include the regular sites visited by the agent, the keyword he often looks for, the maximum credits spent...etc The profile is then used to fill up the different values in the agent header. The second level of intelligence is at the agent level. More decision making can improve the agent reliability. For example, if the mobile agent cannot migrate to a computer because it is down, or no longer in the network, an intelligent approach is to consider the following address in the agent itinerary and try to migrate to it. The third level of intelligence is at the presentation level. When the agent returns with the results, and the user is no longer logged on to his workstation, the agent should try to find the user and notify him of his return. The notification can be through a phone call, the user's pager, etc.

At the time of the defense, in March 2000, the suggestions made for the future at the time of the completion of the thesis in June 1997 are still somewhat unresolved. Presently, although the commercial and public agent systems [SIL99] [BRY99], mostly based in Java, have advanced in

many aspects, they have not yet reached maturity with all of the issues mentioned above. Ongoing research [SCH99] [KOT99] and future development of agent systems can only aid and strengthen the ideas suggested in this thesis.

Bibliography

[BON88] Bond and Gasser, "Readership in Distributed Artificial Intelligence", Morgan Kaufman, San Mateo, CA, 1998

[BRG94] Business Research Group, "The Wireless Data Market", Management Summary, Presentation and Questionnaire, A Client Study, February 1994

[BRY99] Bryce C; Vitek J, "The JavaSeal Mobile Agent Kernel" In: Proceedings of the First International Symposium on Agent Systems and Applications / Third International Symposium on Mobile Agents (ASA/MA'99), IEEE Computer Society, pp. 103-117, 1999

[DAV83] R. Davis, and R. Smith "Negotiation as a Metaphor for Distributed Problem Solving", Artificial Intelligence 20, pages 63-109, 1983

[ETZ94] O. Etzioni, and D. Weld "A Softbot-Based Interface to the Internet", Communications of the ACM, July94/Vol 37, N7.

[FAL95] B. Falchuk, and A. Karmouch "A Multimedia News Delivery System Over an ATM Network", Proc. International Conference on Multimedia Computing and Systems, pp.56-63, Washington D.C., May 1995

[FIN92] T. Finin, D. Mckay, and R. Fritzon "An Overview of KQML: A Knowledge Query and Manipulation Language", Stanford University, Computer Science Dept., March 1992

[GEO83] M. Georgeff "A Theory of Action for Multi-Agent Planning", Proceedings of the Fourth National Conference on Artificial Intelligence, Austin, Texas, San Mateo, California, pages 121-125, 1984

[HAR95] C.G. Harrison et. Al, "Mobile Agents: Are they a good idea?", IBM Research Report, RC 19887, 1995

[HEW77] C. Hewitt, "Viewing Control Structures as Patterns of Passing Messages", Artificila Intelligence, 8, No.3, pp 323-364, 1977.

[IBM95] Gilbert, Aparicio, et al. The Role of Intelligent Agents in the Information Infrastructure. IBM, United States, 1995.

[JEN93] A. Jennings, H. Higuchi, H. Liu, "A Personal News Service Based on a User Model Neural Network", Proceedings of the IJCAI-91 Workshop `Agent Modelling for Intelligent Interaction', Sydney, Australia, pp 1-33, 1993.

[KAR93] A. Karmouch "A Multimedia Information and Communication System : MEDIABASE", Proc. Multimedia Communication'93, Banff, April 1993

[KAU94] H. Kautz, B. Selman, M. Coen, and S. Ketchpel "An experiment in the design of software agents" Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI-94, Seattle, WA. 1994

[KER94] P. Kearney, A. Sehmi, and R. Smith "Emergent behavior in a multi-agent economics simulation", Cohn A G (Ed): Proceedings of the 11th European Conference on Artificial Intelligence, London, John Wiley, 1994.

[KOT99] David Kotz and Robert S. Gray, "Mobile Agents and the Future of the Internet" Department of Computer Science / Thayer School of Engineering, Dartmouth College Hanover, New Hampshire 03755 , ACM Operating Systems Review, 33(3), August 1999, pages 7-13.

[LAS94] Y. Lashkari, M. Metral, and P. Maes "Collaborative Interface Agents" Proc of the National Conference on AI, MIT Press, Cambridge, Mass. 1994

[LES81] N. Lesser, and D. Corkill "Functionality accurate, cooperative distributed systems", IEEE Transactions on Systems, Man and Cybernetics, 11 No.1, pages 81-96, 1981

[LES83] N. Lesser, and D. Corkill "The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks", AI Magazine, 4, No3, pages15-33, 1983

[MAG96] T. Magendaz, K. Rothermel, S. Krause "Intelligent Agents: An Emerging Technology for Next Generation Telecommunications ?", INFOCOM '96, March24-28 1996, San Francisco, CA, USA

[MIT94] T. Mitchell et al. "Experience with a learning Personal Assistant" Communications of the ACM, July94/Vol 37, N7

[ODO94] G. O'Donnel, "Canadian Mobile/Wireless Communications Market Report", 1994 Edition

[ORE91] T. Oren "Memex, Getting Back on the Trail. In *From Memex to Hypertext*" James Nyce and Paul Kahn,Eds. Academic Press, San Diego, CA 1991.

[OUS94] J.K. Ousterhout, (Ed), "Tcl and the Tk Toolkit", Addison-Wesley Publishing Company, Reading MA, 1994

[ROT97] K. Rothermel, R. Popescu-Zeletin (Eds), "The Architecture of the Ara Platform for Mobile Agents" ARA, Proceedings of the First International Workshop on Mobile Agents, MA'97, April 7-8th, 1997, Berlin, Germany

[SCH99] Schelderup K, Olnes J, "Mobile Agent Security-Issues and Directions" In: H.

Zuidweg, M. Campolargo, J. Delgado, A. Mullery (Eds.): Intelligence in Services and Networks. Paving the Way for an Open Service Market. Proceedings of the 6th International Conference on Intelligence and Services in Networks, IS&N'99, Barcelona, Spain, April 1999

[SIL99] L M Silva, G Soares, P Martins, V Batista and L Santos, "The Performance of Mobile Agent Platforms" In: Proceedings of the First International Symposium on Agent Systems and Applications / Third International Symposium on Mobile Agents (ASA/MA'99)

[SMI81] R. Smith, and R. Davis "Frameworks for Cooperation in Distributed Problem Solving", IEEE Transactions on Sytems, MAN and Cybernetics, Bol SMC-11, No.1, Jan 1991

[SUN95] Sun Microsystems: "The JAVA Language Environment: A White Paper", 1995 available from "http://javasoft.com/whitepaper/java-whitepaper_1.html"

[THO95] B. Thomsen, L. Leth, F. Knabe, and P.Y. Chevalier "Mobile Agents", European Computer-Industry Research Centre, External Report ECRC-95-21, 1995

[TUB94] S. Tubbs, (Ed), "A System Approach to Small Group Interaction", 2nd edition Addison-Wesley, Reading, MA, 1994.

[URL1] Web Hound Web Page URL:-http://webhound.www.media.mit.edu/projects/webhound

[WER90] K. Werkman "Knowledge-based model of negotiation using sharable perspectives", Proceedings of the 10th International Workshop on DAI, Texas, 1990.

[WHI94] J.E White, "Telescript Technology: The Foundation for the Electronic Marketplace", General Magic Inc., Moutain View, CA, 1994

[WHI96] J. E. White, "Telescript Technology: Mobile Agents" General Magic White Paper, 1996

[YAN96] T. Yan, M. Jacobsen, H. Garcia-Molina, U. Dayal, "From User Access Patterns to Dynamic Hypertext Linking", Fifth International World Wide Web Conference, May 6-10, 1996, Paris, France